Appeared in: Computers & Security 21/5 (2002) 461-471

The Open Source Approach

– Opportunities and Limitations with Respect to Security and Privacy –*

Marit Hansen[1], Kristian Köhntopp[2], Andreas Pfitzmann[3]

February 14, 2002

Abstract

Today's software often does not even fulfil basic security or privacy requirements. Some people regard the open source paradigm as *the* solution to this problem. First, we carefully explain the security and privacy aspects of open source, which in particular offer the possibility for a dramatic increase in trustworthiness for and autonomy of the user. We show which expectations for an improvement of the software trustworthiness dilemma are realistic. Finally, we describe measures necessary for developing secure and trustworthy open source systems.

Keywords: Open Source, Software Development, Security, Privacy, Trustworthiness

1 What is Open Source?

Open source software means:

- The source code is distributed along with the executable program.
- It is free to use.
- It includes a license allowing anyone to modify and redistribute the software.

1.1 Representation of Programs

Contrary to open source software, most commercial software does not allow others than the developers to view the source code. Instead only compiled object code is distributed. The *source code* is the structured and modularised representation of the software's functionality written in a programming language targeted for understanding and changeability by humans. From the source code, the *object code* is compiled by specific software, so-called compilers. The object code is machine-oriented and therefore very hard to read and understand by humans.

Example:

This simple program written in the high-level language C outputs the string "Hello world!", as might be obvious to at least every student of computer science:

---

* This paper has been presented at *ISSE 2001, September 26-28, London*. This version is slightly revised after the conference.

[1] Marit Hansen, Independent Centre for Privacy Protection, Kiel, Germany, marit.hansen@datenschutzzentrum.de.

[2] Kristian Köhntopp, Developer in the Open Source Project PHP, Kiel, Germany, kris@koehntopp.de.

[3] Andreas Pfitzmann, Dresden University of Technology, Germany, pfitza@inf.tu-dresden.de.

```
main() {
  printf("Hello world!\n");
}
```

Before a computer can execute the program, it has to be analysed and compiled to object code by a C compiler. The following object code of the above program makes use of memory addresses and direct machine commands.

```
 0:  55                    push   %ebp
 1:  89 e5                 mov    %esp,%ebp
 3:  83 ec 08              sub    $0x8,%esp
 6:  83 c4 f4              add    $0xfffffff4,%esp
 9:  68 00 00 00 00        push   $0x0
 e:  e8 fc ff ff ff        call   0xf
13:  83 c4 10              add    $0x10,%esp
16:  89 ec                 mov    %ebp,%esp
18:  5d                    pop    %ebp
19:  c3                    ret
```

The numeric expressions in the first two columns (here in hex code instead of the longer binary code) are represented by assembler expressions in the last two columns to make them more understandable to humans, while the computer only needs the information contained in the first two columns. But also other forms of representation may be used, e.g. intermediate stages during a compilation.

In contrast to the source code, the object code (at least when of a certain size) is generally ill-suited for comprehension, bugfixing, and hard to modify by humans. Though “de-compilers” can generate some source code from given object code, these de-compiled versions normally are not the same as the original code and in general barely understandable, either. E.g. de-compilers cannot provide annotations as in the original source code or variable names hinting to the underlying semantics.

1.2 Properties of Different Software Models

Today’s software models mainly differ in two aspects:

- degree of openness and
- requirements of the licensing model.

The *degree of openness* varies in different software models, *black box*, *closed box*, and *open box*:

Degree of openness	Input and output behaviour observable	Open object code	Open source code
Black box	X		
Closed box	X	x	
Open box	X	x	x

While the functionality of the inner system is unknown in a *black box*, where only the input and output behaviour can be observed, a c*losed box* system additionally provides the object code, and an *open box* system additionally provides both object and source code. Open box does not necessarily mean an openness of code for everybody, but it is a property of openness *relative* to specific parties or persons.

Often the term *closed source* is used for closed box models. In closed box models, the source code of programs as well as detailed strategies of the programs' design are treated as business secrets and are not made public to reserve any further development for the company's own programmers.

Also tools such as compilers, operating systems, or hardware used in the development of software are subject to the criteria for disclosure: Are the tools used for the generation of program code known? Are they provided with the software? If so, only as object code, or as source code? Recursively, the same questions may be applied to the tools used in the production of these tools.

The main differences in *licensing models* are shown in the following table:

License	Free of cost	Distribution without restrictions	Use without restrictions	Source code open to everybody	Source code may be modified by anybody
Commercial					
Shareware		x			
Freeware	x	x	x		
Shared source				(x)	
Open source	x	x	x	x	x

"Commercial" here indicates the type of software most prevalent now, where the buyer only acquires a personal license to use it. Other license models also permit a free distribution or an unrestricted use.[4] Microsoft's *shared source* concept opens the source code for customers and partners, but does not allow any change or distribution.[5] Open source also allows the user to edit the source code, to change it, and to distribute the changed code.

[4] Details are described by Bruno Perens*: The Open Source Definition,* in: Open Sources: Voices from the Open Source Revolution, O'Reilly, January 1999; http://www.oreilly.com/catalog/opensources/book/perens.html.
[5] http://www.microsoft.com/business/licensing/sharedsource.asp.

1.3 Open Source Definition

According to the Open Source Definition (OSD)[6], open source does not just mean access to the source code. The distribution terms of open source software must comply with the following criteria:

Open Source Definition (http://www.opensource.org/osd.html)

1. Free Redistribution

The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.

2. Source Code

The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost–preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

3. Derived Works

The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

4. Integrity of The Author's Source Code

The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

5. No Discrimination Against Persons or Groups

The license must not discriminate against any person or group of persons.

6. No Discrimination Against Fields of Endeavor

The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

7. Distribution of License

The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

[6] http://www.opensource.org/osd.html; the Open Source Definition (OSD) was developed from the guidelines for Debian GNU/Linux software.

8. License Must Not Be Specific to a Product

The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

9. License Must Not Contaminate Other Software

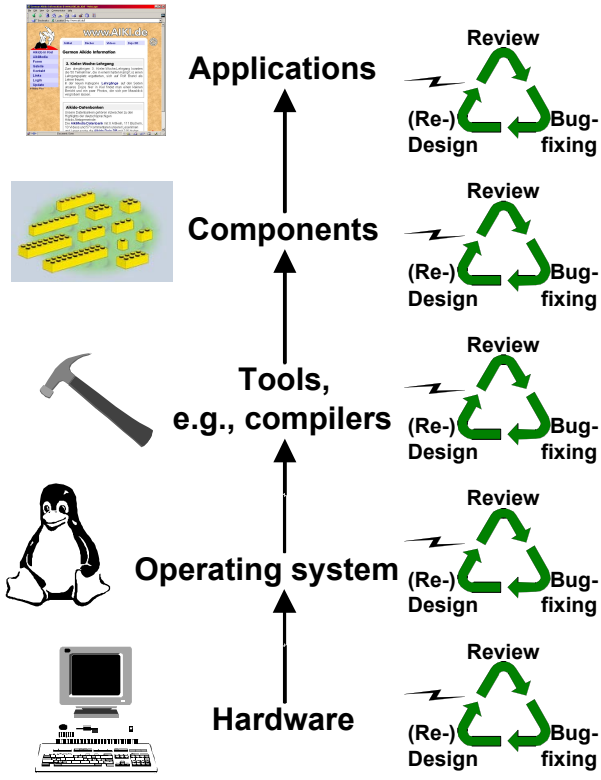
The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

For software complying with these criteria, the Open Source Initiative (OSI) introduced the seal "OSI Certified". As open source in a broader sense are counted license models differing for instance in to what extent changes made on an open source program have to comply to the same license as the original program, or to what extent restrictions on the license to use the program are allowed.

2 IT System Development

2.1 The Cycle of Software Development

The typical cycle of software development consists of three repeating phases: design, review, and bugfixing. After the system design and implementation, its behaviour is analysed and tested in a review process. Bugs and weaknesses found are eliminated by applying fixes. In many cases not only small errors have to be cured, but system design must be improved in general. The first phase, re-design, is repeated when further requirements occur, restarting the process.



The process of *design* includes a development from the idea to a written statement in form of a concept or guideline up to a realization in the form of hardware or software.

The process of *review* consists of a system's analysis, in order to determine whether it operates as desired. In a black box test, first the system's behaviour in various tests is observed from outside. In a white box test also further information on the system is included, especially the source code. Thus, experts are given a deeper insight, and the formalised methods to check the system's integrity can be applied. For instance, tests can be thus constructed, that every single execution path in the code (i.e. all multi-choice branches and loops) is being checked. Besides, it is possible to check single

modules for the defined pre- and postconditions by means of formal methods. Nowadays, this is at least possible for smaller chunks of code.

An apt modularisation combined with a respective interface specification is prerequisite for a revisable system design still manageable in its complexity. Generally, not even in white box tests the whole behaviour of the system can be evaluated. In a software review, merely the program's representation in the source code is being considered. Particularly, these inspections are only seldom applied to tools used in the process of design incl. of hardware, operating system, and compilers.

A review usually results in change requests that are realised as patches on short notice (*bugfixing*) or as part of a system re-design with a longer perspective. Patches can generally only be provided by those capable of reading and editing the source code. The change requests are passed back into the design process so they can be considered in newer versions. Thus, the feedback leads cyclic system development. The iteration frequency depends on various factors. The disclosure of the system usually shortens this cycle, for patches and feedback in this case can be performed more directly, i.e. particularly the time between review and bugfixing is reduced.

2.2 Software Development in Open Source Projects

The open source software development process in most successful projects is not anarchic, but well organised: Generally a small core team is responsible for quality control [Warf98], and each member has to prove his/her competence by delivering good code before gaining influence in the software development. The software development process of the open source model bears some resemblance to the academic research model, which is based on similar premises (e.g., reputation is being built by showing published results) and processes (e.g., peer review of published results). Technical means support the distributed co-operation, e.g. Concurrent Versioning Systems or mailing lists and data bases for bug reports.

In the context of open source projects, there are potentially many programmers working on new versions of the code that are constantly published on the Internet, to give feedback, fix bugs, or add new features [Raym00]. Simultaneous work on the code is supported by version-management tools. In certain intervals consolidated versions are released for public usage. The users may obtain the current version from the Internet or from distributors who often provide additional service.

In most cases new fellow developers do not accidentally join an existing project, but often have been using the software themselves more or less successfully for some time. Often minor bugs or deficiencies are the reason for a user to take a look at the source code and performing fixes which in turn will become part of the official source tree – if only for the reason that these fixes will not have to be manually applied for each new version.

Therefore, all well organized projects have a central repository, where the source codes and all changes are stored in form of a version history. The repository usually is maintained with the Concurrent Versioning System (CVS), also an open source project. The policies regulating access to the CVS may differ, but in many cases the CVS can be read by anyone, so that the current version developed as well as all older versions, including information on the changes made in each line, are available for any interested user of the Internet. An example:

Vers	**(User**	**Date):**	**Line**
1.495	(shane	30-Jan-99):	PHPAPI void php3_puts(const char *s)
1.207	(zeev	29-Nov-97):	{
1.232	(shane	05-Dec-97):	TLS_VARS;
1.397	(zeev	29-Mar-98):	
1.495	(shane	30-Jan-99):	#if APACHE
1.232	(shane	05-Dec-97):	if (GLOBAL(php3_rqst)) {

Displayed for each line are the file's version number where the line last was changed, the developer's name who brought in the line, and the date of change. The CVS never overwrites information. It is always possible to list every single change made between two different versions, to reconstruct versions of an older date, or to undo changes.

Changes in the repository are sent via a mailing list, to which all developers have subscribed. Generally, a comment is included that illustrates the purpose of the change. In this manner developers are kept informed of changes made on program modules, and have the possibility to discuss these changes on this mailing list or – depending on the volume of traffic – another separate list. Also changes made to the source code and possible further development directions are being discussed.

While it is comparatively easy to get read access to a project's CVS repositories, one generally only gets write access after having proven to be competent. One way of doing this is to send feedback in the form of patches to the list or a developer with write access to CVS. This patch is either accepted or rejected – in the latter case the decision is usually well-grounded. Having provided some useful patches and thus proven to the project managers sufficient involvement and interest into the project not to cause any accidental damage, a developer will be given a distinguished CVS account (a login granting an access to write to the repository). In future the developer will thus be able to commit patches directly.

The CVS repository, the mailing lists, and the web site providing downloads are tools that represent the key elements of an infrastructure able to support a community of developers. Additional services such as a database for the tracking of bug reports, or web panels to analyse CVS changes or source codes, are quite useful but less widespread. Some service providers as for instance sourceforge.net are providing these services for interested developers who do not want to run their own servers.

3 Open Source related to Security, Privacy, and Trustworthiness

When dealing with technical systems one may differentiate between:

- *security* and *privacy*, which principally can be objectively stated, and
- *trustworthiness*, which strongly depends on the subjective experience and feelings of the user.

For a wide use of IT systems in information societies there should not only be a feeling of security from the users' side, but also an actual security claim that is made subject to validation. Both is true not only for security, but also for privacy. Open source with its qualities has the capacity influence both factors, feelings and validation.

For an extensive security investigation it is necessary to analyse the whole system, including the application software, its source code, and also the tools used for the development of the object code. These tools are compilers, operating systems, hardware, and the whole development environment. The canonical example for a successful attack on the development environment is [Thom84]: a Trojan built into development tools without any traces of it in the actual source code of the application.

3.1 How Can Open Source Support Security?

By means of disclosure, in principle anyone is able to check an open source program for its specified functionality, or whether it contains any Trojans. In this way, open source enhances a software's transparency. By performing a code review, security risks can be detected, bugs can be fixed, and patches can be made available to the public via the Internet, so that the process of development is expedited. A quick response time while bugfixing – often the patch is provided at the same time as the bug is reported – contributes to a greater amount of trust from the users side. In many open source projects bug databases are run, where users can send bug reports. As soon as the problem is fixed, they will be informed immediately.

The disclosure of the source code and the criteria for its design have for a long time been considered a necessary condition for security investigations of cryptographic algorithms.[7] The fact that the concept of "security by obscurity" is very unreliable and questionable, is even today shown by various examples, where the nondisclosure of the source code could not be ensured or security-relevant bugs have been found by accident, by aimed attacks on the executable program, or by reverse-engineering [Schn99].

[7] Kerckhoff's Principle: The strength of a crypto-system should reside entirely in the difficulty in determining the key, not in the secrecy of the algorithm.

3.2 Open Source with Respect to Privacy

The trustworthiness enhanced by disclosure of the source code particularly affects privacy. While other qualities such as integrity or availability can be formulated as do's (requirements concerning actions; the software is supposed to *do* something) and be validated to some degree by practical experience, privacy requirements are very often don'ts (requirements concerning no-actions; the software is supposed *not* to do something or not to expose some information). The most prominent security goal of privacy is confidentiality, which is clearly a don't (expose information). Such requirements as well as formal proof of don'ts can only be validated by disclosure of sources.

Transparency is not only the core of open source – it is a necessary prerequisite for privacy. To be more precise: Transparency is a prerequisite for the right to informational self-determination, i.e. everyone has the right to know who is knowing what about him at what time. People can only exert control over the accuracy of their data or the use of it if they know when which data is being collected, who will have access to it and how it will be used. Privacy Acts like the European Data Protection Directive 95/46/EC and international data protection principles like the Fair Information Practices[8] address transparency of all data processing including the capture, collection, dissemination, and use of personal information. E.g. Article 12 EU Directive describes the right of each individual to obtain from the responsible party "communication to him in an intelligible form of the data undergoing processing and of any available information as to their source" and "knowledge of the logic involved in any automatic processing of data concerning him at least in the case of the automated decisions ..." Open source does not automatically fulfil the requirement of intelligible information, but it is a suitable basis for the responsible party to understand the data processing themselves in all its details.

3.3 The Status Quo of Open Source Supporting Security and Privacy

As a result of the co-operation of many interested software developers, there already exists a wide range of robust, reliable open source programs today. The concept of open source permits a customisation to personal demands if necessary, or a further development. Apart from the fact that the authors' names usually are known and they are within reach, for this reason the user is also less dependent on a single producer. Support, maintenance, and extensions can equally be provided by other parties.[9] In fact there are various companies providing hotlines and services as consultation and development for open source products. For some users the concept of open source furthermore represents an invitation to participate in the development and quality assurance of software, after first having made their contribution to review or bugfixing processes.

---

[8] Fair Information Practices, collected by the Center for Democracy and Privacy: http://www.cdt.org/privacy/guide/basic/fips.html.

[9] See Alan Cox: *The Risk of Closed Source Computing,* osOpinion: Tech Opinion commentary for the people, by the people, October 1999; http://www.osopinion.com/Opinions/AlanCox/AlanCox1.html.

The following table shows the possible benefits of open source properties for security and privacy, particularly by enhancing trustworthiness for and autonomy of the user [KöKP00]. Restrictions and limitations are illustrated as well as solutions or measures to cope with these restrictions. Thus, taking into account the listed protection measures, open source can actually improve security and especially trustworthiness. The latter is particularly important with regard to privacy.

Possible Profit for Security & Privacy: Trustworthiness	**Open Source Property**	**Restrictions and Limitations**	**Solutions and Protection Measures**
source code review possible for any independent expert	openness of source code	no guarantee that independent experts have actually fully reviewed the code and that the code is understandable	establishing methods for review and evaluation; compliance to general principles of software development
no reliance on "security by obscurity" [Schn99]	openness of source code	only works with "mature" security and privacy technologies like cryptography	avoidance of dubious and "immature" security technologies like, e.g., watermarking
no Trojans	openness of source code	only insofar as the *entire* system (including the generation of object code) is open and has been evaluated [Thom84]	review and evaluation of the entire system; open source for all tools used as well
quick elimination of bugs by distributing fixes; tailoring to meet personal demands	openness of source code; possibility for change and (re)distribution	possibility that new bugs are inadvertently built in and that internal attackers implement Trojans into their system	test before installation of fixes; encapsulation of the production version and of fixes in order to prevent unauthorised modification (e.g., by digital signatures and certificates)
user can participate in development and quality control	openness of source code; possibility for change and (re)distribution	only practicable for persons with programming knowledge	IT education campaigns
no dependence on a single manufacturer	openness of source code; possibility for change and (re)distribution	no contractual relationship, no liability and no guarantee of the developers	maintenance and distribution contracts
quality control through personal motivation of developers; reputation instead of market pressure	openness of source code; free of cost; distribution	orientation only towards personal interests of developers, e.g. often de-emphasis of soft aspects (usability) compared to hard aspects (algorithm choice)	distributors with orientation towards customers (e.g., user interfaces, support, maintenance, hotlines)
big fund of reliable/well-established code which can be re-used in new projects	distribution		

Many of the restrictions and limitations do not only apply to open source software development, but are universal requirements for any software development. However, some properties of open source like the openness and the established tradition of using digital signatures for preventing unnoticed manipulation of distributed code, definitely form an appropriate prerequisite for improving both security and trustworthiness.

3.4 Open Source = More Security?

The open source concept alone is no panacea for security aspects [Neum00]. The disclosure of the source code is only as useful as it is actually being analysed by experts who make their results public. Presently, it is widely left to chance to what extent a source code is being evaluated by the Internet community and how fast the bugs are being fixed.[10]

Thus, open source itself does not guarantee good quality and security. Moreover, the openness of the source code, and the fact that large numbers of users may look for and fix security holes can also lull people into a false sense of security [Vieg00].

Some software is called "open source" when *subsequently* disclosing the source code for everybody, e.g., Netscape's browser or StarOffice. In many cases, this software consists of megabytes of monolithic code. Even if the corresponding documentation and design principles are published, it is not very probable that other programmers will review the code or join the project. Therefore, all security considerations due to the "many-eyeballs effect" do not apply here unless professional evaluation is done [Whee01]. Because of lacking a defined review process, some open source software has contained security- or privacy-relevant bugs for many years, e.g., SSH (Secure Shell)[11] or PGP (Pretty Good Privacy)[12].

The code will not automatically improve by being published, also. Improvements are driven by the developers personal motivation in building a reputation, or if commercial software is being freed from usual marketing constraints, including that the products have to be delivered within the agreed time even before they are released by the developers. In any case, just as with any other good software development, strict standards have to be set for quality assurance, e.g. when aspects as a clear design, the modularization, the documentation, and a release only after an adequate period of testing are concerned.[13]

Besides, comfortable user interfaces play an important role in application software, especially if the users are expected to be essentially responsible for their own security and privacy. Here the open source concept is confronting diverse needs, because the developers' interests are more centered on providing functionality and not ease-of-use, which is usually considered boring and non-challenging work. Therefore, distributors and support companies have taken on the task to realise further customer-oriented developments.

---

[10] Germano Caronni who found a long unknown bug in the open source crypto-program PGP 5.0, stresses: "Public code review is a good thing – if it happens." *(Key Generation Security Flaw in PGP 5.0,* bugtraq mailing list, May 22. 2000, message ID: <20000523141323.A28431@olymp.org>).

[11] See http://www.securityfocus.com/bid/2347 (=CAN-2001-0144 at http://cve.mitre.org/).

[12] See CERT Advisories CA-2000-09 (=CVE-2000-0445) and CA-2000-18 (=CVE-2000-0678).

[13] This applies to commercial software all the same. John Pescatore, a Gartner Analyst, criticizes Microsoft's Internet Information Server (IIS): "Gartner remains concerned that viruses and worms will continue to attack IIS until Microsoft has released a completely rewritten release of ISS that is thoroughly and publicly tested. Sufficient operational testing should follow to ensure that the initial wave of security vulnerabilities every software product experiences has been uncovered and fixed." (Gartner Note "Nimda Worm Shows You Can't Always Patch Fast Enough", Note Number: FT-14-5524, September 19, 2001, http://www3.gartner.com/resources/101000/101034/101034.pdf).

If the possibility of change is given to the user in form of patches provided for him or modifications of his own, this may – by chance or launched by attackers – result in problems critical to his security. For internal attackers it is easier to build Trojans into the disclosed code before compilation. Here security measures such as an encapsulation of the product version against unauthorized changes by means of certificates that make use of digital signatures could be taken to mend these problems. In the same way it is possible to check the authenticity of provided code and patches.

A general limitation is represented by the fact that during a security investigation all systems have to be thoroughly evaluated as a whole. After all, beneath a number of applications, there are also software development tools and operating systems available as open source products. Also the concept of open hardware is being propagated. But of course, it is not easy to come to terms with the complexity of systems that are being developed and used today.

Open source does not free the users from their own responsibility for the processing of data. The license regulations often explicitly exclude cases of warranty and liability. In any case – no matter if open or closed source – it is the users who have to provide for the security of their own systems. The continuous detection of new security breaches and attacks nowadays demands a system administration that is constantly dealing with this topic, and a regular maintenance of the systems. Instant reactions are made possible by supporting alert-services.

3.5 Open Source Security & Privacy Tools

As trustworthiness plays a particularly important role for security & privacy tools, especially they are increasingly provided as open source applications,. They actually profit from the fact that many eyes can see more than just two, i.e. that independent programmers are participating in the process of review, reporting possible security breaches, and fixing bugs. As these persons make use of the Internet to spread their contributions, the supplied files incl. the source codes are provided with a digital signature by the author or a trusted third party as evidence of their integrity and authenticity. The user always should check the digital signature for its validity before compiling and installing the files.

Meanwhile, an increasing number of open source projects concentrate on security and privacy, e.g., the category “Security” of the open source platform http://sourceforge.net contains more than 500 projects, others are hosted by http://www.devshed.com or http://www.linuxsecurity.com/. Even security-oriented operating systems are being developed, e.g., Security-enhanced Linux by the NSA (http://www.nsa.gov/selinux/) and TrustedBSD (http://www.trustedbsd.org/). The open source approach is a powerful way of distributing privacy enhancing technologies (PET) as well. *“Open Privacy”* can be seen as a new way for not only distributing PET, but also striking against the principle “security by obscurity”. Additionally, transparency is a major privacy requirement for all kinds of technology and law, not only because of the Data Protection Acts (e.g., the European Data Protection

Directive Art. 12), but also because of the right to know[14], which is described in Freedom of Information Acts all over the world.

4 Conclusion

Open source is no panacea for security problems [Neum00], but can give a big boost to trustworthiness. As has been proved in many open source projects, an open and co-operative software development can lead to robust and reliable products. Nevertheless, this is no automatism, but requires adequate diligence during the entire development process and during the evaluation by experts. Security is a tough topic – the people developing and reviewing the code have to know how to write secure programs [Whee01]. The Government can help by running IT education campaigns.

It is both a national interest and the interest of each user or service provider to use trustworthy IT systems.[15] Without control over the IT systems, nobody can take responsibility for data processing in his or her IT system. In the context of a growing need for PKI and other trusted third parties, the fact that these parties could not sincerely provide for security of their IT systems, is terrifying. The same accounts for all e-commerce or e-government applications. Remedy can be achieved by building provably trustworthy hardware and software systems. No single state can finance this development alone. The only chance is to use existing open source systems, enhance them, and provide a defined way of evaluation, review, and distribution of patches. As a matter of fact, open source makes it possible for competing nations with diverging national interests to work together and build a system that can be trustworthy to all of them, whereas it would be unacceptable to use closed source software from the respective other party.

5 References

[KöKP00] Köhntopp, Kristian/Köhntopp, Marit/Pfitzmann, Andreas*: Sicherheit durch Open Source? Chancen und Grenzen,* in: DuD 24/9 (2000), Vieweg, Wiesbaden 2000, pp. 508-513; http://www.koehntopp.de/marit/pub/opensource/

[Neum00] Neumann, Peter G.: *Robust Nonproprietary Software,* IEEE Symposium on Security and Privacy, Oakland CA May 15-17, 2000; http://www.csl.sri.com/neumann/ieee00+.pdf

[Raym00] Raymond, Eric S.: *The Cathedral and the Bazaar,* Version 1.51, August 2000; http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/

[Schn99] Schneier, Bruce: *Crypto-Gram September 15, 1999;* http://www.counterpane.com/crypto-gram-9909.html#OpenSourceandSecurity

[14] Also called “right to access“.

[15] From the European Parliament Resolution on ECHELON, Minutes of September 5, 2001: “Measures to encourage self-protection by citizens and firms
[...] 29. Urges the Commission and Member States to devise appropriate measures to promote, develop and manufacture European encryption technology and software and above all to support projects aimed at developing user-friendly open-source encryption software;
30. Calls on the Commission and Member States to promote software projects whose source text is made public (open-source software), as this is the only way of guaranteeing that no backdoors are built into programmes;
31. Calls on the Commission to lay down a standard for the level of security of e-mail software packages, placing those packages whose source code has not been made public in the "least reliable" category [...]”
(http://www2.europarl.eu.int/omk/OM-Europarl?PROG=REPORT&L=EN&PUBREF=-//EP//TEXT+REPORT+A5-2001-0264+0+NOT+SGML+V0//EN, mirrored at http://cryptome.org/echelon-090501.htm#Minutes).

[Thom84] Thompson, Ken: *Reflections on Trusting Trust,* Turing Award Lecture, Communications of the ACM, Vol. 27, No. 8, August 1984, pp. 761-763; http://www.acm.org/classics/sep95/

[Vieg00] Viega, John: *The Myth of Open Source Security*, May 26, 2000; http://webdeveloper.earthweb.com/websecu/article/0,,12013_621851,00.html

[Warf98] Warfield, Michael H.: *Musings on Open Source Security Models – Does Open Source Mean Open Season for Crackers?,* 1998; http://www.linuxworld.com/linuxworld/lw-1998-11/lw-11-ramparts.html

[Whee01] Wheeler, David A.: Secure Programming for Linux and Unix HOWTO – Chapter 2.3: Is Open Source Good for Security?, 1999-2001, http://www.linuxdoc.org/HOWTO/Secure-Programs-HOWTO/open-source-security.html

Contact address of the corresponding author:
Marit Hansen
Independent Centre for Privacy Protection Schleswig-Holstein
Unabhängiges Landeszentrum für Datenschutz Schleswig-Holstein
Holstenstraße 98
D-24103 Kiel
Germany
Phone: +49 431 988 1214
Fax: +49 431 988 1223
E-Mail: marit.hansen@datenschutzzentrum.de

Biographies

Kristian Köhntopp, computer scientist, is a senior security consultant. For more information see his homepage: http://www.koehntopp.de/kris/.

Marit Hansen, computer scientist, is head of the “Privacy Enhancing Technologies (PET)” Section at the Independent Centre for Privacy Protection Schleswig-Holstein (the state privacy commission), Germany. Since her diploma in 1995 she has been working on security and privacy aspects especially concerning the Internet, anonymity, pseudonymity, identity management, biometrics, multilateral security, and e-privacy from both the technical and the legal perspectives. In several projects she and her team actively participate in technology design in order to support PET and give feedback to the legislation. They co-operate with various project partners, especially Dresden University of Technology and many Privacy Commissioners, e.g., via the Virtual Privacy Office (http://www.privacyservice.org/). She chairs the Working Group on Dependable IT Systems of the German Society for Computer Science (GI) (http://www.gi-ev.de/) and is member of W3C’s P3P (Platform for Privacy Preferences) Working Groups.

Andreas Pfitzmann is a professor of computer science at Dresden University of Technology. His research interests include privacy and multilateral security, mainly in communication networks, mobile computing, and distributed applications. He has authored or coauthored about 70 papers in these fields. He received diploma and doctoral degrees in computer science from the University of Karlsruhe. He is a member of ACM, IEEE, and GI, where he served as chairman of the Special Interest Group on Dependable IT-Systems for ten years.