Stand: 2000-07-06. Kristian Köhntopp, Marit Köhntopp, Andreas Pfitzmann: Sicherheit durch Open Source? Chancen und Grenzen; eingereicht für DuD 9/2000.

# Sicherheit durch Open Source? Chancen und Grenzen

*Kristian Köhntopp, Marit Köhntopp, Andreas Pfitzmann*

*Es ist bekannt, dass die Sicherheit von gängiger Software heutzutage zu wünschen übrig lässt. An Open-Source-Software knüpfen sich große Hoffnungen auf Besserung dieser Situation. Die Autoren erläutern, welche dieser Hoffnungen realistisch sind.*

## Einleitung

Open Source ist in: Diese Methode der Software-Entwicklung wird heutzutage als *die* Lösung für mehr Sicherheit in der Informationstechnik angepriesen – einer der Gründe, weshalb Regierungen verschiedener Länder beschlossen haben, Open-Source-Projekte zu fördern oder sogar im öffentlichen Bereich den Einsatz solcher Software vorzuschreiben.

Dieser Beitrag untersucht, inwieweit Sicherheit tatsächlich durch den Open-Source-Entwicklungsmechanismus gefördert werden kann und welche zusätzlichen Anforderungen für dieses Ziel zu erfüllen sind.

## 1 Begriffe rund um Open Source

Unter Open Source („offene Quelle“) versteht man Software, deren Quellcode offengelegt und für jeden frei verfügbar ist. Um zu verstehen, warum dies etwas Besonderes ist und was es bedeutet, muss man sich zunächst die einzelnen Schritte der Programmentwicklung bewusst machen.

### 1.1 Repräsentationen von Programmen

Computerprogramme sind Folgen von Anweisungen in einer Programmiersprache. Ein Rechner versteht unmittelbar nur die hardwarenahe Maschinensprache (also eine Repräsentation von Einsen und Nullen) und kann daher nur solche Programme direkt ausführen. Eine Programmierung in Maschinensprache ist für Menschen sehr unkomfortabel und fehlerträchtig. Deswegen wird meist eine höhere Programmiersprache verwendet. Damit ein Computer Programme, die in solchen Sprachen geschrieben werden, ausführen kann, müssen sie mithilfe spezieller Software, einem Interpreter oder einem Compiler, in Maschinensprache übersetzt werden. Während Interpreter die einzelnen Programmbefehle während jeder Ausführung des Programms in für den Computer verständliche Maschinensprache überführen, setzen Compiler den Programmtext (Quellcode, Sourcecode, Source) vollständig in Maschinencode um und erzeugen dadurch ein ausführbares Programm (Binärcode, Objektcode, Binaries).

**Beispiel:**
Dieses einfache Programm in der höheren Programmiersprache C gibt die Zeichenkette „Hallo, Welt!“ aus, wie dies mindestens jedem Informatik-Studenten sofort ersichtlich ist:

```
main() {
  printf("Hallo, Welt!\n");
}
```

Damit ein Computer das Programm ausführen kann, muss es vom C-Compiler analysiert und in Maschinensprache umgesetzt werden. In der folgenden Darstellung des Objektcodes vom obigen Programm wird mit Speicheradressen und maschinennahen Befehlen hantiert:

```
 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
```

Die letzten beiden Spalten sind eine Assembler-Repräsentation der links stehenden Zahlenkolonnen und dienen einem menschlichen Betrachter zur besseren Verständlichkeit; für den Rechner reichen die ersten beiden Spalten aus (hier in der kürzeren Hexadezimal- statt Binärschreibweise). Daneben sind weitere Repräsentationen gängig, z. B. als Zwischenstufen bei einem Compilerlauf.

Generell gilt, dass im Gegensatz zum Quellcode der Objektcode – spätestens ab einer gewissen Größe – nicht dazu geeignet ist, dass Menschen anhand dessen das Programm verstehen, auf Fehler überprüfen oder ändern können. Zwar gibt es Tools zum Decompilieren (Rückübersetzen) von Objektcode in vorgegebene Programmiersprachen. Weil es sich beim Objektcode um eine durch den Compiler optimierte Fassung handelt, bei der beispielsweise sprechende Variablennamen und Anmerkungen im Programm zur Dokumentation, da für den Rechner irrelevant, beseitigt werden, stehen diese Informationen in einer rückübersetzten Fassung nicht zur Verfügung. Damit stimmt das Resultat in der Regel nicht mit dem Ursprungsprogramm überein und ist auch sehr viel weniger von Menschen verstehbar.

## 1.2 Eigenschaften verschiedener Software-Modelle

Die heutigen Software-Modelle unterscheiden sich hauptsächlich in zwei Punkten:

- im Grad der Offenlegung der Systeme und
- in den Anforderungen der jeweiligen Lizenz.

Der *Grad der Offenlegung* lässt sich mit den Begriffen Black Box, Closed Box und Open Box beschreiben:

| Grad der Offenlegung | Ein- und Ausgabeverhalten beobachtbar | Objektcode offen | Quellcode offen |
|---|---|---|---|
| Black Box | x | | |
| Closed Box | x | x | |
| Open Box | x | x | x |

Tab. 1: Grad der Offenlegung

Während bei einer *Black Box* nichts über die innere Funktion des Systems bekannt ist und nur das Ein- und Ausgabeverhalten von außen beobachtet werden kann, liegen bei einem *Closed-Box-System* zusätzlich der Objektcode und bei einem *Open-Box-System* nochmals zusätzlich der Quellcode vor. Für diejenigen, denen der Quellcode eines Programms offen vorliegt, handelt es sich also insoweit um eine Open-Box-Software.

Statt Closed Box verwendet man häufig den Begriff Closed Source: Dabei werden der Quellcode von Programmen sowie die detaillierten Designkriterien als Unternehmensgeheimnis unter Verschluss gehalten, um eine Weiterentwicklung den eigenen Programmierern vorzubehalten.

Auch für die Werkzeuge, die zur Erstellung einer Software dienen wie beispielsweise Compiler, Betriebssysteme oder die Hardware, ist das Kriterium der Offenlegung anzulegen: Sind die Werkzeuge für die Codeerstellung bekannt? Liegen sie vor? Wenn ja, lediglich im Objekt- oder auch im Quellcode? Dies lässt sich rekursiv wiederum für die Tools, die zur Erstellung dieser Werkzeuge dienen, anwenden.

Neben einem unterschiedlichen Grad an Offenlegung gibt es *lizenzbezogene Eigenschaften*, die typisch für bestimmte Software-Lizenzmodelle sind:

| Lizenz | Unentgeltlich | Frei weitergebbar | Uneingeschränkt nutzbar | Quellcode für alle offen | Quellcode änderbar |
|---|---|---|---|---|---|
| Commercial | | | | | |
| Shareware | | x | | | |
| Freeware | x | x | x | | |
| Open Source | x | x | x | x | x |

Tab. 2: Verschiedene Lizenzmodelle

„Commercial“ bezeichnet hier die Art der zurzeit am meisten verkauften Software, bei der der Käufer lediglich eine Lizenz für die persönliche Nutzung erwirbt. Andere Lizenzmodelle erlauben darüber hinaus das freie Weitergeben oder die uneingeschränkte Nutzung.[1] Open Source ermöglicht weiterhin das Arbeiten mit dem Quellcode sowie die Änderung des Quellcodes und die Verteilung solcher Änderungen.[2]

### 1.3 Die Open Source Definition

Die Open Source Definition (OSD)[3], die aus Richtlinien für die Software Debian GNU/Linux hervorgegangen ist, sieht für Open-Source-Software die folgenden Eigenschaften vor:

1. *Freie Weiterverbreitung*

Die Lizenz darf niemanden im Verkauf oder in der Weitergabe der Software als Teil einer aus verschiedenen Quellen zusammengesetzten Software-Distribution einschränken. Die Lizenz darf keinerlei Lizenzgebühren oder andersartige Beiträge verlangen.

[1] Details dazu beschreibt Bruno Perens*: The Open Source Definition,* in: Open Sources: Voices from the Open Source Revolution, O’Reilly, Januar 1999; http://www.oreilly.com/catalog/opensources/book/perens.html.

[2] Das Urheberrecht bleibt übrigens durch diese gängigen Lizenzmodelle unberührt, selbst wenn die Autoren ihren Nutzern ebenfalls Änderungs- und Verbreitungsrechte einräumen.

[3] http://www.opensource.org/osd.html; deutsche Übersetzung in Martin Müller: *Open Source – eine Standortbestimmung*, in: Open Source – kurz & gut, O’Reilly, Mai 1999; http://www.oreilly.de/german/freebooks/os_tb/os_tb_1.htm.

2. *Quellcode*
Das Programm muss den Quellcode beinhalten und sowohl die Verbreitung als Quellcode als auch in kompilierter Form gestatten. Wird ein Teil des Produkts nicht mit Quellcode verbreitet, so muss auf eine Möglichkeit, den Quellcode gebührenfrei aus dem Internet downzuloaden, ausdrücklich hingewiesen werden. Der Quellcode muss in einer Form zur Verfügung gestellt werden, in der ein Programmierer ihn verändern kann. Absichtlich verwirrend geschriebener Quellcode ist nicht erlaubt. Ebenso sind Zwischenformen, wie die Ausgabe eines Präprozessors oder eines Übersetzers, als Ersatz für Quellcode verboten.

3. *Auf dem Programm basierende Werke*
Die Lizenz muss die Veränderung des Programms, auf dem Programm basierende Werke sowie deren Verbreitung unter den gleichen Lizenzbedingungen gestatten.

4. *Unversehrtheit des Originalcodes*
Die Lizenz darf die Verbreitung von modifiziertem Quellcode nur dann einschränken, wenn sie die Verbreitung von so genannten Patch-Dateien in Verbindung mit dem Originalcode gestattet, damit das Programm vor der Benutzung verändert werden kann. Die Lizenz muss ausdrücklich die Verbreitung von Software erlauben, die mit verändertem Quellcode erstellt wurde. Die Lizenz darf allerdings von auf dem Programm basierenden Werken verlangen, einen von der Originalsoftware verschiedenen Namen oder eine andere Versionsnummer zu tragen.

5. *Keine Diskriminierung von einzelnen Personen oder Gruppen*
Die Lizenz darf keinerlei Personen oder Personengruppen diskriminieren, d. h. jeder muss das gleiche Recht haben, zu Open-Source-Software beizutragen.

6. *Keine Einschränkungen für bestimmte Anwendungsbereiche*
Die Lizenz darf niemanden in der Benutzung des Programms in einem bestimmten Einsatzgebiet einschränken. Sie darf beispielsweise nicht die kommerzielle Nutzung oder die Benutzung in der Genforschung verbieten.

7. *Verbreitung der Lizenz*
Die zum Programm gehörigen Rechte müssen für jeden gelten, der das Programm erhalten hat, ohne dass eine weitere Lizenz beachtet werden muss.

8. *Die Lizenz darf nicht für ein bestimmtes Produkt gelten*
Die zum Programm gehörigen Rechte dürfen nicht davon abhängen, dass das Programm Teil einer bestimmten Software-Distribution ist. Wird das Programm außerhalb einer solchen Distribution genutzt oder verbreitet, so gelten für den Benutzer dieselben Rechte, die in der Original-Distribution gewährt werden.

9. *Die Lizenz darf andere Software nicht beeinträchtigen*
Die Lizenz darf die Verbreitung anderer Software zusammen mit der lizenzierten Software nicht einschränken.

Für Software, die diese Kriterien erfüllt, hat die Open Source Initiative (OSI) das Kennzeichen „OSI Certified“ eingeführt. Zu Open Source im weiteren Sinne werden verschiedene Lizenzmodelle gerechnet, die sich beispielsweise darin unterscheiden, inwieweit eigene Veränderungen an einem Open-Source-Programm wiederum unter dieselbe Lizenz fallen müssen wie das Originalprogramm oder Einschränkungen der Nutzungslizenz erlaubt sind.

# 2 Der Entwicklungsprozess von IT-Systemen

## 2.1 Der Entwicklungszyklus

Ein typischer Entwicklungszyklus für IT-Systeme besteht aus drei immer wiederkehrenden Teilen:

- Design,
- Review und
- Patch.

Das *Design* umfasst den Entwurf des jeweiligen Systems von der Idee über die schriftliche Fixierung in einem Konzept oder Pflichtenheft bis zur Realisierung in Hard- und/oder Software.

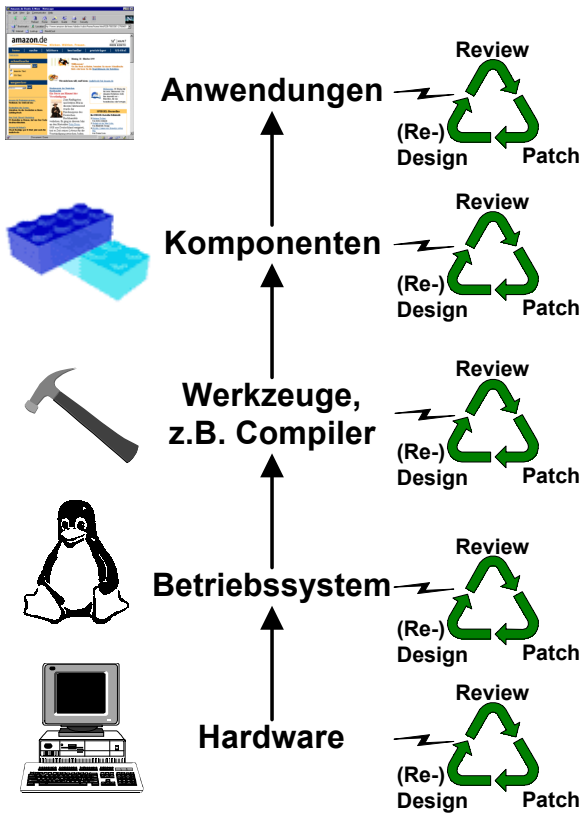


Abb. 1: Systementwicklung

Der *Review-Prozess* besteht in einer Analyse des Systems, ob es genau so wie gewünscht funktioniert. In einem Black-Box-Test wird lediglich das Verhalten des Systems für bestimmte Testfälle von außen beobachtet. Bei einem White-Box-Test stehen weitere Informationen über das System zur Verfügung, insbesondere der Quellcode.[4] Dies ermöglicht Fachleuten einen tieferen Einblick und die Anwendung von formalisierten Verfahren zum Nachweis der Korrektheit. Beispielsweise können Testfälle so konstruiert werden, dass jedes

[4] White-Box-Tests richten sich hauptsächlich an Open-Box-Systeme. Es ist aber möglich, Teile davon lediglich anhand des Objektcodes, also in Closed-Box-Systemen, durchzuführen. Fred Cohen weist in *Protection Testing* (Software Development Magazine, September 1998, http://www.sdmagazine.com/supplement/ss/features/s9809f1.shtml) darauf hin, dass bei White-Box-Tests normalerweise nur der Quellcode von einem Stück Software, nicht jedoch vom gesamten System einschließlich der Details zum Hardware-Entwurf zur Verfügung steht, wie es wünschenswert wäre.

Stück des Codes (d. h. alle Fallunterscheidungen und Schleifen) durchlaufen wird.[5] Außerdem lassen sich für die einzelnen Module definierte Vor- und Nachbedingungen durch formale Methoden beweisen – heutzutage ist dies zumindest für kleine Codeteile realistisch.

Eine geeignete Modularisierung, verbunden mit einer entsprechenden Schnittstellen-spezifikation, ist Voraussetzung für einen revisionsfähigen Systementwurf, der in seiner Komplexität noch handhabbar ist. Auch bei White-Box-Tests kann meist nicht das vollständige System evaluiert werden. Bei einem Software-Review wird stattdessen lediglich eine Repräsentation des Programms im Quellcode betrachtet. Insbesondere erstrecken sich die Untersuchungen nur selten auf die Werkzeuge, die im Prozess des Designs einschließlich der Hardware, dem Betriebssystem und Compilern eingesetzt werden.

Aus dem Review ergeben sich in der Regel Änderungswünsche, die kurzfristig als Patch („Flicken“, automatisch verarbeitbare Änderungsanweisungen zur Fehlerkorrektur) oder langfristig in einem Re-Design des Systems umgesetzt werden. Patches lassen sich im Allgemeinen nur von denjenigen bereitstellen, die den Quellcode lesen und ändern können.[6] Die Änderungswünsche werden zurück in den Entwurfsprozess gegeben, damit sie bei neuen Versionen Berücksichtigung finden. So führt diese Feedback-Schleife zu einer Spirale des System-Entwicklungsprozesses. Wie lange dieser Kreislauf dauert, hängt von vielen Faktoren ab. Die Offenlegung des Systems verkürzt ihn in der Regel, da dann Fehlerbehebungen direkt durchgeführt und rückgemeldet werden können, d. h. insbesondere die Zeitspanne zwischen Review und Patch reduziert wird.

## 2.2 Software-Entwicklung in Open-Source-Projekten

In Open-Source-Projekten arbeiten potenziell viele Programmierer weltweit zusammen an den ständig über das Internet publizierten neuen Fassungen des Codes, um Rückmeldungen zu geben, Fehler zu beseitigen oder neue Funktionen einzubringen [Raym_99]. Das gleichzeitige Bearbeiten des Codes wird durch Versionsverwaltungs-Tools unterstützt. In gewissen Abständen werden konsolidierte Versionen zum allgemeinen Einsatz freigegeben. Die Nutzer beziehen die jeweils aktuelle Version aus dem Internet oder von Distributoren, die häufig noch zusätzliche Dienstleistungen anbieten.

In den meisten Fällen kommen neue Mitentwickler nicht plötzlich und nicht ohne Grund zu einem existierenden Projekt hinzu, sondern sie sind Nutzer der Software und setzen diese bereits einige Zeit beinahe erfolgreich ein. Oft sind es einige kleine Unzulänglichkeiten oder Fehler, die einen dieser Nutzer dazu getrieben haben, sich den Quellcode anzusehen und Fehlerkorrekturen vorzunehmen, die nun Bestandteil des offiziellen Sourcenbaums werden sollen – schon alleine damit diese Änderungen nicht in jeder neuen Version von Hand selbst nachgepflegt werden müssen.

Alle gut organisierten Projekte haben zu diesem Zweck ein zentrales Repository, in dem die Quellcodes und alle Änderungen darin in einer Versionshistorie aufbewahrt werden. Das Repository wird in der Regel mit dem Concurrent Versioning System (CVS), ebenfalls ein Open-Source-Projekt, gepflegt. Die Policies, die den Zugang zum CVS regeln, können unterschiedlich sein, aber sehr häufig ist das CVS für jeden lesbar, sodass die aktuelle Entwicklungsversion sowie sämtliche ältere Versionen mit zeilengenauen Änderungs-informationen für jeden interessierten Internet-Nutzer lesbar sind. Ein Beispiel:

[5] Anderenfalls kann toter Code entfernt werden.
[6] Dies ist auch bei Hardware denkbar. Nur ist die Verbreitung solcher Patches nicht so einfach möglich wie bei Software.

```
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)) {
```

Erkennbar ist für jede einzelne Zeile die Versionsnummer der Datei, in der die Zeile zuletzt geändert worden ist. Außerdem werden der Name des Entwicklers, der die Zeile eingespielt hat, und das Datum der Änderung angezeigt. CVS überschreibt niemals Informationen: Es ist jederzeit möglich, jede einzelne Änderung zwischen zwei beliebigen Dateiversionen auflisten zu lassen, komplette ältere Stände zu rekonstruieren oder Änderungen rückgängig zu machen.

Änderungen am Repository werden mit dem Kommentar, der zu jeder Änderung mitgeliefert werden muss, über eine Mailinglist versendet, auf der alle Entwickler eingeschrieben sind. Sie werden auf diese Weise über Änderungen an Programmmodulen informiert und haben die Möglichkeit, diese Änderungen zu diskutieren. Über diese Mailinglist oder – je nach Verkehrsaufkommen – eine gesonderte Liste werden auch Änderungen am Quellcode koordiniert und mögliche Weiterentwicklungen diskutiert.

Während ein Lesezugriff zu CVS-Repositories eines Projektes in der Regel leicht zu bekommen ist, bekommt man Schreibzugriff im Allgemeinen nur, nachdem man seine Kompetenz bewiesen hat. Dies kann man tun, indem man Verbesserungsvorschläge in Form von Patches an die Liste oder einen Entwickler mit CVS-Schreibzugriff sendet. Dieser Patch wird entweder akzeptiert oder – dann meist begründet – abgelehnt. Hat ein Entwickler den Projektleitern durch einige sinnvolle Patches bewiesen, dass er sich mit dem Projekt ausreichend auskennt, um nicht versehentlich Schaden anzurichten, und dass er an einer Weiterentwicklung interessiert ist, wird ihm ein eigener CVS-Account (ein Login, das Schreibzugriff auf das Repository erlaubt) zugeteilt, und er kann in Zukunft seine Patches direkt committen (in das Repository einspielen).

Die drei Instrumente CVS-Repository, Mailinglists und Website mit Download-Gelegenheit sind die zentralen Bausteine der Infrastruktur, die eine Entwickler-Community tragen kann. Weitere Dienste, etwa eine Datenbank für das Tracking (Nachvollziehen) von Bugreports oder Weboberflächen zum Analysieren von CVS-Änderungen und von Quellcodes, sind nützlich, aber weniger weit verbreitet. Einige Dienstleister wie zum Beispiel sourceforge.net oder das Open-Source-Zentrum, das auf Initiative des BMWi in Deutschland begründet werden soll, stellen diese Dienste interessierten Entwicklern zur Verfügung, falls diese ihre Server nicht selbst betreiben wollen.

## 3 Open Source und Sicherheit

Bei der Betrachtung technischer Systeme kann man unterscheiden zwischen:

- *Sicherheit*, die im Prinzip objektiv feststellbar ist, und
- *Vertrauenswürdigkeit*, die stark subjektiv, d. h. vom Beurteiler, seiner Erfahrung und seinen Gefühlen abhängig ist.

Für einen breiten Einsatz von IT-Systemen in der Wissensgesellschaft sollte nicht nur ein Gefühl der Sicherheit bei den Nutzern, sondern auch tatsächliche, validierbare Sicherheit gegeben sein. Open Source und seine Eigenschaften können beides beeinflussen.

Für eine umfassende Sicherheitsuntersuchung ist es erforderlich, das Gesamtsystem zu analysieren, d. h. außer der Anwendungssoftware einschließlich ihres Quellcodes auch die Werkzeuge, die zur Erstellung des Objektcodes verwendet werden, wie Compiler, Betriebssystem und Hardware sowie die gesamte Einsatzumgebung. So können beispielsweise in den Entwicklungswerkzeugen eingebaute trojanische Pferde ihre Schadenswirkung auch dann entfalten, wenn der Anwendungsquellcode davon keine Spuren aufweist, z. B. indem der Compiler immer manipulierten ausführbaren Code erzeugt [Thom_84].[7]

### 3.1 Wie kann Open Source die Sicherheit unterstützen?

Durch die Offenlegung kann bei Open Source prinzipiell jeder prüfen, ob das Programm tatsächlich die angegebene Funktionalität realisiert und kein trojanisches Pferd enthält. Damit steigert Open Source die Transparenz und die Revisionsfähigkeit von Software. Durch einen Code-Review können Sicherheitsrisiken ausfindig gemacht, Fehler behoben und Patches über das Internet allgemein zur Verfügung gestellt werden, sodass sich der Entwicklungsprozess beschleunigt. Schnelle Reaktionszeiten bei der Fehlerbehebung – häufig wird der Patch gleichzeitig mit der Meldung des Fehlers geliefert – tragen zu einem erhöhten Vertrauen der Nutzer bei. In vielen Open-Source-Projekten werden Fehlerdatenbanken betrieben, in die Nutzer Fehlerberichte einsenden können und nach Bearbeitung des Problems unverzüglich informiert werden.

Die Offenlegung des Quellcodes und der Designkriterien wird für kryptographische Algorithmen seit langem als eine notwendige Bedingung für eine Sicherheitsuntersuchung angesehen.[8] Dass „Security by Obscurity“, also Sicherheit durch Verschleierung, ein höchst unzuverlässiges und fragwürdiges Prinzip ist, zeigen auch heute noch immer wieder Beispiele, in denen sich die Geheimhaltung des Quellcodes doch nicht gewährleisten ließ oder sicherheitsrelevante Fehler durch Zufall, gezielte Angriffe auf das ausführbare Programm oder Reverse-Engineering gefunden wurden [Schn_99].

Die durch eine Offenlegung gesteigerte Vertrauenswürdigkeit betrifft insbesondere die Vertraulichkeit, denn im Gegensatz zu Integritäts- und Verfügbarkeitseigenschaften, die großenteils auch durch Erfahrung im Einsatz validiert werden können, bemerkt der Betroffene meist nicht unmittelbar, wenn Unbefugte Kenntnis von seinen Daten erlangen. Notwendig ist eine Offenlegung der Quellen für eine tiefgehende Evaluation und Zertifizierung jedes Systems, beispielsweise nach IT-Sicherheitskriterien.

Die Kooperation vieler Interessierter bei der gemeinsamen Software-Fortentwicklung hat dazu geführt, dass mittlerweile schon eine breite Palette an robusten und verlässlichen Open-Source-Programmen existiert. Open Source ermöglicht bei Bedarf eine Anpassung an eigene Gegebenheiten oder Weiterentwicklung. Abgesehen davon, dass in der Regel die Autoren namentlich bekannt und erreichbar sind, ist der Anwender auch deswegen in geringerem Maße von einem einzelnen Hersteller abhängig, da Support, Wartung und Erweiterungen ebenso von anderen ausgeführt werden können.[9] Tatsächlich bieten mittlerweile etliche Firmen Hotlines, Beratungs- oder Entwicklungsdienstleistungen für Open-Source-Produkte

[7] Ken Thompson kommentiert dies folgendermaßen: „You can't trust code that you did not totally create yourself.“

[8] Prinzip von Kerckhoff (1835-1903): Die Sicherheit eines Kryptosystems darf nicht von der Geheimhaltung des Algorithmus abhängen, sondern nur von der Geheimhaltung des Schlüssels.

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

an. Für einige Nutzer ist Open Source darüber hinaus der Einstieg in die Beteiligung an der Software-Entwicklung und ihrer Qualitätssicherung, nachdem sie sich zunächst gelegentlich am Review- oder Patch-Prozess beteiligt haben.

| **Möglicher Sicherheits- / Vertrauensgewinn** | **Open-Source-Eigenschaft** | **Einschränkungen und Grenzen** | **Lösungen und Schutzmaßnahmen** |
|---|---|---|---|
| Quellcode-Review möglich durch beliebige unabhängige Experten | Offenlegung | keine Garantie, dass tatsächlich ein vollständiger Review durch unabhängige Experten durchgeführt wird und dass der Code verständlich ist | Verwenden formalisierter Verfahren für Review und Evaluation; Berücksichtigung allgemeiner Grundsätze der Software-Entwicklung |
| System basiert nicht auf „Security by Obscurity“ | Offenlegung | geht nur bei „reifen“ Sicherheitstechniken wie Kryptographie | fragwürdige und „unreife“ Sicherheitstechniken, wie z. B. Watermarking, vermeiden |
| keine trojanischen Pferde im System | Offenlegung | nur insoweit das *vollständige* System (incl. Erzeugung des Objektcodes) offengelegt und entsprechend evaluiert wurde | Review und Evaluation des gesamten Systems; Open Source auch für alle verwendeten Werkzeuge |
| schnelle Fehlerbehebung möglich durch Verbreitung von Patches; Anpassung an eigene Gegebenheiten | Offenlegung, Änderungs- und Verbreitungsmöglichkeit | dadurch Möglichkeit, dass sich neue Fehler einschleichen oder dass interne Angreifer in der eigenen Version flexibler trojanische Pferde einbauen | Test vor Installation des Patches; Kapselung der Produktionsversion gegen unbefugte Veränderung (z. B. durch digitale Signaturen und Zertifikate) |
| Nutzer kann selbst einsteigen und sich an der Entwicklung und der Qualitätssicherung beteiligen | Offenlegung, Änderungs- und Verbreitungsmöglichkeit | nur für Personen mit Programmierkenntnissen praktikabel | Bildungskampagnen für den Bereich IT |
| keine Abhängigkeit von einzelnen Herstellern | Offenlegung, Änderungs- und Verbreitungsmöglichkeit | keine Vertragsbeziehung, keine Haftung und Gewährleistung der Autoren | Wartungs- und Distributionsvertrag |
| Qualitätssicherung durch persönliche Motivation der Entwickler; Reputation statt Zeitdruck durch Marketingzwänge | Offenlegung, Kostenfreiheit, Verbreitung | Ausrichtung nur auf persönliche Interessen der Entwickler (oft mehr algorithmischer Kern als Oberflächengestaltung) | Ausrichtung auf Kunden durch Distributoren (z. B. Oberflächen, Support, Wartung, Hotlines); Gewährleisten von Authentizität bei Code und Patches |
| großer Fundus von bewährtem Code, der in neuen Projekten verwendet werden kann | Verbreitung | | |

Tab. 3: Überblick über Open Source und Sicherheit

### 3.2 Open Source = mehr Sicherheit?

Open Source ist kein alleiniges Allheilmittel für Sicherheit [Neum_00]. Die Offenlegung des Quellcodes ist nur so gut, wie ihn auch tatsächlich Fachkundige analysieren, die nicht im Sinn haben, gefundene Fehler für Angriffe auszunutzen, und wie diese ihre Resultate bekannt geben. Zurzeit bleibt weitgehend dem Zufall überlassen, inwieweit welcher Quellcode von der Internet-Community evaluiert wird und wie schnell und gut die Fehler behoben werden.[10]

Auch wird der Code nicht automatisch dadurch besser, dass er veröffentlicht wird – allenfalls durch die persönliche Motivation der Entwickler, die damit ihre Reputation aufbauen, oder durch die Entkopplung von bei kommerzieller Software gängigen Marketingzwängen, dass Produkte termingerecht auch vor einer Freigabe durch die Entwickler auszuliefern sind. In jedem Fall sind wie auch sonst bei einer guten Software-Entwicklung strenge Maßstäbe zur Qualitätssicherung anzulegen, z. B. was die übersichtliche Gestaltung, die Modularisierung, die Dokumentation und die Auslieferung erst nach ausreichenden Tests betrifft.

Außerdem spielen komfortable Bedienoberflächen in Anwendungssoftware eine große Rolle, gerade wenn die Nutzer künftig in einem höheren Maße selbst für ihre Sicherheit und ihren Datenschutz verantwortlich sein sollen. Hier besteht bei Open Source vielfach Nachholbedarf, weil sich die Interessen der Entwickler mehr um die Bereitstellung der technischen Funktionalität, nicht jedoch um das – meist noch einmal genauso aufwändige – Programmieren der Oberflächen drehen. Aus diesem Grund haben es Distributoren und Supportfirmen übernommen, die nähere Ausrichtung auf die Kunden zu realisieren.

Die Änderungsmöglichkeit durch den Anwender bewirkt, dass sich bei ihm durch bereitgestellte Patches – versehentlich oder auch durch Angreifer lanciert – oder durch eigene Modifikationen sicherheitsbedenkliche Fehler einschleichen können. Internen Angreifern ist es leichter möglich, direkt vor der Übersetzung trojanische Pferde in den offengelegten Code einzubauen. Hier können Schutzmaßnahmen wie Kapselung der Produktionsversion gegen unbefugte Veränderung durch Zertifikate mithilfe digitaler Signaturen greifen. Auf dieselbe Weise kann man die Authentizität von bereitgestelltem Code und Patches prüfen.

Eine generelle Grenze besteht darin, dass bei einer Sicherheitsanalyse die Systeme als Ganzes vollständig evaluiert werden müssen. Immerhin stehen neben einer Reihe von Anwendungsprogrammen mittlerweile auch Software-Entwicklungswerkzeuge und Betriebssysteme im Open Source zur Verfügung. Die Idee der Open Hardware wird ebenfalls propagiert.[11] Doch ist es natürlich nicht einfach, die Komplexität der heute eingesetzten und entwickelten Systeme in den Griff zu bekommen.

Open Source entbindet den Anwender nicht von der eigenen Verantwortung für die Datenverarbeitung. Die Lizenzbestimmungen schließen häufig explizit Gewährleistung und Haftung aus.[12] In jedem Fall – egal ob Open oder Closed Source – müssen sich die Anwender darum kümmern, dass die Sicherheit ihres Systems erhalten bleibt. Das laufende Bekanntwerden neuer Sicherheitslücken und Angriffe erfordert heutzutage von der System-

[10] Germano Caronni, Entdecker eines jahrelang unbemerkten Fehlers in der Open-Source-Verschlüsselungssoftware PGP 5.0, betont: „Public code review is a good thing – if it happens." *(Key Generation Security Flaw in PGP 5.0,* BugTraq-Mailinglist, 22. Mai 2000, Message-ID: <20000523141323.A28431@olymp.org>).

[11] http://www.openhardware.org; das Zertifizierungsprogramm für offene Hardware in Deutschland: http://www.de.debian.org/OpenHardware/.

[12] Inwieweit dies wirksam ist, wird juristisch diskutiert, z. B. Axel Metzger, Till Jaeger: *Open Source Software und deutsches Urheberrecht,* GRUR Int. 1999, S. 839; http://www.ifross.de/ifross_html/art1.html;
Till Jaeger: *GPL und Haftung: Ohne Verantwortung?,* Linux-Magazin 5/2000, S. 134; http://www.ifross.de/ifross_html/art3.html;
Jürgen Siepmann: *Lizenz- und haftungsrechtliche Fragen bei der kommerziellen Nutzung Freier Software,* JurPC Web-Dok. 163/1999, Abs. 1-289; http://www.jurpc.de/aufsatz/19990163.htm.

administration eine ständige Beschäftigung mit dem Thema und eine regelmäßige Wartung der Systeme. Durch unterstützende Alert-Dienste sind schnelle Reaktionen möglich.

### 3.3 Open Source Security Tools

Gerade Security-Tools[13] werden immer mehr als Open Source bereitgestellt, da bei ihnen die Vertrauenswürdigkeit eine besonders große Rolle spielt. Sie profitieren dabei von der Tatsache, dass viele Augen mehr als zwei sehen, d. h. dass sich unabhängige Programmierer am Review-Prozess beteiligen, etwaige Sicherheitsrisiken melden und Fehler beheben. Da sie sich des Verbreitungsmechanismus über das Internet bedienen, werden die mitgelieferten Dateien incl. Quellcode mit einer digitalen Signatur von den Autoren oder einer Vertrauensstelle versehen, um ihre Integrität und Authentizität belegen zu können. Der Nutzer sollte die digitale Signatur vor einem Compilieren und Installieren auf ihre Gültigkeit prüfen.

Das Bundesamt für Sicherheit in der Informationstechnik (BSI) arbeitet auf der Grundlage von Open-Source-Betriebssystemen an Projekten, die die Entwicklung eines gesicherten PCs und dessen sicherer An- und Einbindung in Netze zum Ziel haben [KBSt_00].

### Fazit

Die Offenlegung des Quellcodes ist ein wichtiges Hilfsmittel, um Sicherheit und Vertrauenswürdigkeit von Systemen zu erreichen. Das offene und kooperative Software-Entwicklungsmodell von Open-Source-Projekten hat gezeigt, dass man auf diese Weise robuste und verlässliche Programme erstellen kann. Allerdings ist dies kein Automatismus, sondern erfordert Sorgfalt im gesamten Entwicklungsprozess sowie bei der Evaluierung durch Experten.

Zwar reicht allein die Offenlegung des Codes für Sicherheit nicht aus, jedoch ist sie eine essentielle Voraussetzung für effektive Sicherheitsuntersuchungen: Im herkömmlichen Closed-Source-Modell können trojanische Pferde nicht ausgeschlossen werden. Solche Systeme sollten gerade in sicherheitskritischen Bereichen nicht eingesetzt werden.

Es besteht ein nationales Interesse daran, dass vertrauenswürdige Hard- und Software-Systeme bereitgestellt und eine „Fernsteuerung“ beispielsweise durch Firmen oder Geheimdienste anderer Länder unterbunden wird.

Da völlige Eigenentwicklungen für die meisten Nationen zu aufwändig wären, sollten sie eine Infrastruktur und einen Rahmen dafür schaffen, mithilfe von Open Source zu sicheren und vertrauenswürdigen Produkten zu kommen. Dazu gehören z. B. die Klärung von offenen juristischen Fragen, der Aufbau eines Netzes von unabhängigen Prüfstellen und die Schulung und Ausbildung der Nutzer im IT-Bereich. Auf keinen Fall sollten Software-Patente in einer Form eingeführt werden, die kontraproduktiv auf Open Source wirken würde.[14]

---

[13] Siehe beispielsweise unter http://www.infosecuritymag.com/feb2000/opensourcetools.htm oder http://www.performancecomputing.com/features/9907f1.shtml.

[14] Brigitte Zarzer: *Software-Patente: Verbiegung des Rechts?,* Telepolis-Interview mit Hartmut Pilch, 2000-06-11; http://www.heise.de/tp/deutsch/inhalt/te/8239/1.html.

**Literatur**

[KBSt_00] Bahr, Rudolf E./Reiländer, Ralf/Troles, Egon: *Open Source Software in der Bundesverwaltung,* KBSt-Brief Nr. 2/2000; http://linux.kbst.bund.de/brief2-2000.html

[Neum_00] 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

[Raym_99] Raymond, Eric S.: *The Cathedral and the Bazaar,* Version 1.45, August 1999; http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar.html

[Schn_99] Schneier, Bruce: *Crypto-Gram September 15, 1999;* http://www.counterpane.com/crypto-gram-9909.html#OpenSourceandSecurity

[Thom_84] 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/