Open Source Software: Eine sicherheitstechnische Beurteilung

PD Dr. Rolf Oppliger*

14. August 2003†

Zusammenfassung

Die Diskussionen um Open Source Software und entsprechende Lizenzierungsmodelle sind heute allgegenwärtig. Von den Verfechtern von Open Source Software wird dabei immer wieder behauptet, dass sich die hauptsächliche Eigenschaft von Open Source Software, dass der Quellcode freigegeben und öffentlich verfügbar ist, auch positiv auf die Sicherheitseigenschaften der Software auswirke. Für diese Behauptung gibt es allerdings bis heute keine wissenschaftlich erhärteten Indizien. In diesem Bericht wird Open Source Software sicherheitstechnisch untersucht und beurteilt, d.h. es wird insbesondere die Frage aufgegriffen, ob die Offenlegung von Quellcode geeignet ist, die Sicherheit von Software nachhaltig zu verbessern. Der Bericht kommt dabei zu einem verneinenden Schluss, d.h. die Offenlegung von Quellcode hat keinen nachweisbaren Einfluss auf die Sicherheitseigenschaften von Software und kann entsprechend nicht als Sicherheitskriterium herangezogen werden. Stattdessen haben die Erfolgsfaktoren, die man aus dem Software Engineering kennt, einen nachhaltigen Einfluss auf die Sicherheitseigenschaften von Software. Die Diskussionen um Open Source Software und entsprechende Lizenzierungsmodelle sind vor allem wirtschaftlich motiviert und müssen auf der Basis von Wirtschaftlichkeitsargumenten geführt werden.

*eSECURITY Technologies Rolf Oppliger, Beethovenstrasse 10, CH-3073 Gümligen, Phone/Fax: +41 (0)79 654 8437, E-Mail: rolf.oppliger@esecurity.ch

†Geringfügige Nachbesserungen sind am 19. bzw. 21. August 2003 gemacht worden.

Inhaltsverzeichnis

1 Einleitung

Die Diskussionen um Open Source Software und entsprechende Lizenzierungsmodelle sind heute allgegenwärtig. Von den Verfechtern von Open Source Software wird dabei immer wieder behauptet, dass sich die hauptsächliche Eigenschaft von Open Source Software, dass der Quellcode freigegeben und öffentlich verfügbar ist, auch positiv auf die Sicherheitseigenschaften der Software auswirke. Für diese Behauptung gibt es allerdings bis heute keine wissenschaftlich erhärteten Indizien, und in der Tat zeichnen die einzigen wissenschaftlichen Arbeiten, die zu diesem Thema verfasst worden sind, ein anderes (und differenzierteres) Bild. So haben britische Wissenschaftler z.B. ein Zuverlässigkeitszuwachsmodell entwickelt, in dem man zeigen kann, dass sich unter bestimmten — in gewissem Sinne idealen — Voraussetzungen die Offenlegung von Quellcode weder positiv noch negativ auf die Sicherheitseigenschaften der entsprechenden Software auswirkt (das Modell wird mit seinen Implikationen in Abschnitt 3 kurz vorgestellt und diskutiert).

In diesem Bericht wird Open Source Software sicherheitstechnisch untersucht und beurteilt, d.h. es wird insbesondere die Frage aufgegriffen, ob die Offenlegung von Quellcode geeignet ist, die Sicherheit von Software nachhaltig zu verbessern. Der Bericht kommt dabei zu einem verneinenden Schluss, d.h. die Offenlegung von Quellcode hat keinen nachweisbaren Einfluss auf die Sicherheitseigenschaften von Software und kann entsprechend nicht als Sicherheitskriterium herangezogen werden. Die Diskussionen um Open Source Software und entsprechende Lizenzierungsmodelle sind vor allem wirtschaftlich motiviert und müssen auf der Basis von Wirtschaftlichkeitsargumenten geführt werden. Allerdings ergibt sich eine Schwierigkeit aus der Tatsache, dass statistisch relevantes Datenmaterial über die Vollkosten von Open Source Software bis heute nicht verfügbar ist.

Unabhängig von der Frage, ob der Quellcode von Software offengelegt ist, haben grosse Softwaresysteme immer mit dem Problem zu kämpfen, dass das Finden von sicherheitsrelevanten Fehlern — aufgrund ihrer grossen Zahl — immer einfach möglich ist, und dass die Wahrscheinlichkeit, dass ein Softwarehersteller unter Zuhilfenahme von beliebig aufwendigen Testverfahren genau den (die) Fehler findet, den (die) auch ein Angreifer gefunden hat, verschwindend klein ist [And01]. Unter der Annahme, dass eine Software z.B. 1'000 sicherheitsrelevante Fehler enthält[1], von denen in einer bestimmten Zeitspanne (z.B. ein Monat) ein Angreifer einen findet und der Softwarehersteller 100 finden und korrigieren kann, beträgt die Wahrscheinlichkeit, dass sich unter den gefundenen und korrigierten Fehlern der vom Angreifer gefundene Fehler befindet, nur gerade 10 Prozent (d.h. mit einer Wahrscheinlichkeit von 90 Prozent ist der vom Angreifer gefundene Fehler zum Zeitpunkt seiner Auffindung noch nicht korrigiert). Je weiter die Schere zwischen zwar vorhandenen aber nicht gefundenen (und entsprechend auch noch nicht korrigierten) Fehlern auseinanderklafft, umso kleiner wird die Wahrscheinlichkeit, dass der Softwarehersteller genau den (die) Fehler findet und korrigiert, über den (die) das System letztlich angegriffen wird. Wie bereits erwähnt, gilt dieses Problem generell für grosse Softwaresysteme und hat nichts mit der Frage zu tun, ob der entsprechende Quellcode offengelegt ist. Das Problem hat aber einen erheblichen Einfluss auf den grundsätzlich erreichbaren Grad an Sicherheit und Zuverlässigkeit von Software.

[1] Die Grössenordnung dieser Zahl ist für grosse Softwaresysteme durchaus realistisch. Wenn man davon ausgeht, dass jeweils 1'000 Programmzeilen im Durchschnitt einen Fehler enthalten, dann kann man davon ausgehen, dass das ca. 45'000'000 Programmzeilen umfassende Betriebssystem Windows XP ca. 45'000 sicherheitsrelevante Fehler enthält. Die meisten dieser Fehler bleiben während der Laufzeit der Software unentdeckt.

2 Open Source und Closed Source Software

Im Rahmen dieses Berichtes wird Software als *Open Source* bezeichnet, wenn der Quellcode offengelegt (d.h. öffentlich und frei verfügbar) ist. Wenn eine Software nicht Open Source ist, dann wird sie als *Closed Source* bezeichnet. Man beachte, dass es auch im Rahmen von Closed Source Software möglich ist, den Quellcode gegenüber einem definierten und begrenzten Personenkreis offenzulegen (zum Beispiel im Rahmen von Geheimhaltungsvereinbarungen). Die entsprechende Software wird dann treuhänderisch hinterlegt und der Lizenznehmer erhält das (nicht exklusive) Recht, den Quellcode zu insipizieren[2]. Eine solche Möglichkeit bietet sich in der Tat an, um die zum Teil etwas festgefahrenen Meinungen und Diskussionen über die Notwendigkeit von Quellcodeinspektionen zu entschärfen. Viele Softwarehersteller bieten ihren Grosskunden jedenfalls heute eine solche Möglichkeit an. In der in diesem Bericht verwendeten Terminologie ist und bleibt die Software aber Closed Source. Für die Unterscheidung zwischen Open und Closed Source Software ist alleine die Frage ausschlaggebend, ob der Zugang zum Quellcode auf irgendeine Art und Weise beschränkt ist. Ist der Zugang beschränkt, dann handelt es sich um Closed Source Software; ist der Zugang nicht beschränkt, dann handelt es sich um Open Source Software.

Der Entwicklungsprozess von Open Source Software ist in [MMP00] beschrieben. Grundsätzlich gibt es viele Möglichkeiten, Software als Open Source Software zu lizenzieren und unter `http://www.opensource.org/licenses/index.html` und `http://www.gnu.org/licenses/license-list.html` sind die wichtigsten Lizenzierungsmöglichkeiten und entsprechende Lizenzverträge zusammengestellt und mit ihren Vor- und Nachteilen diskutiert. In der Praxis ist vor allem die GNU General Public Licence (GPL) der Free Software Foundation (FSF[3]) wichtig.

In den Diskussionen um Open Source Software und deren Sicherheit wird immer wieder behauptet, dass Open Source Software offene Standards implementiere, während Closed Source Software proprietäre Standards implementiere. Diese Behauptung ist in dieser Form falsch und irreführend. In der Tat sind die Fragen, ob eine Software Open oder Closed Source ist, bzw. ob sie offene oder proprietäre Standards implementiert, unabhängig voneinander und Closed Source Software (Open Source Software) kann durchaus auch offene (proprietäre) Standards implementieren. Vor diesem Hintegrund ist z.B. das in [Sch99,Schn00] geäusserte Bekenntnis zu Open Source Software zu sehen und zu relativieren. Das Bekenntnis betrifft in erster Linie den Einsatz von offenen Standards (insbesondere für kryptografische Algorithmen und Protokolle), und ein solcher Einsatz ist sowohl im Rahmen von Open Source Software als auch im Rahmen von Closed Source Software möglich und sinnvoll. Der (falschen) Behauptung folgend wird das Closed Source Software zugrunde liegende Sicherheitsmodell manchmal auch mit "security through obscurity" und das Open Source Software zugrunde liegende Modell mit "security through reviewed code" umschrieben. Beide Umschreibungen sind tendenziös und irreführend. Auf der einen Seite impliziert Closed Source Software nicht notwendigerweise "security through obscurity" (weil — wie bereits erwähnt — Closed Source Software durchaus auch offene Standards implementieren kann), und auf der anderen Seite impliziert Open Source Software auch nicht notwendigerweise, dass der entsprechende Quellcode mit der erforderlichen Sorgfalt und in hinreichender Tiefe überprüft und kontrolliert worden ist. Schliesslich ist vor diesem Hintergrund auch der Bezug auf das der Kryptografie entlehnte Prinzip von Kerckhoffs [Ker83] fragwürdig[4]. Das

[2] Peter G. Neumann spricht in diesem Zusammenhang von *Open-Box* Software [Neu00]. Dieser Begriff unterscheidet sich kaum von *Open Source*, so dass es möglicherweise sinnvoller wäre, in diesem Zusammenhang von *Openable* oder *Inspectable* Source Code Software zu sprechen.

[3] `http://www.fsf.org`

[4] Das Prinzip von Kerckhoffs wird in Sicherheitsdiskussionen immer wieder ins Feld geführt. Ein

Prinzip von Kerckhoffs besagt im Wesentlichen, dass man bei der Beurteilung der Sicherheit eines kryptografischen Systems nicht davon ausgehen sollte, dass der Angreifer das System nicht kennt. Damit ist aber nicht notwendigerweise gesagt, dass dem Angreifer die Implementation des Systems im Quellcode vorliegen muss (in der Tat hat es in der Zeit, in der Kerckhoffs das später nach ihm benannte Prinzip formuliert und postuliert hat, so etwas wie "Quellcode" noch gar nicht gegeben).

3 Vorarbeiten

Trotz der hohen Visibiliät und Brisanz des Themas, hat sich die Wissenschaft bis heute noch kaum zu den Sicherheitseigenschaften von Open Source Software geäussert. In Kapitel 2.4 seines Handbuches "Secure Programming for Linux and Unix HOWTO" hat David A. Wheeler die Meinungen von verschiedenen Personen und Exponenten der Open Source Software Bewegung zusammengetragen[5]. Am verbreitetsten ist die Meinung, dass Open Source Software zwar ein gewisses Potential in bezug auf sichere Software hat, dass es in dieser Sache aber keinen Automatismus gibt (d.h. eine Software ist nicht nur deshalb sicher, weil ihr Quellcode offengelegt ist).

Im Juni 2002 hat die Alexis de Tocqueville Institution einen kontroversen Bericht zum Thema herausgebracht [Bro02]. Aus sicherheitstechnischer Sicht ist vor allem die im Bericht aufgestellte Behauptung umstritten, dass sich der Einsatz von Open Source Software im Sinne der GPL durchaus auch zu einem nationalen Sicherheitsproblem entwickeln könne[6]. Der Bericht liefert für diese Behauptung kaum plausible Begründungen und in [VF02] werden denn auch einige der dem Bericht zugrundeliegenden Falschannahmen korrigiert und richtiggestellt.

Eine der wenigen wissenschaftlichen Untersuchungen zum Thema Sicherheit von Open Source Software geht auf Ross Anderson und seine Forschergruppe an der britischen Cambridge University zurück [BAB99,And02,And03]. Anderson hat versucht, den Zuwachs an Zuverlässigkeit in einem (hinreichend grossen[7]) System in Abhängigkeit von durchgeführten Tests mathematisch zu beschreiben, und dabei ein Zuverlässigkeitszuwachsmodell (engl. "reliability growth model") entwickelt. Das Modell beschreibt und erklärt die empirisch ermittelte Tatsache, dass ab einer bestimmten Grösse eines Systems die Zuverlässigkeit nur noch proportional zum geleisteten Testaufwand steigt (über ähnliche Resultate wird unter anderem auch in [BB96] und [Lyu96] berichtet). Entsprechend kann die Wahrscheinlichkeit E eines Ausfalls zu einem Zeitpunkt t, zu dem bereits n Fehler korrigiert worden sind, folgendermassen approximiert werden:

$$E = \sum_{i=n+1}^{\infty} e^{-E_i t} \approx K/t$$

Dabei entspricht $e^{-E_i t}$ der Wahrscheinlichkeit, dass der i.te Fehler auch nach t Tests unentdeckt bleibt. Entsprechend errechnet sich die Wahrscheinlichkeit E

anderes Beispiel ist z.B. die Frage, ob es richtig und sinnvoll ist, über gefundene Programm- und Programmierfehler möglichst früh und breit zu publizieren (um die entsprechenden Softwarehersteller in Zugzwang zu setzen). Häufig wird das Prinzip von Kerckhoffs dabei ein wenig überstrapaziert und man kann durchaus auch den Standpunkt vertreten, dass das Prinzip von Kerckhoffs überhaupt nur auf kryptografische Fragestellungen angewandt werden kann. Weder die in diesem Bericht angesprochene Fragestellung noch die Fragestellung betreffend der Veröffentlichung von Programmfehlern sind kryptografischer Art.

[5] `http://www.tldp.org/HOWTO/Secure-Programs-HOWTO/open-source-security.html`

[6] Auf Seite 18 des Berichtes wird behauptet: "open source GPL use by government agencies could easily become a national security concern".

[7] "Hinreichend gross" bedeutet in diesem Zusammenhang so gross, dass statistische Verfahren greifen und eingesetzt werden können.

als Summe dieser Wahrscheinlichkeiten ab dem $(n+1)$.ten Fehler. Diese Summe kann mit K/t approximiert werden[8], wobei K für eine bestimmte Software konstant ist und als Integrationskonstante bezeichnet wird. Aus dieser Formel kann man schliessen, dass die Ausfallwahrscheinlichkeit einer Software nur von der Qualität des Codes (ausgedrückt durch K) und den durchgeführten Tests (ausgedrückt durch t) abhängt. Entsprechend geht man im Software Engineering davon aus, dass eine Software, die eine *Meantime To Failure* (MTTF) von 10'000 Stunden aufweisen soll, im Durchschnitt auch etwa 10'000 Stunden getestet werden sollte (diese Regel ist z.B. in [BF93] beschrieben).

Im Zuverlässigkeitszuwachsmodell von Anderson kann man zeigen, dass sich unter bestimmten — in gewissem Sinne idealen — Voraussetzungen die Offenlegung von Quellcode weder positiv noch negativ auf die Sicherheitseigenschaften der betroffenen Software auswirkt. Im Rahmen seiner Untersuchungen hat Anderson aber auch auf viele Faktoren hingewiesen, die zu einer Situation führen können, in der die Voraussetzungen von den idealen Voraussetzungen abweichen, und in der möglicherweise das Gleichgewicht zwischen sicherheitstechnischen Vor- und Nachteilen von Open Source Software gegenüber Closed Source Software nicht mehr gilt (vgl. [And03]). Die meisten dieser Faktoren sind wirtschaftlicher bzw. wirtschaftspolitischer Art.

Das Zuverlässigkeitszuwachsmodell von Anderson liefert ein erstes wissenschaftliches Indiz dafür, dass die Offenlegung von Quellcode keinen nachweisbaren Einfluss auf die Sicherheitseigenschaften von Software hat. Allerdings darf man das Indiz auch nicht überbewerten. Auf der einen Seite weist Anderson in seinen Arbeiten immer wieder selbst darauf hin, dass das Gleichgewicht zwischen den sicherheitstechnischen Vor- und Nachteilen von Open Source Software gegenüber Closed Source Software fragil ist (d.h. es gibt — wie oben erwähnt — eine grosse Zahl von Faktoren, die das Gleichgewicht stören können), und auf der anderen Seite sind Zuverlässigkeit und Sicherheit zwar ähnliche aber eben doch verschiedene Systemaspekte. Im Bereich der Zuverlässigkeit interessiert vor allem die Verfügbarkeit und das korrekte Funktionieren eines Systems (im Sinne seiner Spezifikation). Demgegenüber interessiert man sich im Sicherheitsbereich auch und gerade für Vertraulichkeits- und Integritätsaspekte, und untersucht, ob und wenn ja wie ein potentieller Angreifer ein System missbrauchen kann. Das angegriffene System kann sich dabei durchaus getreu seiner Spezifikation verhalten.

In den folgenden Abschnitten werden die hauptsächlichen Sicherheitsargumente für Open und Closed Source Software erläutert und diskutiert. Dabei gilt es zu beachten, dass sich die Begutachtung und Kontrolle von Quellcode nicht auf die betrachtete Software alleine beschränken kann, sondern dass alle bei der Entwicklung der Software eingesetzten Dienstprogramme und Werkzeuge (z.B. Kompiler) in die Betrachtung miteinbezogen werden müssen. Im Rahmen der Überreichung des ACM Turing Awards hat Ken Thompson bereits 1984 gezeigt, wie einfach es möglich ist, einen Kompiler dahingehend zu manipulieren, dass er kompromittierenden Code (z.B. ein Trojanisches Pferd oder eine Hintertür) während der Kompilation von Quellcode in den entsprechenden Maschinencode einbringt [Tho84]. Der kompromittierende Code ist dann im Quellcode nicht sicht- bzw. erkennbar. Die Möglichkeit der Einbringung von komprimittierendem Code auch während der Erstellung von Maschinencode gilt es zu beachten, wenn eine Quellcodeinspektion ernsthaft in Erwägung gezogen wird.

[8] In [BAB99] und in den Anhängen von [And02] und [And03] wird die Approximation formal hergeleitet.

4 Sicherheitsargumente für Open Source Software

Es gibt mindestens vier hauptsächliche Sicherheitsargumente für Open Source Software: dass der Quellcode von vielen Personen begutachtet und kontrolliert werden kann (das "Viele Augen"-Argument), dass Fehler und Hintertüren überhaupt nur im Quellcode gefunden werden können (das "Fehlersuche ist nur im Quellcode möglich"-Argument), dass sich Open Source Software generell durch besseren Code auszeichnet (das "Besserer Code"-Argument), und dass die Entwickler von Open Source Software unabhängig von kommerziellen Interessen sind (das Unabhängigkeitsargument).

4.1 Das "Viele Augen"-Argument

Die Verfechter von Open Source Software argumentieren meistens, dass offengelegter (d.h. freigegebener und öffentlich frei verfügbarer) Quellcode potentiell von sehr vielen Personen begutachtet und kontrolliert werden kann, dass dadurch mehr Fehler und Hintertüren entdeckt und korrigiert werden, dass dadurch umgekehrt die Wahrscheinlich sinkt, dass die Software unbekannte Fehler und Hintertüren enthält, und dass dadurch letztlich ein gegenüber Closed Source Software erhöhtes Sicherheitsniveau resultiert.

Im Kern dieser (verführerisch einfachen) Argumentation steckt die Annahme, dass die potentiell vorhandene Möglichkeit, den Quellcode von Open Source Software zu begutachten und zu kontrollieren, von einer hinreichend grossen Zahl von Personen auch tatsächlich wahrgenommen wird. Diese Annahme ist bis heute empirisch nicht untersucht, d.h. es gibt keine wissenschaftliche Untersuchung, die Aufschluss darüber gibt, ob und wenn ja wieviele Personen mit welchen Aufwänden den Quellcode von Open Source Software in bezug auf Fehler und Hintertüren begutachten und kontrollieren, bzw. wie die resultierenden Aufwände und Erfolge in Relation zu den Aufwänden und Erfolgen der Hersteller von anderer (insbesondere Closed Source) Software stehen. Die Möglichkeit, Quellcode zu begutachten und zu kontrollieren, bewirkt nicht automatisch, dass dies auch wirklich getan wird. So gibt es viele (Open und Closed) Source Software, die ausser dem (den) Entwickler(n) selbst wahrscheinlich noch niemand ernsthaft begutachtet und kontrolliert hat (man denke hier etwa an spezielle Gerätetreiber). Die Erfahrungen mit der Software *Pretty Good Privacy* (PGP) haben jedenfalls gezeigt, dass die Veröffentlichung von Quellcode nicht dazu führt, dass eine grosse Zahl von Fehlern und/oder Hintertüren aufgedeckt wird [Opp01]. Zudem gibt es ein in der Soziologie bekanntes Phänomen, das bewirkt, dass je mehr Personen einen Umstand sehen, bei dem ein Handlungsbedarf existiert (z.B. ein Unfall oder ein Gewaltverbrechen), umso weniger sind einzelne Personen als Individuen bereit, etwas dagegen zu unternehmen und sich persönlich zu engagieren. Man kann durchaus auch argumentieren, dass ein ähnliches Phänomen bei der Begutachtung und Kontrolle von Quellcode eine Rolle spielen kann.

Das "Viele Augen"-Argument geht weiter davon aus, dass sich die den Quellcode begutachtenden und kontrollierenden Personen in einem gewissen Sinne "fair" verhalten und gefundene Fehler und Hintertüren auch melden. An dieser Stelle kann man sich die Frage stellen, was eine Person macht, wenn sie einen Fehler oder eine Hintertür gefunden hat. Ist die Person gutwillig, dann wird sie den gefundenen Fehler oder die gefundene Hintertür wahrscheinlich bekanntgeben und/oder selbst korrigieren. Ist die Person aber weniger gutwillig oder sogar böswillig, dann hat man keine Gewähr, dass eine Bekanntgabe des Fehlers oder der Hintertür auch wirklich erfolgt. In der Tat wäre es denkbar, dass das Wissen um einen Fehler oder eine Hintertür entweder selbst ausgenutzt oder veräussert wird. Je gravierender der Fehler oder die Hintertür ist, umso mehr steigt die Wahrscheinlichkeit, dass der zweite

Weg beschritten wird. Natürlich ist die Ausgangslage in den Testlabors von Entwicklungsfirmen von Closed Source Software nicht anders; hier stehen aber definierte Mengen an Personen zur Verfügung, die vertragsrechtlich an den Softwarehersteller gebunden sind. Diese Ausgangslage ist aus sicherheitstechnischer Sicht sicherlich von Vorteil.

Zusammenfassend kann man sagen, dass das "Viele-Augen"-Argument in Einzelfällen wohl zutreffen kann und auch zutreffen wird, dass aber eine Verallgemeinerung auf beliebige Software gefährlich ist, weil es die Nutzer in einer falschen Sicherheit wiegt. Viele Nutzer von Open Source Software schliessen aus der Möglichkeit, dass der Quellcode begutachtet und kontrolliert werden kann, dass der Code auch tatsächlich begutachtet und kontrolliert worden ist, und diese potentielle Fehleinschätzung kann bewirken, dass auf eine eigene Überprüfung verzichtet oder die Software falsch eingesetzt wird (z.B. zur Verarbeitung von klassifizierten Daten). Es entspricht einer alten Weisheit, dass scheinbare Sicherheit schlechter ist als keine Sicherheit (weil dadurch benutzerseitig ein falsches Verhalten provoziert werden kann).

4.2 Das "Fehlersuche ist nur im Quellcode möglich"-Argument

Die Verfechter von Open Source Software argumentieren zuweilen, dass das Suchen und Finden von sicherheitsrelevanten Fehlern und Hintertüren (z.B. durch freiwillige Softwareentwickler) nur auf der Basis von Quellcode möglich sei.

Diesem "Fehlersuche ist nur im Quellcode möglich"-Argument liegt letztlich die Annahme zugrunde, dass ein Kompiler eine Einwegfunktion implementiert, d.h. dass aus einem Machinencode kein (richtiger und sinnvoller) Quellcode zurückgewonnen werden kann. Diese Annahme ist falsch und mit heute verfügbaren Werkzeugen (insbesondere Debugger und Dekompiler) ist es durchaus möglich, aus einem Maschinencode einen funktional korrekten Quellcode zu erzeugen (auch wenn der resultierende Quellcode vom originalen abweicht). In [VF02] wird denn auch behauptet, dass das Finden von sicherheitsrelevanten Programmierfehlern und Hintertüren im Maschinencode von kompilierten Programmen nicht viel schwieriger sei, als das Finden im Quellcode[9]. In einer in diesem Zusammenhang immer wieder genannten Analogie wird der Quellcode einer Software mit dem Bauplan eines Gebäudes verglichen und der ausführbare Machinencode mit dem entsprechenden (real existierenden) Gebäude. Natürlich ist es immer möglich, von einem gegebenen Gebäude einen Bauplan zu erstellen; ob dieser Plan dann allerdings mit dem ursprünglich (beim Bau des Gebäudes) verwendeten Plan übereinstimmt ist zwar fraglich aber letztlich auch nicht relevant.

Zusammenfassend kann man sagen, dass das "Fehlersuche ist nur im Quellcode möglich"-Argument — wenigstens in seiner absoluten Form — nicht stimmt. Es scheint nicht einmal so zu sein, dass eine Fehlersuche im Quellcode wesentlich einfacher ist, als eine Fehlersuche im Machinencode (wenn geeignete Werkzeuge zur Verfügung stehen). Wenn man das "Fehlersuche ist nur im Quellcode möglich"-Argument mit dem von Thompson eingebrachten Argument kombiniert, kommt man zum Schluss, dass — wenn man mit der Inspektion von Quellcode einmal begonnen hat — man rekursiv sämtliche Software, die auf einem Computersystem eingesetzt wird, im Quellcode inspizieren muss. Das ist theoretisch vielleicht möglich (wenn man alle Software im Quellcode vorliegen hat), praktisch wird man aber relativ schnell an seine Grenzen stossen. Man beachte in diesem Zusammenhang insbesondere auch, dass das sicherheitsmässsige Begutachten und Kontrollieren von

[9] Wenn eine Software funktional erweitert werden soll, dann spielt es allerdings schon eine Rolle, ob der Quellcode verfügbar ist.

Quellcode mindestens so aufwendig ist, wie das Verfassen des Codes. Entsprechend könnte man sämtliche auf einem Computersystem eingesetzte Software auch gleich selbst programmieren.

4.3 Das "Besserer Code"-Argument

Von den Verfechtern von Open Source Software wird etwa behauptet, dass alleine die Tatsache, dass der Quellcode offengelgt ist, dazu führe, dass sich die Entwickler besser an Programmierrichtlinien zur Erstellung von "gutem" und qualitativ hochstehendem Quellcode halten. In der Terminologie des Zuverlässigkeitszuwachsmodells von Anderson würde die Offenlegung des Quellcodes eine tiefere Integrationskonstante K bedeuten. Äquivalent dazu steigere die Offenlegung von Quellcode auch die Transparenz und Revisionsfähigkeit der entsprechenden Software.

Diesem "besserer Code"-Argument liegt letztlich die Annahme zugrunde, dass ein Softwareentwickler zwar "guten" und qualitativ hochstehenden Quellcode erzeugen kann, dass er das im Rahmen von Closed Source Software aber vielleicht nicht tut, weil er nicht fürchten muss, dass andere Personen seinen Quellcode anschauen und kritisch würdigen. Diese Annahme ist in zweifacher Hinsicht problematisch. Auf der einen Seite ist es fraglich, ob ein Softwareentwickler seinen Programmierstil ändert, nur weil möglicherweise andere Personen die von ihm entwickelte Software im Quellcode sichten können. Auf der anderen Seite ist es auch bei der Entwicklung von Closed Source Software typischerweise so, dass — im Rahmen von "Peer Reviews" — eine Begutachtung und Kontrolle des Quellcodes stattfindet. Je professioneller eine Softwareentwicklung erfolgt, umso seriöser sind solche Reviews.

Zusammenfassend kann man sagen, dass das "Besserer Code"-Argument einigermassen willkürlich und kaum plausibel ist. Man könnte genausogut das Gegenteil vertreten: dass ein Entwickler von Closed Source Software für seine Programmierleistung bezahlt wird, und dass er schon deshalb eher bereit ist, der Qualität des von ihm erstellten Quellcodes eine grössere Beachtung zu schenken.

4.4 Das Unabhängigkeitsargument

Schliesslich wird von den Verfechtern von Open Source Software auch immer wieder behauptet, dass die Entwickler von Open Source Software grundsätzlich unabhängig(er) und frei von kommerziellen Interessen seien.

Dieses Unabhängigkeitsargument ist teilweise richtig. Es ist sicherlich so, dass ein Entwickler, der für seine Arbeit nicht (materiell) entlöhnt wird, kommerzielle Interessen von Firmen nicht oder wenigstens nicht sehr gut vertreten wird. Auf der anderen Seite bedeutet die Tatsache, dass ein Entwickler unabhängig von solchen Interessen ist, nicht automatisch auch, dass er frei von anderen (z.B. nachrichtendienstlichen) Interessen ist. Das Problem bei solchen Fragestellungen ist, dass man die Antwort im Prinzip nicht kennt und auch nicht ermitteln kann. Niemand weiss, wie die Interessen der verschiedenen Softwareentwicklergruppen wirklich sind. Entsprechend ist man gut beraten, eine Situation zu schaffen, in der solche Interessen keine oder höchstens eine untergeordnete Rolle spielen können. Eine solche Situation wird z.B. durch eine konsequente Berücksichtigung und Befolgung der Prinzipien des Software Engineering erreicht. Auf diesen Punkt wird in den Schlussfolgerungen noch vertieft eingegangen.

5 Sicherheitsargumente für Closed Source Software

Es gibt mindestens fünf hauptsächliche Sicherheitsargumente für Closed Source Software: dass "sichere" Betriebssysteme in der Vergangenheit immer als Closed Source Software implementiert worden sind (das "Sichere Betriebssysteme sind Closed Source"-Argument), dass weniger Personen Einblick in den Quellcode haben (das "Wenige Augen"-Argument), dass Aspekte der Implementierung im Rahmen von Closed Source Software geheimgehalten werden können (das Geheimhaltungsargument), dass Verantwortlichkeiten und Haftungsfragen generell besser geregelt sind (das Verantwortungs- und Haftungsargument), und dass Closed Source Software in der Regel auch mit besseren Dokumentationen und Unterstützungsleistungen verbunden sind (das "bessere Dokumentation und Unterstützungs"-Argument).

5.1 Das "Sichere Betriebssysteme sind Closed Source"-Argument

Die Verfechter von Closed Source Software berufen sich gerne auf die Tatsache, dass alle in bezug auf einen bestimmten Sicherheitskatalog[10] evaluierten und zertifizierten "sicheren" Betriebssysteme[11] Closed Source sind, und leiten daraus einen sicherheitstechnischen Vorteil für Closed Source Software ab.

Das "sichere Betriebssysteme sind Closed Source"-Argument trifft vordergrüdig zu. Hintergründig gilt es aber zu beachten, dass eine Evaluation und Zertifikation eines Systems (z.B. Softwaresystems) immer mit erheblichen Kosten verbunden ist, dass diese Kosten für Closed Source Software über Lizenzgebühren abgegolten werden können, und dass es eine solche Möglichkeit für Open Source Software nicht gibt (weil die entsprechenden Lizenzen in der Regel gebührenfrei sind). Entsprechend ist die Evaluation und Zertifikation von Software grundsätzlich nicht mit der Idee von Open Source Software (im strengen Sinne) vereinbar. Diese grundsätzliche Unvereinbarkeit gilt es zu beachten, wenn Staaten sowohl die Idee der Evaluation und Zertifikation von Systemen als auch Open Source Software fördern.

Zusammenfassend kann man sagen, dass das "sichere Betriebssysteme sind Closed Source"-Argument vordergründig zwar zutrifft, dass aber die Gründe dafür nicht in grundsätzlich besseren Sicherheitseigenschaften zu suchen sind, sondern in der Art und Weise, wie Evaluations- und Zertifikationskosten üblicherweise finanziert werden. Zudem ist die Frage, ob und wenn ja in welchem Rahmen der Markt Sicherheitsevaluationen und entsprechende Zertifikate überhaupt verlangt, bis heute nicht schlüssig geklärt.

5.2 Das "Wenige Augen"-Argument

Ähnlich wie die Verfechter von Open Source Software behaupten, dass die Tatsache, dass viele Augen den Quellcode begutachten und kontrollieren können, aus sicherheitstechnischer Sicht vorteilhaft sei, behaupten die Verfechter von Closed Source Software das Gegenteil: dass die Tatsache, dass nur wenige Augen einen Quellcode begutachten und kontrollieren können, aus sicherheitstechnischer Sicht ein Vorteil darstelle.

Diesem "Wenige Augen"-Argument liegt die Annahme zugrunde, dass je weniger Augen einen Quellcode sehen, umso unwahrscheinlicher es ist, dass sicherheitsrelevante Fehler gefunden werden. Diese Argumentation ist gefährlich. Auf der einen

[10] Solche Kriterienkataloge sind z.B. die *Trusted Computer Security Evaluation Criteria* (TCSEC) in den USA, die *Information Technology Security Evaluation Criteria* (ITSEC) in Europa oder die Common Criteria auf internationaler Ebene.

[11] Was ein "sicheres" Betriebssystem genau ist, variiert von Kriterienkatalog zu Kriterienkatalog.

Seite ist es immer möglich, dass sich nicht autorisierte Kreise Zugang zum Quellcode verschaffen können, und auf der anderen Seite ist die Zahl der Augen immer ein schlechtes Indiz für die Ernsthaftigkeit einer Fehlersuche. Das Meiste, das im Zusammenhang mit dem "Viele Augen"-Argument in Abschnitt 4.1 gesagt worden ist, gilt — mit umgekehrtem Vorzeichen — auch für das "Wenige Augen"-Argument. Entsprechend widersprechen und neutralisieren sich die beiden Argumente bis zu einem gewissen Punkt.

5.3 Das Geheimhaltungsargument

Das "Wenige-Augen"-Argument ist eng verwandt mit dem Geheimhaltungsargument, das im Wesentlichen besagt, dass es einfacher sei, die Arbeits- und Funktionsweise von Closed Source Software geheimzuhalten, als die von Open Source Software[12].

Wenn es darum geht, die Arbeits- und Funktionsweise von Software geheimzuhalten, dann ist das Geheimhaltungsargument sicherlich richtig und stichhaltig. Ob die Geheimhaltung der Arbeits- und Funktionsweise aber der richtige Ansatz ist, um die Sicherheit von Software zu verbessern, ist fraglich. In Abschnitt 2 ist argumentiert worden, dass Closed Source Software nicht notwendigerweise "security through obscurity" impliziert. Wenn ein Hersteller allerdings versucht, durch eine bewusste Nichtoffenlegung von Quellcode die Arbeits- und Funktionsweise von Software geheimzuhalten, dann befindet er sich schon in der gefährlichen Welt von "security through obscurity". Grundsätzlich sollte die Arbeits- und Funktionsweise von Software bekannt sein und (auch öffentlich) diskutiert werden können. In diesem Fall verliert das Geheimhaltungsargument seine Bedeutung.

5.4 Das Verantwortungs- und Haftungsargument

Die Verfechter von Closed Source Software behaupten, dass für Open Source Software niemand verantwortlich sei, dass entsprechend auch niemand hafte, und dass die explizite Berücksichtigung von Verantwortungs- und Haftungsfragen in konventionellen Softwarelizenzverträgen ein wichtiges Argument für Closed Source Software darstelle.

Dieses Verantwortungs- und Haftungsargument trifft wohl zu. In den meisten Lizenzverträgen für Open Source Software sind allfällige Haftungsansprüche ausbedungen (eine genaue Analyse findet sich z.B. in [Spi03]). Allerdings gilt es in diesem Zusammenhang auch zu beachten, dass die Lizenzverträge für Closed Source Software oft nicht viel besser sind, und dass die entsprechende Produktehaftung meist minimal ist. Ein wichtiger Vorteil von konventionellen Softwarelizenzverträgen gegenüber vielen Lizenzverträgen für Open Source Software betrifft den Schutz vor Schadensersatzforderungen für Schutzverletzungen von geistigem Eigentum (z.B. Patente, Urheberrechte, etc.). Lizenzverträge für Open Source Software beinhalten meist keinen deratigen Schutz, d.h. der Lizenznehmer haftet hier für derartige Schutzverletzungen.

Zusammenfassend kann man sagen, dass das Haftungsargument zwar zutrifft, dass es aber im Einzelfall (auf der Basis der Lizenzverträge und -vereinbarungen) überprüft werden muss, ob und wie weit die Haftungsansprüche für Closed Source Software wirklich gehen.

[12] Es gibt Verfechter von Closed Source Software, die den Quellcode ihrer Software nur deshalb nicht veröffentlichen, weil sie fürchten, von Konkurrenten in bezug auf Verletzung von Patentansprüchen verklagt zu werden.

5.5 Das "bessere Dokumentation und Unterstützungs"-Argument

Schliesslich behaupten die Verfechter von Closed Source Software zuweilen auch, dass Open Source Software schlecht dokumentiert sei, und dass für diese Art von Software in der Regel keine oder nur schlechte Unterstützung angeboten werde. Demgegenüber würde Closed Source Software von kommerziellen Softwarefirmen hergestellt und lizenziert, und entsprechend würden auch Dokumentation und Unterstützung professionell gehandhabt.

Das "bessere Dokumentation und Unterstützungs"-Argument ist im Unterstützungsbereich sicherlich zutreffend (die meisten Softwarehersteller bieten bei Problemen professionelle Unterstützung an). In diesem Zusammenhang ist insbesondere auch zu erwähnen, dass die meisten Hersteller von Closed Source Software sogenannte "Roadmaps" haben, die die geplanten und zu erwartenden Entwicklungen in den nächsten paar Jahren aufzeigen. Solche Informationen sind im Bereich von Open Source Software meist nicht verfügbar (insbesondere weil es niemanden gibt, der autorisiert ist, eine solche Roadmap zu erstellen).

Im Dokumentationsbereich muss darauf hingewiesen werden, dass es auch für Closed Source Software einen grossen Markt für Sekundärliteratur gibt, und dass in diesem Bereich zwischen Open und Closed Source Software kein wesentlicher Unterschied mehr existiert. Die FSF hat z.B. auch eine *GNU Free Documentation License* (GNU FDL) definiert[13], die es den Autoren von Dokumentationsmaterial erlaubt, dieses Material einer ähnlichen Lizenz zu unterstellen, wie Open Source Software. So gibt es bereits ein paar grössere Projekte, die auf der Basis der GNU FDL basieren (z.B. die freie Enzyklopädie Wikipedia[14]). Ob sich die GNU FDL in der Praxis allerdings durchsetzt, bleibt abzuwarten.

6 Schlussfolgerungen und Ausblick

Aufgrund der Ausführungen in den Abschnitten 4 und 5 muss man schliessen, dass viele der vorgebrachten Argumente für Open und Closed Source Software entweder nicht relevant sind oder einer seriösen Betrachtung nicht standhalten, dass sie zum Teil auf falschen oder nicht realistischen Annahmen beruhen, oder dass sie andersweitig illusorisch sind. Stattdessen gilt es festzuhalten, dass die Offenlegung von Quellcode keinen nachweisbaren Einfluss auf die Sicherheitseigenschaften einer Software hat, und dass die Verfügbarkeit von Quellcode entsprechend auch nicht als Sicherheitskriterium herangezogen werden kann. Diese Erkenntnis wird auch dadurch bekräftigt, dass es auf dem Markt sichere und unsichere Open Source Software gibt, genauso wie es sichere und unsichere Closed Source Software gibt.

In [Spa02] kommt Gene Spafford vom Center for Education and Research in Information Assurance and Security (CERIAS) der Purdue University zu einem ähnlichen Schluss: dass die Frage, ob Open oder Closed Source Software sicherer ist, in dieser (allgemeinen) Form nicht beantwortet werden kann. Anstelle der Offenlegung von Quellcode gibt er eine Reihe von anderen Erfolgsfaktoren an, die die Sicherheit von Software nachhaltig positiv beeinflussen können. Viele dieser Faktoren stammen aus dem Software Engineering und gehen auf eine seriöse und professionelle Softwareentwicklung zurück. So ist es z.B. erforderlich,

- dass der Entwurf vollständig und konsistent ist,
- dass sich die Softwareentwickler in den zugrunde liegenden (Sicherheits-) Technologien gut auskennen und entsprechend geschult sind,

[13] `http://www.fsf.org/licenses/fdl.html`
[14] `http://www.wikipedia.org/`

- dass qualitativ hochstehene Softwareentwicklungswerkzeuge eingesetzt werden,
- dass einmal entwickelte Softwaremodule seriös und umfassend getestet werden,
- dass der Komplexität der Benutzerschnittstelle besondere Beachtung geschenkt wird, und
- dass die Behebung von Sicherheitslücken ("Patchen") professionell und auf eine institutionalisierte Art und Weise angegangen wird.

Im Hinblick auf den letztgenannten Punkt ist eine professionelle Information und Instruktion der Benutzer unumgänglich. Alle Punkte haben zwar einen grossen Einfluss auf die Sicherheitseigenschaften von Software, haben aber wenig damit zu tun, ob der Quellcode offengelegt ist[15].

Natürlich sind die Diskussionen um Open Source Software und entsprechende Lizenzierungsmodelle nicht unwichtig, nur weil sie sicherheitstechnisch nicht (sinnvoll) geführt werden können. Die Diskussionen sind wichtig und müssen in der erforderlichen Tiefe auch geführt werden. Die Diskussionen sind aber wirtschaftlich motiviert und müssen sich entsprechend auch auf Wirtschaftlichkeitsargumente beziehen. Den Nutzer von Software interessieren letztlich nur die Vollkosten[16], die sich aus den Lizenzgebühren und den Unterhaltskosten der Software zusammensetzen. Von diesen Komponenten sind nur die Lizenzgebühren bekannt und im voraus kalkulierbar. Für die Unterhaltskosten von Open Source Software fehlt bis heute statistisch relevantes Datenmaterial. Entsprechend sind Vollkostenvergleiche aus heutiger Sicht immer spekulativ. In ein paar Jahren, wenn die ersten Vollkostenrechnungen für Open Source Software auf dem Tisch liegen, wird man mehr wissen und erste Schlussfolgerungen ziehen können. In der Zwischenzeit bleibt zu hoffen, dass sich die Politik in dieser Frage neutral verhält und nicht beginnt, mit volkswirtschaftlich fragwürdigen Investitionsprogrammen den Wettbewerb zu verzerren.

A Referenzen

[**And01**] R. Anderson, "Why Information Security is Hard—An Economic Perspective," *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC 2001)*, Dezember 2001, Seiten 358–365
`http://www.ftp.cl.cam.ac.uk/ftp/users/rja14/econ.pdf`

[**And02**] R. Anderson, "Security in Open versus Closed Systems — The Dance of Boltzmann, Coase and Moore," *Proceedings of Open Source Software: Economics, Law and Policy*, 20.-21. Juni 2002
`http://www.ftp.cl.cam.ac.uk/ftp/users/rja14/toulouse.pdf`

[**And03**] R. Anderson, *Open and Closed Systems are Equivalent (that is, in an ideal world)*
`http://www.ftp.cl.cam.ac.uk/ftp/users/rja14/toulousebook.pdf`

[**BAB99**] R.M. Brady, R. Anderson und R.C. Ball, *Murphy's law, the fitness of evolving species, and the limits of software reliability*, Technical Report UCAM-CL-TR-471, University of Cambridge, Computer Laboratory, United Kingdom, September 1999
`http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-471.pdf`

[**BB96**] P.G. Bishop und R.E. Bloomfield, "A Conservative Theory for Long-Term Reliability Growth Prediction," *IEEE Transactions on Reliability*, Vol. 45 (1996), No 4., Seiten 550–560
`http://www.adelard.co.uk/resources/papers/pdf/issre96m.pdf`

[**BF93**] R.W. Butler und G.B. Finelli, "The infeasibility of experimental quantification of life-critical software reliability," *IEEE Transactions on Software Engineering*, Vol. 19, No. 1, Januar 1993, Seiten 3–12
`http://techreports.larc.nasa.gov/ltrs/PDF/NASA-91-acm-rwb.pdf`

[15] Zu einem ähnlichen Schluss kommt auch Peter G. Neumann [Neu00] und die Autoren von [MMP00].

[16] Man spricht in diesem Zusammenhang von *Total Cost of Ownership* (TCO).

[**Bro02**] K. Brown, *Opening the Open Source Debate: A White Paper*, Alexis de Tocqueville Institution, Juni 2002
`http://www.adti.net/html_files/defense/opensource_debate.html`

[**Ker83**] A. Kerckhoffs, "La cryptographie militaire," *Journal des sciences militaires*, Vol. IX, Januar 1883, Seiten 5-38, Februar 1883, Seiten 161-191
`http://www.cl.cam.ac.uk/users/fapp2/kerckhoffs`

[**Lyu96**] M.R. Lyu (Ed.), *Handbook of Software Reliability Engineering*, McGraw Hill, 1996

[**MMP00**] K. Köhntopp, M. Köhntopp und A. Pfitzmann, "Sicherheit durch Open Source? Chancen und Grenzen," *Datenschutz und Datensicherheit (DuD)*, Vol. 24, No. 9, September 2000, Seiten 508–513
`http://www.semper.org/sirene/publ/KoeKP_00.pdf`

[**Neu00**] P.G. Neumann, "Robust Nonproprietary Software," *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, Mai 2000
`http://www.csl.sri.com/users/neumann/ieee00+.pdf`

[**Opp01**] R. Oppliger, *Secure Messaging with PGP and S/MIME*, Artech House, Norwood, MA, 2001

[**Sch99**] B. Schneier, *Open Source and Security*, Crypto-Gram Newsletter, September 1999
`http://www.counterpane.com/crypto-gram-9909.html#OpenSourceandSecurity`

[**Sch00**] B. Schneier, *Secrets & Lies: Digital Security in a Networked World*, John Wiley & Sons, 2000

[**Spa02**] G. Spafford, *Is Open Source More Secure*
`http://www.techtv.com/screensavers/linux/story/0,24330,3406300,00.html`

[**Spi03**] G. Spindler, "Rechtsfragen der Open Source Software", Universität Göttingen, erscheint in *Open Source*, Verlag Otto Schmidt, Köln, 2003
`http://www.vsi.de/inhalte/aktuell/studie_final.pdf`

[**Tho84**] K. Thompson, "Reflections on Trusting Trust," *Communications of the ACM*, Vol. 27, No. 8, August 1984, Seiten 761–763
`http://www.acm.org/classics/sep95/`

[**VF02**] J. Viega und B. Fleck, *Dispelling Myths about the GPL and Free Software*, Cyberspace Policy Institute, 2002
`http://www.cpi.seas.gwu.edu/oss/cpi_rebuttal.pdf`