UNIVERSITÄT LEIPZIG
Fakultät für Mathematik und Informatik
Institut für Informatik

Sicherheit von Open Source Software

Wie sicher ist Open Source Software?

Lukas Kairies

Seminararbeit

Leipzig, März 2015

Inhaltsverzeichnis

1 Einleitung

Open Source Software spielt eine entscheidende Rolle in der Entstehung und Weiterentwicklung des Internets und ist in vielen sicherheitskritischen Bereichen weit verbreitet. Beispielsweise liefen laut Netcraft 2014 ca. 50% aller aktiven Webseiten im Internet auf der Open Source Software „Apache Http Server“.[1] Des weiteren wird Open Source Software wie OpenSSL zur Kommunikationsverschlüsslung im Internet und GnuPG zur Verschlüsselung von E-mails verwendet. Ein aktueller Anlass für die Diskussion über die Sicherheit von Open Source Software ist der 2014 aufgetretene Heartbleed Bug. Der Heartbleed Bug in der Verschlüsselungsbibliothek OpenSSL wird als eine der schwerwiegendste Sicherheitslücken die je auftrat bezeichnet und hatte weitreichende Folgen für die gesamte Internetinfrastruktur.[2]

Die Diskussion über die Sicherheit von Open Source Software wird oftmals als eine ideologische Diskussion über den überlegenden Entwicklungsprozess und der öffentlichen Verfügbarkeit des Quellcodes von Open Source Software und den von wirtschaftlichen Interessen geprägten Entwicklungsprozess proprietärer Software verstanden. Dabei spielt der Mythos von sicherer Open Source Software eine wichtige Rolle. In dieser Arbeit soll der Frage nach der Sicherheit von Open Source Software nachgegangen werden. Dabei werden verschiedene Argumente von Befürwortern und Gegnern von Open Source Software gegenüber gestellt.

Dabei wird die These aufgestellt, dass es grundsätzlich keinen Unterschied zwischen der Sicherheit in Open Source Software und proprietärer Software gibt.

In dieser Arbeit sollen zunächst die verwendeten Begriffe definiert werden, um eine allgemeine Basis für das Verständnis der Arbeit zu schaffen. Dabei werden unter anderem die Begriffe Open Source Software und Freie Software definiert und voneinander abgegrenzt. Anschließend werden die Entwicklungs- und Sicherheitskonzepte von Open Source und proprietärer Software beschrieben. Im Hauptteil der Arbeit soll das Fallbeispiel des Heartbleed Bugs als Beispiel für eine kritische Sicher-

[1] Netcraft (2014). *Half a million widely trusted websites vulnerable to Heartbleed bug*. Zuletzt besucht am 30.03.2015. URL: http://news.netcraft.com/archives/2014/05/07/may-2014-web-server-survey.html.

[2] Hanno Böck (2014). *Wichtige Fragen und Antworten zu Heartbleed*. Zuletzt besucht am 30.03.2015. URL: http://www.golem.de/news/openssl-wichtige-fragen-und-antworten-zu-heartbleed-1404-105740.html.

heitslücke in offener Software und dessen Auswirkungen betrachtet werden und die Sicherheit von Open Source Software und proprietärer Software verglichen werden. Rückschließend auf die aufgestellte These werden zum Abschluss dieser Arbeit die erarbeiteten Ergebnisse in einem Fazit präzise zusammengefasst und es wird ein Ausblick über verwandte Themen gegeben.

2 Begriffsklärung

In diesem Abschnitt sollen die grundlegenden Begriffe definiert werden. Zunächst sollen die Begriffe Freie Software und Open Source Software eingeführt und voneinander abgegrenzt werden. Der Fokus soll schließlich aber auf Open Source Software gelegt werden, da diese die wesentlichen Eigenschaften zum Bearbeiten der These enthält. Des weiteren werden die Begriffe proprietäre Software und Informationssicherheit in Bezug auf Software definiert.

2.1 Freie Software

Der Begriff der Freien Software (Englisch: Free Software) wurde maßgeblich von Richard Stallmann und der von ihm 1985 gegründeten Free Software Foundation (FSF)[3] geprägt. In dieser Definition werden dem Nutzer von Freier Software bestimmte Freiheiten eingeräumt, die im Weiteren näher definiert werden[4]:

Freiheit 0: Die Freiheit, das Programm auszuführen wie man möchte, für jeden Zweck.

Freiheit 1: Die Freiheit, die Funktionsweise des Programms zu untersuchen und eigenen Bedürfnissen der Datenverarbeitung anzupassen.

Freiheit 2: Die Freiheit, das Programm weiterzuverbreiten und damit seinen Mitmenschen zu helfen.

Freiheit 3: Die Freiheit, das Programm zu verbessern und diese Verbesserungen der Öffentlichkeit freizugeben, damit die gesamte Gemeinschaft davon profitiert.

[3] Free Software Foundation (2015b). *Free Software Foundation*. Zuletzt besucht am 28.03.2015. URL: `https://fsf.org/`.

[4] Free Software Foundation (2015a). *Die Freie Software Definition*. 28.03.2015. URL: `http://www.gnu.org/philosophy/free-sw.de.html`.

Die Freiheiten 1 und 3 setzten den öffentlichen Zugang zum Quellcode der Software voraus, sodass Software, dessen Quellcode nicht öffentlich zugänglich ist als *unfrei* zu bezeichnen ist.[5] Software dessen Lizenz die Freiheiten des Nutzers nicht adäquat umsetzt wird von der FSF als unfrei und damit als unethisch betrachtet. Wenn nicht der Nutzer, sondern der Entwickler die Kontrolle über die Software hat, wie es bei unfreier bzw. proprietärer Software der Fall ist, ist sie ein Instrument ungerechter Macht.[6].

Um die Freiheiten des Nutzers rechtlich zu schützen nutzt die FSF sogenannte Copy-Left Lizenzen, wie etwa die GNU GPL Lizenz[7]. In diesen Lizenzen wird gesichert, das Software, die auf Freier Software aufbaut, die Freiheiten des Nutzers nicht einschränkt. Freie Software muss aber nicht zwangsläufig unter einer Copy-Left Lizenz stehen. Software die unter Public-Domain (Gemeinfreiheit) oder freizügigeren Lizenzen wie der BSD-, Apache- oder MIT-Lizenz stehen, können ebenfalls als frei bezeichnet werden.[8] Des weiteren grenzt die FSF die Bedeutungen des englischen Wortes *free* voneinander ab: *„Um das Konzept zu verstehen, sollten man an frei wie in Redefreiheit denken, nicht wie in Freibier."*[9]

Das Wort *free* lässt sich sowohl mit frei als auch gratis übersetzen. Die FSF bezieht sich auf die Bedeutung *frei*, um deutlich zu machen, dass es sich bei Freier Software nicht zwangsläufig um kostenlose Software handelt, sondern die Freiheit des Nutzers im Vordergrund steht. Um dies zu verdeutlichen wird alternativ auch der Begriff Libre Software verwendet.[10]

[5] Free Software Foundation (2015a). *Die Freie Software Definition*. 28.03.2015. URL: http://www.gnu.org/philosophy/free-sw.de.html.

[6] Free Software Foundation (2015a). *Die Freie Software Definition*. 28.03.2015. URL: http://www.gnu.org/philosophy/free-sw.de.html.

[7] Free Software Foundation (2007). *Gnu General Public License*. Zuletzt besucht am 28.03.2015. URL: https://www.gnu.org/copyleft/gpl.html.

[8] Open Source Initiative. *Open Source Licenses*. Zuletzt besucht am 28.03.2015. URL: http://opensource.org/licenses.

[9] Free Software Foundation (2015a). *Die Freie Software Definition*. 28.03.2015. URL: http://www.gnu.org/philosophy/free-sw.de.html.

[10] Free Software Foundation (2015a). *Die Freie Software Definition*. 28.03.2015. URL: http://www.gnu.org/philosophy/free-sw.de.html.

2.2 Open Source Software

Der Begriff Open Source Software wurde erstmals von der Open Source Initiative (OSI)[11] definiert. Die Definition umfasst folgende Punkte[12]

Freie Weitergabe Die Lizenz darf niemanden darin hindern, das Programm zu verkaufen oder es mit anderen Programmen in einer Software-Distribution weiterzugeben. Die Lizenz darf keine Lizenzgebühren verlangen.

Verfügbarer Quellcode Das Programm muss im Quellcode für alle Nutzer verfügbar sein.

Abgeleitete Arbeiten Die Lizenz muss vom Basisprogramm abgeleitete Arbeiten und deren Distribution unter derselben Lizenz wie das Basisprogramm erlauben.

Integrität des Autoren Quellcodes Die Lizenz muss explizit das Verteilen eines Programms erlauben, das auf einer veränderten Version des Originalquellcodes beruht. Die Lizenz kann verlangen, dass solche Änderungen zu einem neuen Namen oder einer neuen Versionsnummer des Programms führen und die Änderungen dokumentiert werden. Die Lizenz darf verlangen, dass nur Patches zum Originalcode verteilt werden dürfen, wenn diese mit dem Quellcode verteilt werden dürfen.

Keine Diskriminierungen von Personen oder Gruppen Die Lizenz darf einzelnen Personen oder Gruppen die Nutzung der Software nicht untersagen.

Keine Nutzungseinschränkung Die Lizenz darf den Verwendungszweck des Programms nicht einschränken.

Lizenzerteilung Die Lizenz muss für alle zutreffen, welche das Programm erhalten, ohne z. B. eine Registrierung oder eine andere Lizenz erwerben zu müssen.

Produktneutralität Die Lizenz muss produktneutral gestaltet sein und darf sich z. B. nicht ausschließlich auf eine bestimmte Distribution beziehen.

[11] Open Source Initiative. *Open Source Initiative*. Zuletzt besucht am 28.03.2015. URL: `http://opensource.org`.

[12] Open Source Initiative. *The Open Source Definition*. Zuletzt besucht am 28.03.2015. URL: `http://opensource.org/osd`.

Die Lizenz darf andere Software nicht einschränken Die Lizenz darf bspw. nicht verlangen, dass sie nur mit Open Source Software verbreitet werden darf.

Die Lizenz muss Technologie-neutral sein Sie darf bspw. nicht verlangen, dass die Distribution nur auf CD/DVD verteilt werden darf.

Der Begriff Open Source Software soll den praktischen Nutzen und die Überlegenheit des Entwicklungsprozesses hervorheben. Er wurde einführt um die Vermarktung von Open Source zu vereinfachen und so potenzielle Wirtschaftspartner zu gewinnen.[13] Dies stellt einen deutlichen Kontrast zu den moralischen Werten der Freien Software dar. Aus diesem Grund lehnt die FSF den Begriff Open Source Software grundsätzlich ab, obwohl die Definitionen im wesentlichen gleich sind[14]. Im weiteren Teil der Arbeit soll Open Source Software beide Begriffe, also Open Source Software und Freie Software beinhalten, da die unterschiedlichen Anschauungen keinen Mehrwert für die Bearbeitung der These bringen.

2.3 Proprietäre Software

Proprietäre Software gilt als Gegenmodell zu freier bzw. Open Source Software. Bei proprietärer Software ist der Quellcode nicht frei zugänglich und sie basiert auf herstellerspezifischen, i.d.R. nicht offenen Standards.[15] Meist wird auch die Veränderung und Weiterverbreitung der Software durch die entsprechenden Lizenzen untersagt. Hersteller von proprietärer Software haben im allgemeinen drei Möglichkeiten ihre Software zu schützen: Softwarepatente, Urheberrecht und Closed Source.[16] Im Fall von Closed Source wird der Quellcode als Betriebsgeheimnis behandelt und nicht öffentlich zugänglich gemacht. Im weiteren Teil der Arbeit ist mit proprietäre Software immer Software mit geschlossener Quelle gemeint.

[13] Georg C.F. Greve (2015). *Referenzblatt zu den Grundlagen Freier Software*. Zuletzt besucht am 28.03.2015. URL: https://fsfe.org/activities/wipo/fser.de.html.

[14] Richard Stallmann (2014). *Warum Open Source das Ziel von Freie Software verfehlt*. Zuletzt besucht am 28.03.2015. URL: https://www.gnu.org/philosophy/open-source-misses-the-point.

[15] ubuntuusers (2011). *unfreie Software*. Zuletzt besucht am 30.03.2015. URL: http://wiki.ubuntuusers.de/unfreie_Software?rev=231304.

[16] Stephen Donovan (1994). „Patent, copyright and trade secret protection for software“. In: *Potentials, IEEE* 13.3, S. 20–24. ISSN: 0278-6648.

2.4 Informationssicherheit

Informationssicherheit bezeichnet Eigenschaften von informationsverarbeitenden und informationslagernden Systemen.[17] Dabei kann es sich um technische und nichttechnische Systeme handeln. Da sich diese Arbeit aber ausschließlich mit Sicherheit von Software beschäftigt, soll nachfolgend nur die technische Seite betrachtet werden. Alternativ wird auch der Begriff IT-Sicherheit verwendet, um den Bezug zu informationstechnischen Systemen zu verdeutlichen.[18] Ziel von Informationssicherheit sind neben dem gezielten Absichern gegen vorher festgelegte Angriffsmöglichkeiten, das Erschweren von unberechtigten Zugriffen, so dass der Aufwand für das Kompromittieren eines Systems in einem ungünstigen Verhältnis zum möglichen Informationsgewinn für einen potenziellen Angreifer steht.[19] Mit Informationssicherheit sollen des weiteren Schutzziele eingehalten werden, die sich wie folgt definieren lassen:[20]:

Vertraulichkeit Vertrauliche Informationen müssen vor Unbefugten geschützt werden.

Integrität: Daten dürfen nicht unbemerkt verändert werden. Alle Änderungen müssen nachvollziehbar sein.

Verfügbarkeit: Benutzern müssen Dienstleistungen, Funktionen eines IT-Systems und Informationen zum gewünschten Zeitpunkt zu Verfügung stehen.

Nichtabstreitbarkeit: Die Nichtabstreitbarkeit umfasst sowohl das Nichtabstreiten der Herkunft, als auch das Nichtabstreiten des Erhalts. Absendern soll es dadurch unmöglich sein, das Absenden

[17] Bundesamt für Sicherheit in der Informationstechnik (2013). *IT-Grundschutz-Kataloge*. Zuletzt besucht am 30.03.2015. URL: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/Glossar/glossar_node.html.

[18] Bundesamt für Sicherheit in der Informationstechnik (2013). *IT-Grundschutz-Kataloge*. Zuletzt besucht am 30.03.2015. URL: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKataloge/Inhalt/Glossar/glossar_node.html.

[19] Lutz Donnerhacke (2015). *de.comp.security.firewall FAQ*. URL: http://altlasten.lutz.donnerhacke.de/mitarb/lutz/usenet/Firewall.html#Sicherheit.

[20] Bundesamt für Sicherheit in der Informationstechnik – BSI (2012). Techn. Ber. Bundesamt für Sicherheit in der Informationstechnik – BSI. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/Leitfaden/GS-Leitfaden_pdf.pdf?__blob=publicationFile, S. 14.

einer Nachricht nachträglich zu bestreiten und Empfängern unmöglich sein, das Empfangen einer Nachricht nachträglich zu bestreiten.

Neben den drei Hauptschutzzielen lassen sich je nach Anwendungsfall noch weitere Schutzziele definieren, wie etwa Datenschutz oder Datensicherheit.[21]

3 Entwicklungs- und Sicherheitskonzepte

In diesem Abschnitt werden die verschiedenen Entwicklungs- und Sicherheitskonzepte von Open Source- und proprietärer Software unabhängig voneinander betrachtet.

3.1 Konzept von Open Source Software

Wie bereits in Abschnitt 2.2 beschrieben, ist das öffentliche Bereitstellen des Quellcodes ein zentrales Konzept von Open Source Software. Durch diese Bereitstellung ist es jedem möglich den Quellcode auf dessen Qualität und mögliche Fehler zu untersuchen, sodass von einem öffentlichen Reviewprozess gesprochen werden kann. In der Praxis ist dies nur eingeschränkt möglich, da hierfür ein gewissen technisches Verständnis nötig ist, so dass i.d.R. nur versierte Nutzer und Experten dies auch tatsächlich tun.[22] Durch den vermehrten Einsatz von Open Source Software in der Wirtschaft und Industrie ist eine weitere Partei in diesen Prozess eingebunden, so dass neben den Entwicklern eines Open Source Projektes auch Unternehmen Interesse an sicherer und qualitativ hochwertiger Open Source Software haben.[23]

Durch dieses Viele-Augen-Prinzip soll die Sicherheit und Fehlerfreiheit von Open Source Software

[21] Bundesamt für Sicherheit in der Informationstechnik – BSI (2012). Techn. Ber. Bundesamt für Sicherheit in der Informationstechnik – BSI. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/Leitfaden/GS-Leitfaden_pdf.pdf?__blob=publicationFile, S. 14.

[22] Birol Kayapinar Wolfram Riedel Dincel Atac (2004). *Open Source Security*. Zuletzt besucht am 30.03.2015. URL: http://ig.cs.tu-berlin.de/oldstatic/w2003/ir1/uebref/AtacEtAl-Referat-G12-122003.pdf.

[23] Dirk Riehle Amit Deshpande (2008). *The Total Growth of Open Source*. Zuletzt besucht am 30.03.2015. URL: http://dirkriehle.com/publications/2008-2/the-total-growth-of-open-source/.

gesichert werden und es wird angenommen, dass umso mehr Personen an dem öffentlichen Reviewprozess teilnehmen, desto hochwertiger und sicherer ist eine Open Source Software.[24]
Ein weiterer wichtiger Aspekt in der Entwicklung von Open Source Software ist die geringe Einstiegshürde für das Teilnehmen an Projekten. Für Nutzer soll es möglichst einfach sein Änderungen zur Fehlerbehebung oder Erweiterung der Funktionalität einzureichen. Um die Qualität solcher Änderungen sicherzustellen haben sich innerhalb der Open Source Communtiy gewisse Verfahren bewährt. Dabei durchläuft jede Änderung i.d.R. einen Prozess, in dem die Änderung zunächst mittels eines Patches eingereicht wird. Ein Patch beinhaltet dabei nur den geänderten Codes. Anschließend wird die Änderung öffentlich diskutiert. Entweder wird die Änderung angenommen, Verbesserungen an ihr vorgenommen, bis die Qualität sichergestellt ist, oder sie wird abgelehnt. Sollte die Änderung angenommen werden, wird im Letzten Schritt die Änderung von einem Verantwortlichen des Projektes in den Quellcode eingepflegt und in der nächsten Version der Software veröffentlicht.[25]
Dieser Prozess wird als Peer-Review bezeichnet. Im besten Fall nehmen an diesem Prozess nicht nur die Entwickler des Projektes teil, sondern auch unabhängige Entwickler. Durch dieses Verfahren soll sichergestellt werden, dass nur relevante Änderungen am Quellcode vorgenommen werden und keine Fehler in den Quellcode eingepflegt werden, welche die Sicherheit der Software gefährden könnten.[26]

3.2 Konzept von proprietäre Software

Die Entwicklung von proprietärer Software orientiert sich stark an dem wirtschaftlichem Interesse eines Unternehmens. Bevor eine Software geschrieben wird, werden Marktanalysen durchgeführt und eine entsprechende Anforderungsanalyse formuliert. Der gesamte Entwicklungsprozess untersteht einem Budget- und Zeitplan, so dass die rechtzeitige Fertigstellung und das Einhalten des Budgets

[24] Birol Kayapinar Wolfram Riedel Dincel Atac (2004). *Open Source Security*. Zuletzt besucht am 30.03.2015. URL: http://ig.cs.tu-berlin.de/oldstatic/w2003/ir1/uebref/AtacEtAl-Referat-G12-122003.pdf.

[25] Birol Kayapinar Wolfram Riedel Dincel Atac (2004). *Open Source Security*. Zuletzt besucht am 30.03.2015. URL: http://ig.cs.tu-berlin.de/oldstatic/w2003/ir1/uebref/AtacEtAl-Referat-G12-122003.pdf.

[26] Birol Kayapinar Wolfram Riedel Dincel Atac (2004). *Open Source Security*. Zuletzt besucht am 30.03.2015. URL: http://ig.cs.tu-berlin.de/oldstatic/w2003/ir1/uebref/AtacEtAl-Referat-G12-122003.pdf.

wichtiger sein kann als die fehlerfreie Implementierung des Programms.[27].
Wie in Abschnitt 2.3 beschrieben wird der Quellcode von proprietärer Software als Geschäftsgeheimnis behandelt und dementsprechend nicht veröffentlicht. Neben dem Waren von Geschäftsgeheimnissen wird das Geheimhalten des Quellcodes auch oft als Sicherheitskonzept verstanden.
Bei dem *Security through Obscurity*-Sicherheitskonzept (Sicherheit durch Verschleierung) von proprietärer Software ist die Annahme, das durch die Geheimhaltung des Quellcodes und damit aller möglichen Schwachstellen es Angreifern erschwert bzw. unmöglich ist diese auszunutzen. So soll die Sicherheit eines Programms gewährleistet sein.[28] Tatsächlich erschwert diese Methode das Auffinden von Schwachstellen, macht es aber nicht unmöglich. Es gibt eine Vielzahl von Methoden das Verhalten eines Programms zu analysieren um Teile des Quellcodes durch sogenanntes *reverse engineering* zu rekonstruieren, sodass letztendlich nur die Einstiegshürde für Angreifer höher ist und auch in proprietärer Software mit angemessenen zeitlichen Aufwand Fehler auffindbar sind. Software deren Sicherheit auf dem Verbergen des Quellcodes beruht kann daher nicht als sicher angesehen werden.[29]

4 Heartbleed Bug

In diesem Abschnitt wird als aktuelles Beispiel für die Sicherheitsproblematik in Open Source Software der Heartbleed behandelt. Dazu werden der Hintergrund, die Funktionsweise und die Folgen der in OpenSSl auftretenden Sicherheitslücke beschrieben.

[27] Birol Kayapinar Wolfram Riedel Dincel Atac (2004). *Open Source Security*. Zuletzt besucht am 30.03.2015. URL: http://ig.cs.tu-berlin.de/oldstatic/w2003/ir1/uebref/AtacEtAl-Referat-G12-122003.pdf.

[28] Birol Kayapinar Wolfram Riedel Dincel Atac (2004). *Open Source Security*. Zuletzt besucht am 30.03.2015. URL: http://ig.cs.tu-berlin.de/oldstatic/w2003/ir1/uebref/AtacEtAl-Referat-G12-122003.pdf.

[29] Birol Kayapinar Wolfram Riedel Dincel Atac (2004). *Open Source Security*. Zuletzt besucht am 30.03.2015. URL: http://ig.cs.tu-berlin.de/oldstatic/w2003/ir1/uebref/AtacEtAl-Referat-G12-122003.pdf.

4.1 Hintergrund

OpenSSL ist ein Open Source Werkzeug zur verschlüsselten Kommunikation im Internet und implementiert das von der Internet Engineering Task Force (IETF) standardisierte Verschlüsselungsprotokoll Secure Sockets Layer (SSL), welches später in Transport Layer Security (TLS) umbenannt wurde.[30]

Im April 2014 veröffentlichte das OpenSSL-Team einen Sicherheitshinweis für eine kritische Sicherheitslücke in OpenSSL. Diese Lücke wurde als Heartbleed Bug bezeichnet.[31]

Der Heartbleed Bug entstand im Rahmen einer Dissertation eines Informatikstudenten und wurde 31.Dezember 2011 in den Quellcode von OpenSSL aufgenommen und am 14.März 2012 mit der OpenSSL Version 1.0.1 veröffentlicht.[32] Die Sicherheitslücke bestand somit für 27 Monate. Betroffen war allerdings nicht die Grundfunktion von OpenSSL, sondern die Heartbeat Erweiterung, die ebenfalls im Rahmen der Dissertation des Studenten entstanden war. Diese Erweiterung ist nicht Teil des eigentlichen SSL/TLS Standards und betrifft somit nur die Heartbeat Erweiterung von OpenSSL und keine der anderen Implementierungen wie bspw. GnuTLS oder proprietäre Implementierungen wie SChannel (Microsoft) oder RSA BESAFE (RSA Security).[33]

4.2 Funktionsweise

In Abbildung 1 wird die Funktionsweise des Heartbleed Bugs veranschaulicht. Im Normalfall schickt ein Klient eine Heartbeat Anfrage mit einer beliebigen Zeichenkette und deren Länge an den Server. Dieser schickt im fehlerfreien Fall die Zeichenkette zurück. Die Zeichenkette liegt dabei im

[30] OpenSSL Team (2015). *OpenSSL*. Zuletzt besucht am 30.03.2015. URL: `http://dirkriehle.com/publications/2008-2/the-total-growth-of-open-source/`.

[31] Neel Mehta (2014). *CVE-2014-0160*. Zuletzt besucht am 30.03.2015. URL: `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160`.

[32] OpenSSL Team (2014). *OpenSSL Security Advisory [07 Apr 2014]*. Zuletzt besucht am 31.03.2015. URL: `https://www.openssl.org/news/secadv_20140407.txt`.

[33] Fabian A. Scherschel (2014). *So funktioniert der Heartbleed-Exploit*. Zuletzt besucht am 30.03.2015. URL: `http://www.heise.de/security/artikel/So-funktioniert-der-Heartbleed-Exploit-2168010.html`.

[34] Quelle: `http://upload.wikimedia.org/wikipedia/commons/c/cb/Heartbleed_bug_explained.svg`

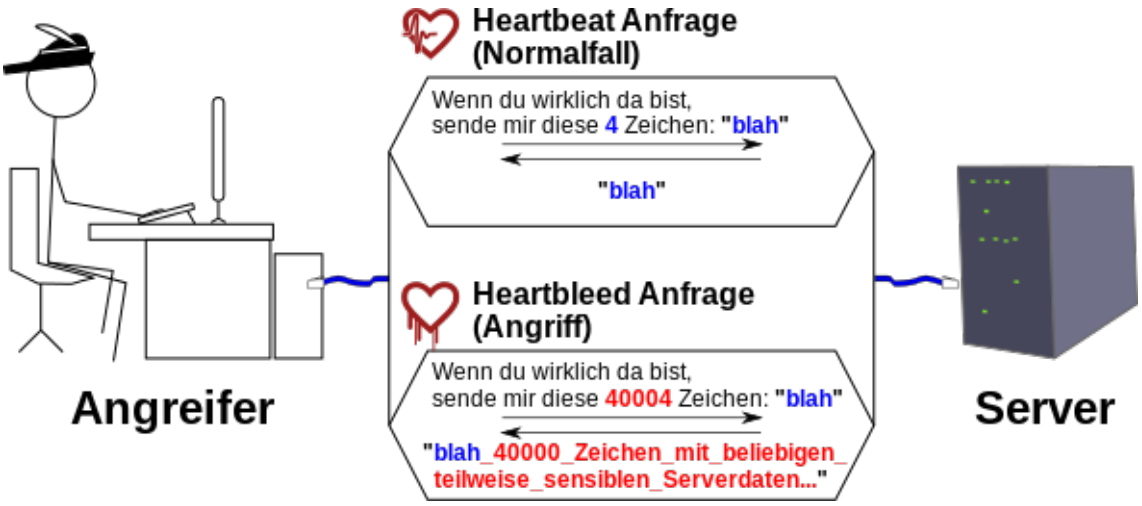

Abbildung 1: Funktionsweise des Heartbleed Bugs[34]

Arbeitsspeicher des Servers.

Ist der Server von dem Heartbleed Bug betroffen und ein Angreifer möchte dies ausnutzen, schickt er wie im Normalfall eine beliebige Zeichenkette, aber anstatt der tatsächlichen Länge der Zeichenkette eine sehr viel größere Zahl. Wie in Abbildung 1 dargestellt, antwortet der Server nun mit der Zeichenkette und den im Arbeitsspeicher dahinter liegenden Daten, bis die vom Angreifer angegeben Länge erreicht ist. So kann ein Angreifer bis zu 64 Kilobyte Daten pro Anfrage vom Server erhalten. Zu den so zugänglich gemachten Daten zählen u.a. Passwörter von Nutzern, Private Schlüssel zur verschlüsselten Kommunikation und Session Cookies, die zur Aufrechterhaltung einer Sitzung zwischen Klient und Server genutzt werden.[35]

[35] Fabian A. Scherschel (2014). *So funktioniert der Heartbleed-Exploit*. Zuletzt besucht am 30.03.2015. URL: `http://www.heise.de/security/artikel/So-funktioniert-der-Heartbleed-Exploit-2168010.html`.

4.3 Folgen

Der Sicherheitsexperte Bruce Schneier vergleicht die Auswirkungen des Heartbleed Bugs mit einer Katastrophe: *„Catastrophic is the right word. On the scale of 1 to 10, this is an 11."*[36] Es finden sich verschiedene quantitative Angaben zur Verbreitung des Heartbleed Bugs. Laut Netcraft waren Schätzungsweise 17,5% aller Webseiten im Internet vom Heartbleed Bug betroffen. Dies führte dazu, dass viele Dienste ihre Nutzer auffordern mussten ihre Passwörter zu ändern und neue Zertifikate zur sicheren Verschlüsselung ausgeteilt werden mussten.[37].

Es ist sehr schwer nachzuverfolgen ob und in welchem Umfang der Heartbleed Bug ausgenutzt wurde, da es keine Beweise für das Ausnutzen des Heartbleed Bug vor seiner Veröffentlichung gibt. Es gibt allerdings Indizien, die darauf hinweisen, dass der Heartbleed Bug durch Geheimdienste und Kriminelle genutzt wurde[38].

Zudem erhielt der Heartbleed große mediale Aufmerksamkeit etwa durch den Spiegel, was zu einer öffentlichen Diskussion führte.[39]

Der Heartbleed Bug ist ein Beispiel dafür, dass der in Abschnitt 1 beschriebene Mythos von sicherer Open Source Software nicht haltbar ist. Trotz der in Abschnitt 3.1 erläuterten Verfahrensweise zur Sicherung der Qualität und Sicherheit von Open Source Software schwerwiegende Sicherheitslücken in Open Source Software auftreten können und das zeitnahe finden und beheben solcher Lücken nicht garantiert ist. Dies wirft viele Fragen innerhalb der Open Source Community und Öffentlichkeit auf, etwa wenn es um die Finanzierung von Open Source Projekten geht.

[36] Bruce Schneier (2014). *Heartbleed.* Zuletzt besucht am 30.03.2015. URL: https://www.schneier.com/blog/archives/2014/04/heartbleed.html.

[37] Torsten Kleinz (2014). *Heartbleed SSL-GAU: Neue Zertifikate braucht das Land.* Zuletzt besucht am 30.03.2015. URL: http://www.heise.de/security/meldung/Heartbleed-SSL-GAU-Neue-Zertifikate-braucht-das-Land-2166639.html.

[38] Peter Eckersley (2014). *Wild at Heart: Were Intelligence Agencies Using Heartbleed in November 2013?* Zuletzt besucht am 30.03.2015. URL: https://www.eff.org/deeplinks/2014/04/wild-heart-were-intelligence-agencies-using-heartbleed-november-2013.

[39] *Sicherheitslücke im Herzen des Internets* (2014). Zuletzt besucht am 30.03.2015. URL: http://www.sueddeutsche.de/digital/heartbleed-bug-in-ssl-verschluesselung-sicherheitsluecke-im-herzen-des-internets-1.1932926.

5 Vergleich der Sicherheit von offener und proprietärer Software

Die Diskussion darüber ob Open Source Software sicherer als geschlossene Software ist, ist von ideologischen Argumenten beider Seiten geprägt. So argumentiert Raymond mit dem in Abschnitt 3.1 genannten Vielen-Augen-Prinzip:*„Given enough eyeballs, all bugs are shallow."*[40]. Seiner Ansicht nach werden Fehler in offenem Quellcode durch Entwickler und Anwender schneller gefunden als bei proprietärer Software, an der i.d.R. nur kleine Entwicklerteams arbeiten.[41]

Unterstützt wird diese Meinung von O'Reilly: *„treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging."*[42] Er merkt allerdings an, dass es nicht *die* eine Open Source Community gibt, die sich mit jedem Projekt gleichermaßen beschäftigt, sondern es an den Projektverantwortlichen liegt ihre Nutzer dazu zu animieren sich an der Fehlersuche und Weiterentwicklung des Projektes zu beteiligen.[43]

Hissam et. al. dagegen relativieren den Vorteil den offener Quellecode zur Sicherheit beträgt.[44] Vor allem Angreifer profitieren von offenen Quellcode und auch das schnelle Ausbessern von Fehlern in offenen Quellen führt in erster Linie dazu, dass Angreifer den Quellcode und die Entwicklungseigenheiten des Projektes besser studieren können.

Des Weiteren stellt Open Source Software eine Gefahr für die Sicherheit von proprietärer Software dar. Proprietäre Software, die auf Open Source Sofware aufbaut, übernimmt mögliche Fehler in der Softwarearchitektur, was die Qualität und Sicherheit der proprietären Software negativ beeinflussen

[40] Eric S. Raymon (1999). *The Cathedral and the Bazaar*. Hrsg. von Tim O'Reilly. 1st. Sebastopol, CA, USA: O'Reilly & Associates, Inc. ISBN: 1565927249, S. 9.

[41] Friedrich-L. Holl Hartmut Heinrich, Jan Tobias Mühlberg Katharina Menzel und Hanno Schüngel Ingo Schäfer (2006). *Open-Source-Software und ihre Bedeutung für Innovatives Handeln*. Zuletzt besucht am 29.03.2015. URL: http://www.bmbf.de/pubRD/oss_studie.pdf.

[42] Tim O'Reilly (2000). *Ten Myths about Open Source Software*. Zuletzt besucht am 29.03.2015. URL: http://www.oreilly.de/opensource/os_artikel/myths_1199.html.

[43] Tim O'Reilly (2000). *Ten Myths about Open Source Software*. Zuletzt besucht am 29.03.2015. URL: http://www.oreilly.de/opensource/os_artikel/myths_1199.html.

[44] Scott A. Hissam, Daniel Plakosh und Charles B. Weinstock (2002). „Trust and vulnerability in open source software." In: *IEE Proceedings - Software* 149.1, S. 47–51. URL: http://dblp.uni-trier.de/db/journals/iee/iee-s149.html#HissamPW02.

kann.[45]

Unterstützt wird diese Meinung von Bezroukov, der kritisiert, dass sich die Open Source Community auf das finden einfacher Fehler im Quellcodes konzentriert und mögliche Fehler in der Softwarearchitektur vernachlässigt. Das Potenzial der Community wird so nicht vollständig ausgeschöpft, was sich negativ auf die Gesamtqualität auswirkt.[46]

Dagegen stellen sich viele Autoren, die grundsätzlich von den Vorteilen offener Quellen überzeugt sind. Witten et. al.[47] stellen in ihrer Untersuchtung über den Einfluss der Quellcodesichtbarkeit auf die Sicherheit zunächst fest, das Open Source keine Garantie für Sicherheit ist, führt aber Argumente auf, die für eine bessere Sicherheit in Open Source Software sprechen. Dabei wird deutlich, dass eine geschlossene Quelle keinen Mehrwert für die Sicherheit bringen kann, da Angreifer in jedem Fall durch *reverse engineering* oder auf illegale Weise, z.B. über einen internen Entwickler, an die relevanten Teile des Quellcodes kommen. Damit unterstützt er die Kritik an dem *Security through Obscurity*-Konzept (siehe Abschnitt 3.2). Geschlossene Quellen sind daher kein Beitrag zur Sicherheit, sondern ein Werkzeug von Rechtszwängen zum Schutz vor Missbrauch des gestiegen Eigentums.[48]

Witten et. al. gehen auch auf das Kompilieren, also das Übersetzen des Quellcode in ein ausführbares Programm, ein. Compiler können durch zusätzliche Sicherheitsprüfungen zur Sicherheit eines Programms beitragen, sie aber auch negativ beeinflussen, wenn bspw. der Complier selbst Fehler aufweist. Nur bei offenen Quellen können Nutzer ein Programm selbst kompilieren, was für Witten et.

[45] Scott A. Hissam, Daniel Plakosh und Charles B. Weinstock (2002). „Trust and vulnerability in open source software.“ In: *IEE Proceedings - Software* 149.1, S. 47–51. URL: http://dblp.uni-trier.de/db/journals/iee/iee-s149.html#HissamPW02.

[46] Nikolai Bezroukov (1999). „A Second Look at the Cathedral and the Bazaar“. In: *First Monday 4(12)*. Zuletzt besucht am 31.03.2015. URL: http://firstmonday.org/article/view/708/618.

[47] Brian Witten, Carl E. Landwehr und Michael A. Caloyannides (2001). „Does Open Source Improve System Security?“ In: *IEEE Software* 18.5, S. 57–61. URL: http://dblp.uni-trier.de/db/journals/software/software18.html#WittenLC01.

[48] Brian Witten, Carl E. Landwehr und Michael A. Caloyannides (2001). „Does Open Source Improve System Security?“ In: *IEEE Software* 18.5, S. 57–61. URL: http://dblp.uni-trier.de/db/journals/software/software18.html#WittenLC01.

al. einen Vorteil darstellt.[49]

Zusätzlich werden quantitative Argumente aufgeführt. Es wird der durchschnittliche Zeitraum zwischen der Aufdeckung und Schließung einer Sicherheitslücke untersucht. Als Beispiel wird Windows als geschlossenes Betriebssystem und Red Hat Linux, als offenes Betriebssystem, gewählt. In Windows beträgt dieser Zeitraum durchschnittlich 16,1 Tage und in Red Hat Linux 11,2 Tage, sodass in diesem Fall von einem positiven Effekt der offene Quelle auf den Fehlerfindungsprozess ausgegangen werden kann.[50]

Als Konsequenz daraus erachten Witten et. al. das Öffnen von Quellen als sinnvoll, da so Nutzer und andere Entwickler zum Erhöhen der Sicherheit beitragen können. Aus der Untersuchung ergibt sich auch, dass Open Source Software weniger anfällig für nicht-bösartige Fehler ist, was für eine bessere Gesamtqualität in Open Source Projekten spricht. Ein Problem sehen Witten et. al. aber in der Motivation der Entwickler. Unabhängig von der Offenheit der Quelle fehlt den Entwicklern der Anreiz sichere Software zu schreiben, so lange weniger sichere Software profitabel ist[51]

Eine ähnliche Untersuchung wurde von Petreley[52] durchgeführt. Hier wurden die letzten 40 Sicherheitslücken in Windows Server 2003 und Red Hat Enterprise Linux AS v.3 verglichen. Auch in dieser Untersuchung wird ein proprietäres Windows Betriebssystem mit einer quelloffenen Linux-Distribution verglichen. Gemessen an Microsofts eigenen Sicherheitsmetriken sind 38% der Lücken in Windows Server 2003 als kritisch einzustufen und im Vergleich dazu 10% in Red Hat Enterprise Linux AS.[53] Die von Microsoft verwendeten Metriken werden als subjektiv und mangelhaft bezeich-

[49] Brian Witten, Carl E. Landwehr und Michael A. Caloyannides (2001). „Does Open Source Improve System Security?“ In: *IEEE Software* 18.5, S. 57–61. URL: http://dblp.uni-trier.de/db/journals/software/software18.html#WittenLC01.

[50] Brian Witten, Carl E. Landwehr und Michael A. Caloyannides (2001). „Does Open Source Improve System Security?“ In: *IEEE Software* 18.5, S. 57–61. URL: http://dblp.uni-trier.de/db/journals/software/software18.html#WittenLC01.

[51] Brian Witten, Carl E. Landwehr und Michael A. Caloyannides (2001). „Does Open Source Improve System Security?“ In: *IEEE Software* 18.5, S. 57–61. URL: http://dblp.uni-trier.de/db/journals/software/software18.html#WittenLC01.

[52] Nicholas Petreley (2004). *Security Report: Windows vs Linux An - independent assessment*. Zuletzt besucht am 30.03.2015. URL: http://www.theregister.co.uk/2004/10/22/security_report_windows_vs_linux/.

[53] Nicholas Petreley (2004). *Security Report: Windows vs Linux An - independent assessment*. Zuletzt besucht am 30.03.2015. URL: http://www.theregister.co.uk/2004/10/22/security_report_windows_

net, sodass nach der Anwendung strikterer Metriken sich der Anteil der kritischen Sicherheitslücken in Windows Server 2003 auf 50% erhöht und aufgrund streitbarer Lücken in Red Had Linux Enterprise AS auf 5% sinkt. Zusätzlich wurde die United States Computer Emergency Readiness Team (CERT) Datenbank angefragt, die das Ergebnis untermauert. Laut der CERT Datenbank werden 39 der 40 (97,5%) aktuellsten Lücken in Windows Server 2003 als kritisch eingestuft, im Vergleich dazu werden 3 von 40 (7,5%) in Red Hat Linux Enterprise AS und 6 von 40 (15%) im Linux Kernel selbst als kritisch eingestuft. Nach Petrelys Untersuchung haben Open Source Betriebssysteme einen deutlichen Sicherheitsvorteil gegenüber Windows Betriebssystemen. Diese Meinung wird von Payne[54] unterstützt.[55]

Payne vergleicht ebenfalls zwei Open Source Betriebssysteme mit einem Windows Betriebssystem und stellt fest, dass diese deutlich weniger Schwachstellen haben. Allerdings relativiert er den Einfluss der Offenheit auf die Sicherheit, da das positive Ergebnis für das quelloffene OpenBSD vor allem durch den hohen Sicherheitsstandard des Projektes zustande kommt und nicht zwangsläufig durch die Offenheit der Quelle.[56]

Weitere Argumente für die bessere Sicherheit in Open Source Software bringt Weehler an.[57] Nach Weehler basiert Open Source Software grundsätzlich auf dem von Saltzer und Schroeder eingeführten Prinzip des *Open Design*, ein Prinzip zur Entwicklung sicherer Systeme.[58] Dabei bringt er ein Beispiel für den Vorteil des Viele-Augen-Prinzips an. Die Softwarefirma Borland baute eine Hintertür in die Datenbanksoftware InterBase ein, welche einen unberechtigten Zugriff auf die Datenbank und

---

vs_linux/.

[54] Christian Payne (2002). „On the security of open source software“. In: *Inf. Syst. J.* 12.1, S. 61. DOI: 10.1046/j.1365-2575.2002.00118.x. URL: http://dx.doi.org/10.1046/j.1365-2575.2002.00118.x.

[55] Nicholas Petreley (2004). *Security Report: Windows vs Linux An - independent assessment*. Zuletzt besucht am 30.03.2015. URL: http://www.theregister.co.uk/2004/10/22/security_report_windows_vs_linux/.

[56] Friedrich-L. Holl Hartmut Heinrich, Jan Tobias Mühlberg Katharina Menzel und Hanno Schüngel Ingo Schäfer (2006). *Open-Source-Software und ihre Bedeutung für Innovatives Handeln*. Zuletzt besucht am 29.03.2015. URL: http://www.bmbf.de/pubRD/oss_studie.pdf.

[57] David A. Wheeler (2014). *Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers!* Zuletzt beischt am 30.03.2015. URL: http://www.dwheeler.com/oss_fs_why.html#security.

[58] Jerome H. Saltzer und Michael D. Schroeder (1975). *The Protection of Information in Computer Systems*. Zuletzt besucht am 30.03.2015. URL: http://www.cs.virginia.edu/~evans/cs551/saltzer/.

das installieren beliebiger Software auf dem System erlaubte. Diese Lücke konnte mindestens 6 Jahre genutzt werden. Erst als der Quellcode für die Weiterentwicklung als Open Source Software freigegeben wurde konnte die Lücke innerhalb von 5 Monaten gefunden und geschlossen werden.[59] Des Weiteren hat Weehler eine Vielzahl quantitativer Studien gesammelt und ausgewertet. Als Gesamtergebnis dieser Studien sieht Weehler die Überlegenheit von Open Source Software, da diese laut der Studien resistenter gegen Angriffe ist. Da viele der Studien von Regierungen durchgeführt wurden, die Open Source Software grundsätzlich positiv gegenüber stehen, sind deren Ergebnisse nicht unbedingt objektiv aussagekräftig.[60]

Einen weiteren Ansatz der Frage nach der Sicherheit in offener und proprietärer Software nachzugehen stammt von Anderson.[61] Er stellte auf Basis eines mathematischen Modells für ausreichend komplexe Systeme fest, dass grundsätzlich kein Unterschied zwischen offenen und geschlossenen Quellen besteht. Eine offene Quelle hilft Angreifern und Entwicklern gleichermaßen, während geschlossene Quellen nicht sicher gegenüber Angreifern sind. Damit unterstützt er die Meinung von Witten et. al. und wird in dieser Ansicht auch von Diffie unterstützt.[62] Es ist zwar möglich die Funktionsweise eines Programms vor der Öffentlichkeit zu verbergen, Angreifern wird dies dennoch immer möglich sein.

Des Weiteren stellt Anderson fest, dass die Auffassung des Sicherheitsbegriffs für Nutzer und Entwickler oft unterschiedlich ist. Entwickler verstehen unter Sicherheit oft den Schutz ihres geistigen Eigentums und nur die Nutzer verstehen Sicherheit wie sie in Abschnitt 2.4 als Informationssicherheit definiert wurde.[63]

[59] David A. Wheeler (2014). *Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers!* Zuletzt beischt am 30.03.2015. URL: http://www.dwheeler.com/oss_fs_why.html#security.

[60] Friedrich-L. Holl Hartmut Heinrich, Jan Tobias Mühlberg Katharina Menzel und Hanno Schüngel Ingo Schäfer (2006). *Open-Source-Software und ihre Bedeutung für Innovatives Handeln*. Zuletzt besucht am 29.03.2015. URL: http://www.bmbf.de/pubRD/oss_studie.pdf.

[61] Ross J. Anderson (2002). *Security in open versus closed systems — the dance of Boltzmann, Coase and Moore*. Technical Report. England: Cambridge University. URL: http://www.cl.cam.ac.uk/~rja14/Papers/toulouse.pdf.

[62] Whitfield Diffie. *Risky business: Keeping security a secret*. Zuletzt besucht am 31.03.2015. URL: http://www.zdnet.com/article/risky-business-keeping-security-a-secret-5000127072/.

[63] Ross J. Anderson (2002). *Security in open versus closed systems — the dance of Boltzmann, Coase and Moore*. Technical Report. England: Cambridge University. URL: http://www.cl.cam.ac.uk/~rja14/Papers/toulouse.

6 Fazit und Ausblick

Die in Abschnitt 1 eingeführte These, dass es grundsätzlich keinen Unterschied zwischen der Sicherheit in Open Source Software und proprietärer Software gibt, lässt sich mit Argumenten aus Abschnitt 5 belegen. Anderson zeigt mittels eines mathematischen Modells, dass es wie in der These beschrieben, keinen Unterschied in der Sicherheit von Open Source Software und proprietärer Software gibt.[64] Unterstützt wird die These zudem von Diffie und Witten et. al..[65][66] Witten et. al. zeigen aber auch ein Beispiel auf, in dem Fehler im Open Source Betriebssystem Red Had Linux AS v.3 schneller behoben werden als in dem proprietären Betriebssystem Windows Server 2003.[67].

Petreley untersucht die selben Betriebssysteme mit Bezugnahme auf die Schwere der entdeckten Sicherheitslücke und kommt zu einem ähnlichen Ergebnis.[68]. Payne untermauert dies mit einer weiteren Untersuchung in der zwei Open Source Betriebssysteme deutlich weniger Sicherheitslücken aufweisen als ein Windows Betriebssystem.[69]

Daraus lässt sich schließen, dass es neben der Offenheit der Quelle weitere Faktoren gibt, die auf die Sicherheit von Software einwirken. Nach Witten et. al. kann einer dieser Faktoren das Kompilieren

---

pdf.

[64] Ross J. Anderson (2002). *Security in open versus closed systems — the dance of Boltzmann, Coase and Moore*. Technical Report. England: Cambridge University. URL: http://www.cl.cam.ac.uk/~rja14/Papers/toulouse.pdf.

[65] Brian Witten, Carl E. Landwehr und Michael A. Caloyannides (2001). „Does Open Source Improve System Security?“ In: *IEEE Software* 18.5, S. 57–61. URL: http://dblp.uni-trier.de/db/journals/software/software18.html#WittenLC01.

[66] Whitfield Diffie. *Risky business: Keeping security a secret*. Zuletzt besucht am 31.03.2015. URL: http://www.zdnet.com/article/risky-business-keeping-security-a-secret-5000127072/.

[67] Brian Witten, Carl E. Landwehr und Michael A. Caloyannides (2001). „Does Open Source Improve System Security?“ In: *IEEE Software* 18.5, S. 57–61. URL: http://dblp.uni-trier.de/db/journals/software/software18.html#WittenLC01.

[68] Nicholas Petreley (2004). *Security Report: Windows vs Linux An - independent assessment*. Zuletzt besucht am 30.03.2015. URL: http://www.theregister.co.uk/2004/10/22/security_report_windows_vs_linux/.

[69] Christian Payne (2002). „On the security of open source software“. In: *Inf. Syst. J.* 12.1, S. 61. DOI: 10.1046/j.1365-2575.2002.00118.x. URL: http://dx.doi.org/10.1046/j.1365-2575.2002.00118.x.

durch die Nutzer sein.[70] Payne sieht zudem den hohen Sicherheitsstandard bei dem offenen Betriebssystem OpenBSD als Grund für die bessere Sicherheit.

Meiner Meinung nach hat auch das Wegfallen von wirtschaftlichen Interessen und die lange Bestandsdauer gut organisierter Open Source Projekte einen entscheiden Faktor auf die Sicherheit. Dies spiegelt unter anderem in Projekten wie OpenBSD wieder.[71] Mit Bezug zur These heißt dies, dass sie nicht auf alle Open Source Software anwendbar ist und es Fälle gibt in denen Open Source Software sicherer als proprietäre Software ist. Dies wird auch durch die Sammlung von Studien durch Wheeler untermauert.[72]

Die Argumente von Raymond und O'Reilly ordne ich der ideologisch geführten Diskussion zu und setze sie daher nicht in Bezug zur These. Das Selbe gilt für die Argumente von Hissam et. al. und Bezroukov.

Der Heartbleed Bug aus Abschnitt 4 zeigt, dass auch in Open Source Projekten schwerwiegenden Sicherheitslücken auftreten können. Warum solche Fehler in Open Source Software auftreten und wie diese verhindert werden können, sind Fragen die zukünftig beantwortet werden müssen. Eine Problemstellung in diesem Zusammenhang ist das Finden von Finanzierungsmodellen für Open Source Projekte die in öffentlichen und sicherheitskritisch Bereichen Anwendung finden.

[70] Brian Witten, Carl E. Landwehr und Michael A. Caloyannides (2001). „Does Open Source Improve System Security?“ In: *IEEE Software* 18.5, S. 57–61. URL: http://dblp.uni-trier.de/db/journals/software/software18.html#WittenLC01.

[71] Christian Payne (2002). „On the security of open source software“. In: *Inf. Syst. J.* 12.1, S. 61. DOI: 10.1046/j.1365-2575.2002.00118.x. URL: http://dx.doi.org/10.1046/j.1365-2575.2002.00118.x.

[72] David A. Wheeler (2014). *Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers!* Zuletzt beischt am 30.03.2015. URL: http://www.dwheeler.com/oss_fs_why.html#security.