Analyse zur Sicherheit von Open-Source Software

Marcus Rogowsky
Betreuer: Dr. Heiko Niedermayer
Seminar Innovative Internettechnologien und Mobilkommunikation, SS 2014
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: m.rogowsky@tum.de

KURZFASSUNG

Open-Source Software ersetzt heute in Firmen, Behörden und privatem Umfeld immer mehr kommerzielle Closed-Source Software und ist durch die weite Verbreitung zu einem attraktiven Angriffsziel für Hacker geworden. Diese Entwicklung hat zu einer Debatte über die Sicherheit von Open-Source Software im Vergleich zu Closed-Source Software geführt. Es ist allerdings schwierig die unterschiedlichen Einflüsse der beiden Strategien auf die Software-Sicherheit zu modellieren. Die veröffentlichten Modelle betrachten entweder nicht den Unterschied zwischen Open-Source Software und Closed-Source Software oder liefern Ergebnisse, die sich nicht mit den Ergebnissen empirischer Studien über die Software-Sicherheit von großen Open-Source Projekten decken. Durch die Berichterstattung über den Heartbleed Bug in der OpenSSL Bibliothek ist diese Problematik in das Blickfeld der öffentliche Wahrnehmung gerückt.

Schlüsselworte

Open-Source Software, Closed-Source Software, Security, Heartbleed

1. EINLEITUNG

Der Einsatz von Open-Source Software hat in den letzten 20 Jahren auf allen Computer Plattformen stark zugenommen. Für Server, Desktop PCs und Smartphones werden Betriebssysteme und Software angeboten, die frei verwendet werden können und deren Code offen für jeden einzusehen ist. Damit unterscheidet sie sich von proprietärer Software, bei der der Code nur dem Entwickler bekannt ist und die vor dem Einsatz meist bezahlt oder lizenziert werden muss.

Durch die weite Verbreitung von Open-Source Software im behördlichen, geschäftlichen und privaten Umfeld wird diese auch ein attraktives Ziel für Angreifer, die Schwachstellen ausnutzen und damit Zugriff auf vertrauliche Daten bekommen oder Schäden verursachen können. Vor allem bei Systemen, die mit offenen Netzwerken wie dem Internet verbunden sind, muss die Sicherheit der verwendeten Software gewährleistet sein.

Da Open-Source Software heute die Basis für viele geschäftskritische Bereiche und kritischer Infrastruktur bildet, müssen hier überprüfbare Standards und Methoden zur Feststellung der Sicherheit von Software entwickelt werden. Ein wichtiger Aspekt ist dabei der Vergleich der Sicherheit von Open-Source Software mit der von Closed-Source Software und die Abwägung der Vor- und Nachteile beider Strategien.

In den beiden nachfolgenden Abschnitten wird eine Einführung in das Thema Open-Source Software und die Debatte über deren Sicherheit gegeben. Im Abschnitt 4 werden allgemeine Modelle zur Software-Sicherheit vorgestellt und ein Modell betrachtet, dass zwischen Open-Source Software und Closed-Source Software unterscheidet. Die Aussagen des Modells werden dann anhand einer empirischen Studie überprüft. Im darauffolgenden Abschnitt 5.2 wird ein Einblick gegeben, wie und in welchem Umfang Open-Source Software in Firmen eingesetzt wird. Da die Berichterstattung über den Heartbleed Bug die Wahrnehmung der Öffentlichkeit zu dem Thema Sicherheit von Open-Source Software grundlegend verändert hat, wird im letzten Abschnitt erklärt wie dieser Bug aufgetreten ist und welche Auswirkungen er gehabt hat.

2. DEFINITION VON OPEN-SOURCE SOFTWARE

Der Begriff Open-Source Software wurde mit der Gründung der Open Source Initiative 1998 eingeführt. Grundsätzlich ist eine Software als Open-Source Software zu verstehen, wenn der Quellcode für jeden zu Verfügung steht und die Software frei genutzt, kopiert und verteilt werden darf. Wie dabei der Entwicklungsprozess der Software genau abläuft und wie Codeänderungen in die originale Version aufgenommen werden, kann sich bei verschiedenen Projekten stark unterscheiden.

Eric S. Raymond, Mitbegründer der Open Source Initiative, beschrieb in seinem Essay „Die Kathedrale und der Basar“ [4] 1999 die zwei weitverbreitetsten Methoden als „Basar“ und „Kathedrale“. Bei der als „Kathedrale“ bezeichneten Methode wird der Entwicklungsprozess von einer kleinen Gruppe von Entwicklern kontrolliert, die die offizielle Version der Software erstellen und den Quellcode für jede veröffentlichte Version zur Verfügung stellen. Zwischen den einzelnen Veröffentlichungen kann die Öffentlichkeit nur indirekten Einfluss auf die Entwicklung nehmen. Dagegen setzt die als „Basar“ bezeichnete Methode auf die Zusammenarbeit mit der Öffentlichkeit über das Internet und erlaubt es jedem den Code in der Versionsverwaltung zu ändern oder neuen Code hinzuzufügen. Der aktuelle Quellcode des Projekts steht somit der Öffentlichkeit zu jedem Zeitpunkt in der Entwicklung zur Verfügung und Änderungen laufen sofort durch einen Peer-Review Prozess.

doi: 10.2313/NET-2014-08-1_20

3. DEBATTE OPEN-SOURCE SOFTWARE VS. CLOSED-SOURCE SOFTWARE

Zwischen den beiden Lagern der Open-Source und Closed-Source Softwareentwickler findet eine Debatte darüber statt, welches Modell der Entwicklung mehr Sicherheit bietet. Einig sind sich beide Seiten darin, dass in offenem Quellcode Bugs schneller gefunden werden und damit mehr Sicherheitslücken aufgedeckt werden. Die Auswirkungen auf die Sicherheit der Software werden jedoch unterschiedlich bewertet.

Befürworter des Open-Source Modells führen an, dass durch den Peer-Review Prozess den die Software durchlaufen muss die Codequalität generell höher ist und weniger Sicherheitslücken in der veröffentlichten Version vorhanden sind. Eric S. Raymond schrieb dazu in seinem Essay [4] „Given enough eyeballs, all bugs are shallow." Allerdings spielt bei der Beseitigung der gefundenen Bugs der verwendetete Entwicklungsprozess eine entscheidende Rolle. Bei einem dem „Basar"-Modell folgendem Entwicklungsprozess kann jeder der Schwachstellen findet diese direkt im originalen Code beseitigen. Wird dagegen das „Kathedralen"-Modell verwendet, kann man als Außenstehender nur die Entwickler des Projekts benachrichtigen, die dann über das weitere Vorgehen entscheiden. Ein entscheidender Vorteil der Open-Source Methode ist, dass man als Anwender die Funktionalität der Software selbst überprüfen kann und nicht darauf vertrauen muss, dass keine versteckten Hintertüren von den Entwicklern eingebaut wurden.

Dagegen argumentieren die Befürworter des Closed-Source Modells, dass das bloße Bereitstellen des Quellcodes keine Garantie für Sicherheit darstellt. Elias Levy merkt in seinem Artikel „Wide Open Source" [5] 2000 an: „Sure, the source code is available. But is anyone reading it?" Seiner Erfahrung nach lesen Endanwender der Software den dazugehörigen Code überhaupt nicht oder sind nicht qualifiziert genug, um Sicherheitslücken zu finden. Viele potenzielle Reviewer gehen zudem davon aus, dass der Quellcode von anderen bereits überprüft wurde. Dabei ist das Projekt oft so komplex, dass es nur im Rahmen professioneller Audits untersucht werden kann, wie es zum Beispiel bei der Verschlüsselungssoftware Truecrypt der Fall ist. In Softwarefirmen gäbe es außerdem feste Vorgaben an die Entwickler und eine klar strukturierte Organisation. Bevor die Software veröffentlicht wird, wird sie von speziell geschultem Personal in der Qualitätssicherungsabteilung überprüft. Und durch die Geheimhaltung des Quellcodes würde ein zusätzlicher Schutz geschaffen, der es für Angreifer schwieriger macht Schwachstellen zu finden und diese würden sich deshalb einfachere Ziele vornehmen.

4. MODELLE ZUR SOFTWARE-SICHERHEIT

Die öffentliche Debatte über die Sicherheit von Open-Source Software ist geprägt von verschiedenen Vorurteilen gegenüber der jeweiligen Entwicklungsmethode. Um allerdings den Einfluss von Open-Source auf die Sicherheit untersuchen zu können, braucht man Modelle, die Vorhersagen darüber machen können, wie verschiedene Faktoren die Sicherheit der veröffentlichten Software bestimmen. Diese Modelle sollten dann Hinweise darauf geben aus welchen Gründen Fehler auftreten, und wie diese in Zukunft vermieden werden können.

4.1 Modelle für Software allgemein

Schryen und Kadura [2] geben einen Überblick über die wichtigsten verwendeten Modelle und untersuchen, worin ihre Schwächen liegen. Die Grundlage dieser Modelle basiert auf der Methode von Littlewood et al. und Kimura, die die Zuverlässigkeitstheorie auf Software-Sicherheit anwenden. Erfolgreiche Angriffe auf eine Software werden dabei als Ereignisse betrachtet, die aufgrund von Sicherheitslücken auftreten. Für die Zeit bis zum nächsten Eindringen in das System wird eine Exponentialverteilung angenommen, wobei die Autoren keine Aussagen über den Faktor λ für die Anzahl der erwarteten Ereignisse pro Zeitintervall machen. Um diesen Faktor genauer zu spezifizieren, beobachteten Jonsson und Olovsson das Verhalten von Angreifern auf einem verteilten UNIX-System und wie schwer es für diese war in das System einzudringen. Mit ihren Ergebnissen stellten sie die Hypothese auf, dass das Auftreten von Sicherheitsverstößen in drei Phasen unterteilt werden kann: Die Lernphase, die Standard-Angriffsphase und die innovative Angriffsphase. Während der Standard-Angriffsphase ist die Wahrscheinlichkeit, dass ein Angriff erfolgreich ist deutlich höher als während den anderen beiden Phasen. Innerhalb der jeweiligen Phasen folgen die Zeiten zwischen erfolgreichen Angriffen der Exponentialverteilung was die Annahme unterstützt, dass das Zuverlässigkeitsmodell auf Software-Sicherheit angewendet werden kann. Summiert man alle bisher gefundenen Sicherheitslücken auf bilden die drei Phasen eine „S" Kurve (Abbildung 1).

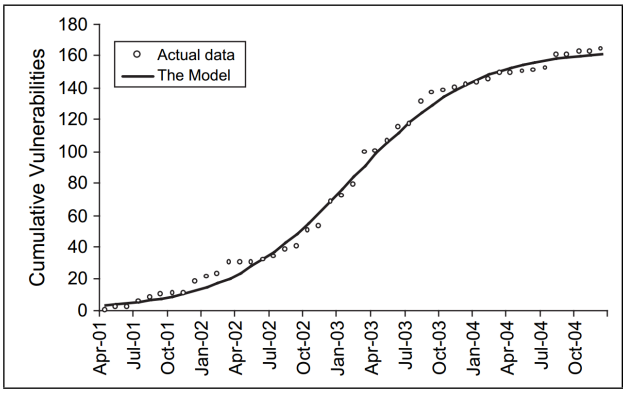

Abbildung 1: [6] S-Kurve für Linux 7.1

Alhazmi et al. [6] erklären diesen Verlauf damit, dass die Rate mit der Schwachstellen gefunden werden linear von zwei Faktoren abhängt: Die Verbreitung der Software und den noch zu finden Schwachstellen in der Software. Basierend auf diesen Annahmen stellten sie eine Gleichung auf, mit der sich anhand des bisherigen Verlaufs der Anzahl an gefunden Schwachstellen die Gesamtanzahl an Schwachstellen in der Software vorhersagen lassen sollte. Beim späteren Vergleich dieser Vorhersagen mit der Anzahl an tatsächlich gefundenen Schwachstellen stellte sich die vorhergesagte Anzahl als deutlich zu niedrig heraus. Mögliche Erklärungen dafür könnten sein, dass durch Patches später zusätzliche Sicherheitslücken in die Software eingefügt wurden, oder dass der Aufwand der betrieben wird um Sicher-

doi: 10.2313/NET-2014-08-1_20

heitslücken zu finden nicht linear mit der Zeit verläuft, zum Beispiel wenn die Anzahl an Reviewern über die Zeit schwankt.

4.2 Modell mit Unterscheidung von Open-Source Software und Closed-Source Software

Das Problem mit den bisher genannten Modellen ist, dass sie den Unterschied zwischen Open-Source und Closed-Source nicht berücksichtigen. Außerdem wird laut Schryen und Kadura [2] die Entwicklung und Überprüfung von Modellen dadurch erschwert, dass zuverlässige Daten zur Software-Sicherheit nur schwer zugänglich oder von zu schlechter Qualität sind.

Um die Unterschiede zwischen Open-Source und Closed-Source berücksichtigen zu können, schlagen Härtig et al. [1] ein Modell vor, dass die Reviewer in zwei Gruppen aufteilt. In der einen Gruppe sind die Angreifer, die Fehler finden wollen, um sie auszunutzen und in der anderen Gruppe die Verteidiger, die Fehler finden wollen, um sie zu beheben und die Qualität der Software zu verbessern. Die Software wird aufgeteilt in einzelne Einheiten, die entweder einen Fehler aufweisen oder fehlerfrei sind und beiden Gruppen jeweils eine Wahrscheinlichkeit zugeordnet Fehler zu finden. Im Fall von Open-Source Software bekommen beide Gruppen die gleiche Wahrscheinlichkeit Fehler zu finden, im Fall von Closed-Source Software wird die Wahrscheinlichkeit für die Angreifer um einen Faktor α verringert.

Die Fehlersuchphase wird als schrittweises Ziehen von Kugeln mit Zurücklegen modelliert, wobei Angreifer und Verteidiger aus unterschiedlichen Urnen ziehen, um die jeweiligen Wahrscheinlichkeiten abzubilden. Für beide Gruppen gelten unterschiedliche Gewinnbedingungen: Die Angreifer können einen beliebigen Fehler finden, der von den Verteidigern noch nicht gefunden wurde, um zu gewinnen. Die Verteidiger müssen jedoch jeden Fehler vor den Angreifern finden, um zu gewinnen. Wenn die Angreifer Fehler A finden, ist es irrelevant ob die Verteidiger Fehler B oder C gefunden haben, nur ob sie Fehler A gefunden haben oder nicht, ist entscheidend. Ziehen ein Angreifer und ein Verteidiger im gleichen Schritt denselben Fehler gewinnt der Angreifer, da er diesen sofort ausnutzen kann, ohne auf die Qualität seiner Schadsoftware zu achten. Der Verteidiger muss dagegen den Fehler erst beheben, ohne unerwünschte Nebeneffekte in der Software zu verursachen und die Endanwender dann den Patch herunterladen und installieren.

Um mit ihrem Modell Aussagen über den Unterschied zwischen Open-Source und Closed-Source Projekten zu machen, wenden sie dieses auf ein fiktives Softwareprojekt an, das typisches Fehlerverhalten aufweist und berechnen damit wie viele Verteidiger benötigt werden, um mit einer vorgegebenen Wahrscheinlichkeit alle Fehler vor den Angreifern zu finden. Mit einer Größe von 1 Millionen Zeilen Code und üblichen Werten von 0,3 Fehlern pro 1.000 Zeilen von denen 5% sicherheitsrelevant sind ergibt das 15 relevante Fehler im gesamten Projekt. Die Anzahl an Angreifern wird auf 500 geschätzt, da keine aussagekräftigen Studien dazu gefunden werden konnten. Für die Wahrscheinlichkeit, dass ein Verteidiger in einem Schritt einen Fehler findet, werden Werte von $q = 0,001\%$ bis $q = 0,005\%$ angenommen. Für die Angreifer werden bei Open-Source Software die gleichen Werte und bei Closed-Source Software um den Faktor $\alpha = 10$ kleinere Werte verwendet.

Tabelle 1: [1]
Anzahl benötigter Verteidiger für $p_w = 0.6$

$p_w = 0.6$	$q = 0.001\%$	$q = 0.002\%$	$q = 0.005\%$
$\alpha = 10$	1455	1466	1500
$\alpha = 1$	15630	17133	26245

Tabelle 2: [1]
Anzahl benötigter Verteidiger für $p_w = 0.9$

$p_w = 0.9$	$q = 0.001\%$	$q = 0.002\%$	$q = 0.005\%$
$\alpha = 10$	7360	7654	8774
$\alpha = 1$	124176	unmöglich	unmöglich

Zunächst wird eine sehr niedrige Gewinnwahrscheinlichkeit von $p_w = 0.6$ für die Verteidiger gewählt, wozu man laut Modell bereits etwa 15.000 bis 26.000 Verteidiger im Fall von Open-Source Software braucht, während man bei Closed-Source Software mit nur etwa 1.500 Verteidigern auskommt (Tabelle 1).

Für eine realistischere Betrachtung des Modells wird ein Wert von $p_w = 0.9$ für die Gewinnwahrscheinlichkeit der Verteidiger gewählt. Mit diesem Wert benötigt man mit Open-Source Software allerdings eine unrealistische hohe Anzahl an Verteidigern von über 100.000 oder es ist unmöglich diesen Wert zu erreichen, während bei Closed-Source Software etwa 8.000 Verteidiger ausreichen (Tabelle 2).

Die Aussage des Modells ist damit, dass egal wie viele Verteidiger es gibt, die Angreifer in einem von drei Fällen gewinnen werden. Die Autoren schließen daraus, dass man Angreifer nur mit Geheimhaltung des Quellcodes effektiv abwehren kann.

5. STUDIEN

Die Aussage des Modells aus Abschnitt 4.2 scheint nicht mit der objektiven Wahrnehmung zur Sicherheit von Open-Source Software übereinzustimmen. Sollten sich die Aussagen des Modells bestätigen, wäre der Einsatz von Open-Source Software in kritischer Infrastruktur grob fahrlässig, da diese praktisch nicht gegen Angreifer zu verteidigen wäre. Außerdem findet in den Modellen keine Unterscheidung von schweren Fehlern zu weniger schweren Fehlern statt, obwohl eine schwere Sicherheitslücke durch die Angreifer Administratorrechte auf dem Zielcomputer bekommen können schlimmer ist als 10 weniger schwere Sicherheitslücken, die nur Lesezugriff auf bestimmte Daten ermöglicht.

5.1 Empirische Studie

Um diese Fragen weiter zu untersuchen erstellte Schryen [3] eine empirische Studie, die die Sicherheit von 17 Softwareprojekten vergleicht. Für jedes Anwendungsgebiet wurden mindestens eine Open-Source Software und eine Closed-Source Software ausgewählt, die weitverbreitet sind und den gleichen Zweck erfüllen. Die Daten für die Analyse

stammen dafür aus der National Vulnerability Database (NVD) des NIST, der größten Datenbank für Softwarebugs und sind nach dem Common Vulnerability and Exposures (CVE) Standard abrufbar. Die CVE Liste enthält eindeutige Bezeichner für jede Sicherheitslücke und wird von der MITRE Corporation in Zusammenarbeit mit Sicherheitsexperten und Softwareherstellern verwaltet.

Tabelle 3: [3] Daten über Schwachstellen (2011)

Software	Entwicklung	#Vuln	MTBVD	EASZ
IE 7	closed	74	13.29	linear
Firefox	open	167	5.16	linear
MS Outlook Express 6	closed	23	120.73	linear
Thunderbird 1	open	110	13.79	nicht linear
IIS 5	closed	83	40.90	nicht linear
Apache 2	open	80	40.63	linear
MS Office 2003	closed	99	19.22	nicht linear
OpenOffice 2	open	19	63.16	linear
Windows 2000	closed	358	9.35	linear
Windows XP	closed	297	8.97	linear
Mac OS X	closed	300	4.64	linear
Red Hat Enterprise 4	open	264	5.48	nicht linear
Debian 3.1	open	207	6.45	nicht linear
mySQL 5	open	33	46.00	linear
PostgreSQL 8	open	25	58.96	linear
Oracle 10g v8	closed	63	29.75	nicht linear
DB2 v8	closed	13	136.38	linear

In der ersten Untersuchung wird die Anzahl an Bugs (#Vuln), die mittlere Zeit in Tagen zwischen veröffentlichten Sicherheitslücken (MTBVD) und die Entwicklung der Anzahl an Schwachstellen über die Zeit (EASZ) für die verschiedenen Softwareprojekte verglichen (Tabelle 3). 12 von 17 Projekten folgen dabei einer linearen Entwicklung für die Anzahl der Fehler über die Zeit, was einer konstanten Rate mit der Fehler gefunden werden entspricht. Die Werte für die verschiedenen Projekt gehen zwar weit auseinander, beim Vergleich von Open-Source mit Closed-Source Software kann allerdings kein signifikanter Unterschied festgestellt werden. Auch zwischen Open-Source Projekten die nach dem „Kathedrale"-Modell und Projekten die nach dem „Basar"-Modell entwickelt wurden lassen sich keine Unterschiede feststellen.

Tabelle 4: [3] Schwere der Schwachstellen nach CVSS (2011)

Software	Mittelwert	Median	CVSS > 7
IE 7	6.65	6.80	45.95%
Firefox	6.38	6.40	36.53%
MS Outlook Express 6	6.18	5.10	39.13%
Thunderbird 1	6.53	6.80	47.27%
IIS 5	6.00	5.00	36.14%
Apache 2	5.36	5.00	18.75%
MS Office 2003	8.11	9.30	67.72%
OpenOffice 2	7.61	7.60	63.16%
Windows 2000	6.58	7.20	57.92%
Windows XP	6.67	7.20	57.92%
Mac OS X	6.18	6.80	41.33%
Red Hat Enterprise 4	4.72	4.90	23.11%
Debian 3.1	4.75	4.90	23.19%
mySQL 5	5.05	4.90	12.12%
PostgreSQL 8	6.17	6.80	36.00%
Oracle 10g v8	5.96	5.50	33.33%
DB2 v8	6.22	7.20	53.85%

In der zweiten Untersuchung werden die Softwareprojekte darauf verglichen, wie schwerwiegend die aufgetreten Sicherheitslücken sind. Dazu werden die Werte aus dem Common Vulnerability Scoring System (CVSS) verwendet, bei dem Sicherheitslücken von Sicherheitsexperten und Softwareanbietern im Konsens auf einer Skala von 0 bis 10 (für schwerste Fehler) eingestuft werden (Tabelle 4). Der Anteil der schweren Sicherheitslücken (CVSS Wert höher als 7) an der Gesamtanzahl, der durchschnittliche CVSS Wert und der Median der Einstufungen werden dazu bestimmt. Auch hier ergibt die statistische Analyse jedoch keine signifikanten Unterschiede zwischen Open-Source und Closed-Source Projekten.

Tabelle 5: [3] Gepatchte und ungepatchte Schwachstellen (2011)

Software	ungepatcht	Median CVSS ungepatcht	Median CVSS gepatcht
IE 7	66.22%	5.0	9.3
Firefox	20.36%	5.0	6.8
MS Outlook Express 6	65.22%	5.0	7.3
Thunderbird 1	5.45%	3.45	6.95
IIS 5	48.19%	5.0	7.2
Apache 2	26.25%	4.7	5.0
MS Office 2003	4.04%	5.05	9.3
OpenOffice 2	21.05%	5.25	9.3
Windows 2000	30.39%	5.1	9.3
Windows XP	30.64%	5.0	7.2
Mac OS X	6.67%	5.0	6.8
Red Hat Enterprise 4	14.77%	4.9	4.9
Debian 3.1	14.48%	4.9	4.9
mySQL 5	24.24%	4.6	4.9
PostgreSQL 8	12.00%	9.0	6.3
Oracle 10g v8	12.70%	7.35	5.5
DB2 v8	-	-	-

Mit den Daten aus der NVD der NIST und den Angaben auf den Entwicklerseiten zu den Veröffentlichungsdaten von Patches ist auch eine Untersuchung des Patchverhaltens der Entwickler möglich (Tabelle 5). Es ist zu beobachten, dass sich Microsoft auf das patchen von schweren Fehlern konzentriert und weniger schwere Fehler nicht patched, während die meisten anderen Hersteller wie zum Beispiel Apple und Open-Source Entwickler versuchen alle Fehler zu patchen. Man kann daraus schließen, dass das Patchverhalten stark von den Entwicklern und deren Prioritäten abhängt und nicht von der angewandten Entwicklungsmethode. Ein Unterschied der auf die verwendete Entwicklungsmethode zurückzuführen wäre ist auch hier nicht erkennbar.

Das Gesamtergebnis der Studie ist damit, dass nicht die Entwicklungsmethode für die Sicherheit entscheidend ist, sondern die jeweilige Strategie, wie mit den Softwarefehlern umzugehen ist. Um die Software-Sicherheit zu verbessern wäre deshalb ein möglicher Ansatz, stärkere ökonomische Anreize zu geben, Sicherheitslücken zu patchen. Das Ergebnis dieser Studie, dass Open-Source und Closed-Source Entwicklungsmethoden zur gleichen Software-Sicherheit führen ist allerdings nur auf große und weitverbreitete Softwareprojekte anzuwenden. Für einen Vergleich von kleinen Softwareprojekten ohne große öffentliche Aufmerksamkeit ist eine empirische Analyse dagegen nicht möglich, da dafür zu den jeweiligen Projekten zu wenig Daten vorliegen.

5.2 Einsatz von Open-Source Software in Firmen und Behörden

Open-Source Software hat sich zu einem Milliardenmarkt entwickelt. Die Softwarefirma Red Hat hatte im letzten Jahr einen Umsatz von 1,5 Milliarden US-Dollar [20], die Firma Mozilla etwa 300 Millionen US-Dollar [21]. Das Geschäftsmodell der Firmen die Open-Source Software entwickeln unterscheidet sich dabei von dem herkömmlicher Softwareentwickler. Umsatz wird hauptsächlich durch das Anbieten von Support und zusätzlichen Services zu der Software generiert. Die Firma Red Hat verkauft zwar ihre Linux-Versionen mit integriertem Support, stellt aber gleichzeitig den Source-Code, mit dem das Betriebssystem nachgebaut werden kann, kostenlos zur Verfügung. Mozilla macht den Hauptteil ihrer Einnahmen durch eine Partnerschaft

mit Google, indem Google als Standard Suchmaschine im Browser voreingestellt ist.

Laut einer Umfrage des Marktforschungsinstitut Gartner [7] setzten im Jahr 2008 im Raum Europa, Nordamerika und Asien 85% der Unternehmen Open-Source Software ein. 69% der Firmen haben allerdings keine spezifischen Vorgaben für den Einsatz von Open-Source Software, die die rechtlichen Besonderheiten regeln, sondern setzen diese wie kommerzielle Software ein.

In Deutschland wurde eine Umfrage zu dem Thema Open-Source Einsatz in Unternehmen im Jahr 2009 von heise open und der Wilken GmbH durchgeführt [8]. Dabei gaben 40% der Unternehmen an, dass Open-Source von unternehmenskritischer Bedeutung ist, wobei Open-Source Software umso eher in unternehmenskritischen Bereichen eingesetzt wird, je länger ein Unternehmen Erfahrungen mit Open-Source Software gesammelt hat. Das bedeutet, dass der Einsatz in unternehmenskritschen Bereichen in Zukunft weiter ansteigen wird.

Weiterhin untersuchte die Studie die Gründe für den Einsatz von Open-Source Software, wobei Mehrfachnennungen möglich waren. Dabei war der am häufigsten angegebene Grund mit 90%, dass Open-Source Software vor allem eingesetzt wird, weil das Management Lizenzkosten sparen will und Open-Source als kostenlose Alternative sieht. Bei kleinen und neu gegründeten Unternehmen ist dieses Motiv besonders wichtig. An zweiter stelle steht mit 70% die Herstellerunabhängigkeit und etwa 60% geben an, dass sie Open-Source Software wegen der Software-Sicherheit verwenden. Nur etwa 40% gaben an, dass sie Open-Source Software wegen der Verfügbarkeit des Quellcodes einsetzen, und noch weniger mit 30% weil sie Fehler in der Software selber finden und beheben können. Open-Source kommt in den Firmen besonders häufig in Infrastrukturbereichen zum Einsatz, die Funktionalität über das Internet bereitstellen: Als Server-Betriebssystem (84%), Webserver (81%), Datenbank (79%) und in der Netzwerkinfrastruktur (73%). 57% der Firmen geben an, dass in mindestens einem Bereich Linux als Betriebssystem zum Einsatz kommt. Weltweit wird auf aktiven Webseiten zu 52% der Apache Webserver verwendet [9] (Stand März 2014).

Behörden müssen vor dem Einsatz von Software in sicherheitsrelevanten Bereichen den Quellcode der Software von Drittfirmen überprüfen lassen um die Funktionalität der Software zu verifizieren. Closed-Source Software für die der Code nicht von unabhängigen Dritten überprüft werden kann sind damit keine Alternative. Große Softwarehersteller haben darauf reagiert und Schnittstellen bereitgestellt über die gegen Entgeld der Quellcode zur verfügung gestellt wird. Microsoft hat dazu zum Beispiel die Shared-Source Initiative gegründet. Nach einer Studie der MITRE Corp. [10] sind bereits über 230 Open-Source Softwareprojekte in US Bundesbehörden im Einsatz. Deshalb hat des Department of Homeland Security das „Vulnerabilty Discovery and Redemption“ Programm gestartet, das täglich die Sicherheit von etwa 40 Open-Source Anwendungen, darunter Linux, Apache und MySQL, überprüft.

Open-Source Software kommt in allen Branchen hauptsächlich im Bereich der Netzwerkinfrastruktur zum Einsatz, was bedeutet, dass die Sicherheit der Firma bei Angriffen über das Internet direkt von der Sicherheit der verwendeten Open-Source Software abhängt. Da die Sicherheit von Software aber nicht im voraus bestimmt werden kann, werden bei der Entscheidung welche Software eingesetzt werden soll hauptsächlich die entstehenden Kosten betrachtet. Dieses Kriterium schließt vor allem auch den personellen Aufwand ein, der bei der Installation und Wartung entstehen.

6. HEARTBLEED

Als Heartbleed wird eine Schwachstelle in der Heartbeatfunktion des Transport Layer Security (TLS) Protokolls der OpenSSL Bibliothek bezeichnet, die in den Versionen 1.0.1 bis 1.0.1f aufgetreten ist. Da TLS keine Verbindung aufrecht halten kann ohne ständigen Datenaustausch, wird wenn gerade keine Nachrichten zwischen den beiden kommunizierenden PCs übertragen werden eine Heartbeatnachricht ausgetauscht. Dazu schickt ein PC einen Heartbeatrequest mit einem Payload, der aus einer zufällig generierten Nachricht mit einer zufälligen Länge besteht, an den anderen PC. Dieser schickt dann eine Kopie des erhalten Payloads in einer Heartbeatresponse zurück um die bestehende Verbindung zu bestätigen. Die Nachricht ist dabei folgendermaßen aufgebaut: Aus dem ersten Byte der Nachricht kann der Empfänger erkennen, dass es sich um den Heartbeat-Nachrichtentyp handelt. Die nächsten 2 Bytes enthalten die Länge des Heartbeat-Payloads und die restlichen Bytes sind der eigentliche Payload.

Beim erstellen des Buffers für die Heartbeatresponse und dem Kopieren der Payloadnachricht wird allerdings nicht überprüft, ob der Payload überhaupt so lang war wie die angegebene Länge in dem Request. Das heißt, dass wenn die Payloadnachricht kürzer war als die angegebene Länge, der Rest des Buffers für die Heartbeatresponse-Nachricht mit dem Inhalt im Speicher nach dem eigentlichen Payload gefüllt und als Antwort zurückgesendet wird. Die maximale Länge der Nachricht ist 64 Kilobyte groß. Da der Speicher der für den Payload der Heartbeatrequest angelegt wird in der OpenSSL Bibliothek liegt, können hier die privaten Schlüssel, die der PC für die asymmetrisch verschlüsselte Kommunikation benötigt liegen. Weitere Daten die ausgelesen werden könnten sind zum Beispiel Usernamen, Passwörter, private Nachrichten oder Emails.

Genau so einfach, wie der Fehler auszunutzen ist, konnte er auch in der OpenSSL Version 1.0.1g behoben werden. Dazu wird, wenn die angegebene Länge des Payloads größer ist als die tatsächliche Länge, der Heartbeatrequest verworfen und keine Heartbeatresponse zurückgeschickt.

6.1 Zeitlicher Ablauf

Am 15. Dezember 2011 schlägt Robin Seggelmann einen Patch vor, der das Heartbeatprotokoll an dessen Entwicklung er beteiligt war, zu OpenSSL hinzufügt [11]. Nach einigen Anpassungen wird der Patch am 31. Dezember 2011 von Stephen Henson der dem Entwicklerteam von OpenSSL angehört in den Hauptcodezweig aufgenommen. Dabei übersieht dieser die fehlende Längenabfrage für den Payload und fügt so den Heartbleed Bug in die offizielle OpenSSL Version ein.

Der Fehler bleibt danach über zwei Jahre lang unentdeckt im Code bis ihn Neel Mehta, der für Google Security arbeitet, am 21. März 2014 findet. Den weiteren Verlauf im Umgang mit der Schwachstelle untersucht Ben Grubb in seinem Artikel „Heartbleed disclosure timeline: who knew what and when“ [12]. Noch am selben Tag wird der Fehler auf allen Servern von Google behoben. Am 1. April 2014 informiert Google das OpenSSL Entwicklerteam über die gefundene Schwachstelle, gibt aber keine weiteren Informationen darüber weiter, welche Partner von Google bereits über den Fehler informiert wurden.

Am 2. April findet die finnische IT-Sicherheitsfirma Codenomicon unabhängig von Google den selben Fehler, informiert daraufhin das „National Cyber Security Centre Finland“ und erfindet den später weitverbreiteten Namen Heartbleed. Einige weitere Firmen, inklusive Red Hat und Facebook, wurden von Google, Codenomicon oder der OpenSSL Entwicklergemeinde vor der Veröffentlichung des Fehlers informiert. Wie der Informationsaustausch genau ablief ist aber unbekannt, da jeder der informiert wurde ein Non-Disclosure-Agreement unterzeichnen musste. Am 7. April informiert Codenomicon ebenfalls das OpenSSL Entwicklerteam, das daraufhin entscheidet noch am selben Tag eine neue Version zu veröffentlichen, da sie das Risiko jetzt höher einschätzen wenn zwei Teams den Fehler unabhängig voneinander fast gleichzeitig gefunden haben. Codenomicon veröffentlicht daraufhin einen Tweet in dem sie erklären, dass sie den Bug ebenfalls gefunden haben und verlinken auf ihre Seite Heartbleed.com. Auf dieser Seite ist das von Codenomicon entworfene Logo für den Bug zu sehen und Erklärungen wie der Bug funktioniert. Innerhalb von 24 Stunden nach der Veröffentlichung wurden die ersten erfoglreichen Angriffe mit Heartbleed durchgeführt [13].

Das Content Delivery Network Cloudflare, das bereits vorab informiert wurde, veröffentlicht ebenfalls am 7. April einen Blogeintrag über die Hintergründe des Fehlers und wie man die alte OpenSSL Versionen so erstellen kann, dass der Fehler nicht auftritt. Noch am selben Tag erstellt Ubuntu einen Patch für das Betriebssystem. Am 9. April beschweren sich die Debian und Red Hat Entwickler darüber, dass der Bug unkoordiniert und trotz des verhängten Embargos veröffentlicht wurde und sie so mit einer Vorwarnzeit von nur 2 Tagen keine ausreichenden Vorbereitungen treffen konnten. Außerdem erhält Neel Mehta von Facebook und Microsoft über das Internet Bug Bounty Programm 15,000 US-Dollar, die er an die Freedom of the Press Foundation spendet.

Da CloudFlare nach eigenen Untersuchungen [14] zu dem Schluss gekommen ist, dass es so gut wie ausgeschlossen ist, dass mittels Heartbleed private Schlüssel des Servers ausgelesen werden können, starten sie am 11. April auf dem Firmenblog die Heartbleed Challange, die dazu aufforderte die privaten Schlüssel eines Servers durch Ausnutzung des Heartbleed Bugs herauszufinden und einzusenden. Dafür setzten sie einen nginx Server auf, auf dem eine OpenSSL Version lief, die mit dem Heartbleed Bug angreifbar ist. Bereits 9 Stunden später wurden die korrekten Schlüssel zwei mal eingesendet. Die beiden Gewinner, Softwareentwickler Fedor Indutny und Ilkka Mattila vom National Cyber Security Centre Finland, hatten dazu jeweils 2.5 Millionen bzw. 100,000 Heartbeatrequests an den Server gesendet. In dem Posting, das die beiden Gewinner bekannt gibt, vermutet der Autor Nick Sullivan [15], dass die Schlüssel ausgelesen werden konnten, weil der Server während der laufenden Challange neugestartet wurde. Aufgrund des Ergebnisses der Heartbleed Challange werden bei CloudFlare bis zum 17. April alle bestehenden SSL-Zertifikate durch neu ersetzt.

Am 14. April wird über den ersten Angriff durch Heartbleed berichtet. Die kanadische Steuerbehörde meldet, dass die Sozialversicherungsnummern von etwa 900 Steuerzahlern durch Ausnutzen der Heartbleed Schwachstelle entwendet wurden.

6.2 Grund für die große Aufmerksamkeit

Laut einer Umfrage des Pew Research Center [16] haben bis Ende April 60% der US-Bürger von Heartbleed gehört und 39% der Internetnutzer geben an aufgrund von Heartbleed ihre Passwörter geändert zu haben. Für eine Sicherheitslücke in einer Open-Source Bibliothek sind diese Bekanntheitswerte sehr hoch und eine Folge der breiten Berichterstattung darüber in den öffentlichen Medien. Durch die weitverbreitete Nutzung der OpenSSL Bibliothek auf Servern von IT-Firmen wie Amazon, Facebook, Google, Netflix und Yahoo ist praktisch jeder direkt durch die Sicherheitslücke betroffen gewesen. Insgesamt waren laut Dan Goodin (Ars Technica) [17] zwei drittel aller Webseiten im Internet betroffen. Laut Nicole Perlroth [18] (The New York Times) haben aber auch Wi-Fi Geräte von Cisco und etwa 50 Millionen Android Smartphones mit der Version 4.1.1, sowie Regirungsbehörden wie das F.B.I. und das Pentagon eine Fehlerhafte OpenSSL Version verwendet.

6.3 Ursachen für Heartbleed und Reaktionen

Der Fehler selbst ist leicht zu sehen, blieb aber über 2 Jahre lang unentdeckt Bestandteil der OpenSSL-Bibliothek. Das zeigt, dass obwohl die Bibliothek von vielen Firmen eingesetzt wurde niemand die betroffene Stelle im Quellcode überprüft hat, was der viele Augen Theorie des Open-Source Modells widerspricht. Für diesen Widerspruch gibt es verschieden Gründe die in diesem Fall zusammenfielen:

- Obwohl OpenSSL von vielen großen IT-Firmen eingesetzt wird blieb das Entwicklerteam im Kern das selbe und ist nicht mit gewachsen. Auf der Webseite des Projekts wird angegeben, dass nur ein Entwickler, Dr. Stephen N. Henson, Vollzeitprogrammierer ist und drei weitere freiwillige Entwickler zum Kernteam gehören. Insgesamt erhielt das Entwicklerteam etwa 2000 US-Doller an Spenden pro Jahr [18], was zeigt, dass bei den Firmen, die die OpenSSL Bibliothek verwendeten keine Zahlungen an die Entwickler der eingesetzten Open-Source Software eingeplant wurden.

- Die OpenSSL Bibliothek gilt als sehr sicher und durch die weite Verbreitung wurde angenommen, dass der Quellcode gut überprüft ist. Im Endeffekt haben aber durch diesen Effekt weniger Reviewer den Quellcode überprüft.

- Es gibt für Firmen keine Anreize die Sicherheit der verwendeten Open-Source Projekte selbst zu überprüfen

und da sich Softwarefirmen heute sehr schnell anpassen und neue Dienste entwickeln müssen, wird in der Projektplanung der Aspekt der Software-Sicherheit vernachlässigt. Das hat dazu geführt, dass bis heute keine industrieweiten einheitlichen Sicherheitsstandards für die Entwicklung von Software durchgesetzt werden konnten.

- Der Code der OpenSSL Bibliothek wird zwar laufend von Codeanalyseprogrammen überprüft, laut David A. Wheeler [19] ist aber keines bekannt, das den Heartbleed Bug gefunden hätte.

Das OpenSSL Projekt besteht bereits seit 1998. Dadurch wurden immer mehr Erweiterungen der Bibliothek hinzugefügt, die von den meisten Anwendern nie benötigt werden. Jede dieser zusätzlichen Funktionalitäten erhöhen das Risiko, dass ein sicherheitsrelevanter Fehler in der Bibliothek auftritt. Aus diesem Grund entschieden sich die OpenBSD Entwickler das LibreSSL-Projekt zu starten, in dem als Grundlage der OpenSSL Quellcode der Version 1.0.1g verwendet wird, aber überflüssige Erweiterungen entfernt werden.

Als Reaktion auf das Bekanntwerden der Diskrepanz zwischen der finanziellen Unterstützung des Projekts und dem weitverbreiteten Einsatz der Bibliothek in kritischer Infrastruktur wurde von der Linux Foundation die Core Infrastructure Initiative gegründet. Mit ihr sollen Stellen in Open-Source Projekten finanziert werden, die mit Programmieren besetzt werden die sich ausschließlich mit Software-Sicherheit beschäftigen. Die Initiative wird finanziell unterstützt von 12 großen IT-Firmen, die jeweils 100.000 US-Dollar pro Jahr beisteuern, und hat damit in den nächsten 3 Jahren insgesamt 3,6 Millionen US-Dollar zur Verfügung. Die Initiative wird dem OpenSSL Projekt 2 Stellen für Vollzeitprogrammierer finanzieren.

7. ZUSAMMENFASSUNG

Die Debatte über die Sicherheit von Open-Source Software ist in der Öffentlichkeit angekommen. Durch ihren weitverbreiteten Einsatz in kritischer Infrastruktur, sind heute viele Bereiche des Alltags direkt davon betroffen. Während die Softwarelösungen immer komplexer werden, reichen die entwickelten Modelle zur Analyse der Einflüsse im Entwicklungsprozess auf die Sicherheit dieser Anwendungen noch nicht aus. Anhand empirischer Studien konnte aber gezeigt werden, dass die Wahl der Entwicklungsmethode, Open-Source oder Closed-Source, alleine keinen Einfluss auf die Software-Sicherheit hat. Vielmehr sind die Vorgaben für Sicherheitsstandards im Entwicklerteam entscheidend. Hier kann man ansetzen und einheitliche Standards in der Softwareentwicklung einführen, die von jeder Software, die in kritischen Bereichen eingesetzt werden soll befolgt werden müssen.

Bei der Suche nach den Ursachen des Heartbleed Bugs ist klar geworden, dass Firmen die Open-Source Software einsetzen dabei hauptsächlich auf die Kosten achten und von sich aus keinen Beitrag zur Verbesserung der Softwarequalität leisten. Sollen in Zukunft Fehler in diesem Ausmaß vermieden werden, müssen ökonomische Anreize für Firmen geschaffen werden sich gemeinsam mit den Entwicklern an der Umsetzung der ursprünglichen Idee hinter Open-Source zu beteiligen.

8. LITERATUR

[1] H. Härtig, C.-J. Hamann, M. Roitzsch: *The Mathematics of Obscurity: On the Trustworthiness of Open Source*, In Proceedings of the Ninth Workshop on the Economics of Information Security (WEIS), 2010

[2] G. Schryen, R. Kadura: *Open Source vs. Closed Source Software Towards Measuring Security*, In Proceedings of the 2009 ACM symposium on Applied Computing, pages 2016-2023, ACM New York, NY, USA, 2009

[3] G. Schryen: *Is Open Source Security a Myth?*, In Communications of the ACM, Volume 54 Issue 5, pages 130-140, ACM New York, NY, USA, 2011

[4] E. S. Raymond: *The Cathedral and the Bazaar*, O'Reilly & Associates, Inc. Sebastopol, CA, USA, 1999

[5] E. Levy *Wide Open Source*, `www.securityfocus.com/news/19`

[6] O. Alhazmi, Y. Malaiya, I. Ray: *Measuring, analyzing and predicting security vulnerabilities in software systems*, Computers & Security, Vol. 26, No. 3., pages 219-228, 2007

[7] *Gartner Says as Number of Business Processes Using Open-Source Software Increases, Companies Must Adopt and Enforce an OSS Policy*, `http://www.gartner.com/newsroom/id/801412`

[8] O. Diedrich *Trendstudie Open Source*, `http://www.heise.de/open/artikel/Trendstudie-Open-Source-221696.html`

[9] Netcraft *April 2014 Web Server Survey*, `http://news.netcraft.com/archives/2014/04/02/april-2014-web-server-survey.html`

[10] R. Narine *DHS backs open-source security*, eWeek, 2006

[11] *#2658: [PATCH] Add TLS/DTLS Heartbeats*, `http://rt.openssl.org/Ticket/Display.html?id=2658\&user=guest\&pass=guest`

[12] B. Grubb *Heartbleed disclosure timeline: who knew what and when*, `http://www.smh.com.au/it-pro/security-it/Heartbleed-disclosure-timeline-who-knew-what-and-when-20140415-zqurk.html`

[13] N. Perlroth *Heartbleed Internet Security Flaw Used in Attack*, `http://bits.blogs.nytimes.com/2014/04/18/Heartbleed-internet-security-flaw-used-in-attack/`

[14] N. Sullivan *Answering the Critical Question: Can You Get Private SSL Keys Using Heartbleed?*, `https://blog.cloudflare.com/answering-the-critical-question-can-you-get-private-ssl-keys-using-heartbleed`

[15] N. Sullivan *The Results of the CloudFlare Challenge*, `http://blog.cloudflare.com/the-results-of-the-cloudflare-challenge`

[16] PewResearch *Heartbleed's Impact*, `http://www.pewinternet.org/2014/04/30/heartbleeds-impact/2/`

[17] D. Goodin *Critical crypto bug in*

OpenSSL opens two-thirds of the Web to eavesdropping, http://arstechnica.com/security/2014/04/critical-crypto-bug-in-OpenSSL-opens-two-thirds-of-the-web-to-eavesdropping/

[18] N. Perlroth *Heartbleed Highlights a Contradiction in the Wen*, http://www.nytimes.com/2014/04/19/technology/heartbleed-highlights-a-contradiction-in-the-web.html

[19] D. A. Wheeler *How to Prevent the next Heartbleed*, http://www.dwheeler.com/essays/heartbleed.html

[20] Red Hat *Red Hat Reports Fourth Quarter and Fiscal Year 2014 Results*, http://www.redhat.com/about/news/press-archive/2014/3/red-hat-reports-fourth-quarter-and-fiscal-year-2014-results

[21] David Murphy *Google Paying Mozilla Almost $1B for Firefox Search: Why?*, http://www.pcmag.com/article2/0,2817,2398046,00.asp

doi: 10.2313/NET-2014-08-1_20