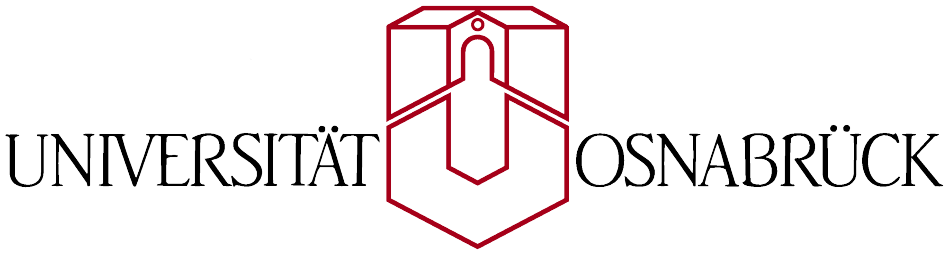
Till Zimmermann
Universität Osnabrück
Sommersemester 2018

Matrikelnummer: 956242
07.09.2018



Ausarbeitung zum Thema

Fallstricke in der FLOSS-Nutzung- & Entwicklung

Im Rahmen des Seminars "Open-Source Software-Entwicklung"

Betreuer:
Prof.Dr.rer.nat Michael Brinkmeier
Lars Kiesow, M.Sc.

Inhaltsverzeichnis

Diese Seminararbeit soll einen kurzen Überblick über einige bekannte Fehler und Fallstricke in der Entwicklung und Nutzung von Freier & Open-Source-Software (FOSS) geben, um Einblicke in Entwicklungsprozesse zu erhalten und damit aufzuzeigen, wie Fehler dieser Art vermieden werden können.

Abkürzungsverzeichnis

FOSS free and open source software

FLOSS free/libre and open source software

GPL General Public License

FSFE Free Software Foundation Europe

SOHO Small Office, Home Office

RFC Request for Comments

IETF Internet Engineering Task Force

1 Motivation & Einführung

In den vergangenen Jahren ist die Bedeutung von Freier & Open-Source-Software (im folgenden »FOSS«) stetig gewachsen. Wurde diese vor einigen Jahrzehnten noch als rein ideologische Idee gesehen, wenden sich heute selbst große Softwarefirmen mehr und mehr der Veröffentlichung ihrer Software oder Teile davon zu, wie etwa Google[1], Microsoft[2] und Apple[3]. Während mehrere Firmen in den Anfangsjahren von FOSS-Alternativen zu ihrer proprietären Software diese mittels diverser - teils eher unlauteren - Praktiken versuchten als unrealistisch zu brandmarken, stellt Freie & Open-Source-Software mittlerweile einen durchaus relevanten Markt in der Computerindustrie- und wissenschaft dar. Somit hat sich die Verbreitung und auch die Entwicklungsarbeit von FOSS-Software stark weiterentwickelt, und ähnelt in vielen Teilen heute auch der Entwicklung im proprietären Umfeld. Nichtsdestotrotz entstehen hierdurch nicht nur Vorteile für die Nutzer dieser Software sowie im Entwicklungsprozess. Aus diesen Gründen soll diese Arbeit einen Überblick über bekannte Fehler in Open-Source-Projekten bieten. Während eine tiefgreifende Aufarbeitung dieser Fehler und der entsprechenden Ursachen den Rahmen weit sprengen würde, ist trotzdem zu verdeutlichen, dass diese Fallstricke keinesfalls aufgrund mangelnder Fähigkeiten oder mangelndem Wissen zu Tage getreten sind. Ziel dieser Arbeit ist also keinesfalls, einzelne Akteure bloßzustellen, sondern viel mehr anderen Entwicklern vor Augen zu führen, welche Fehler sich vermeiden lassen. Selbstverständlich werden hier Fallstricke angesprochen, die auch proprietäre Software betreffen. Grundsätzlich kann daher das reine Wissen über diese Fehler schon als Vorteil von FOSS gesehen werden. Durch die Öffentlichkeit in der Software-Entwicklung im FOSS-Ökosystem werden diese Fehler schließlich erst bekannt. Nichtsdestotrotz erleichtert die kollaborative Softwareentwicklung diese Fehler häufig, wie auch in den einzelnen Beispielen explizit erwähnt. Gleichzeitig werden - insbesondere in dem Kapitel 2.1.1 auch Fehler oder unglückliche Gegebenheiten dargestellt, die im Umfeld von Firmen aufgetreten sind. Hier ist zu erwähnen, dass dies keinesfalls ein schlechtes Licht auf diese Firmen werfen soll, sondern die Fälle rein beispielhaft dienen.

2 Fehler & problematische Nutzung

2.1 Spannungsfeld zwischen FOSS und kommerzieller Software

2.1.1 Recht zur Modifikation v. SOHO-Routern

Im Jahr 2010 vertrieb die Berliner Firma AVM, welche Small-Office, Home-Office (SOHO) & Consumer-Router herstellt, Geräte der Produktlinie »FritzBox Fon« mit einer entsprechenden eingebetteten Firmware. Diese Firmware enthielt u.A. - neben einem gesamten Linux-Kernel auch Teile der GPLv2[4]-lizensierten Software »netfilter« (später »iptables«). Gleichzeitig bot eine weitere Firma eine Software an, welche einen Teil der Firmware auf den Rechner des Nutzers herunterlud, modifizierte und auf den Geräten wieder installierte. Ziel war hier, eine andere Filtereinstellung für den Netzwerkverkehr umzusetzen[5].

Problematik

Der Hersteller AVM verklagte daraufhin den Hersteller der Modifikations-Software auf Unterlassung, da aus Sicht der Firma AVM gegen die Lizenzvereinbarungen der »Fritzbox Fon«-Firmware verstoßen wurde. Wurde dieser Klage zwar in erster Instanz recht gegeben, hob das Gericht dies in der Revision wieder auf. Mit juristischer Unterstützung der Free Software Foundation Europe (FSFE)[6] erging hier schlussendlich ein Urteil[7], wonach die GPL explizit das Recht zur Bearbeitung enthält und somit die Modifikations-Software durchaus mit der Lizenz harmoniert.

Alternativen

Das Ziel der Firma AVM ist dabei durchaus verständlich. Eine Nutzung Ihrer Produkte - einschließlich der Firmware - ausschließlich nach einem einheitlichen Firmwarestand kann den Supportaufwand drastisch reduzieren. Jedoch hätte dieses Ziel bereits bei der Entscheidung für oder gegen die Nutzung von FOSS-Software berücksichtigt werden müssen.

2.1.2 Potentieller Fehlgebrauch von Lizenzrechten

Die folgende Problematik hängt eng mit dem Fall von AVM zusammen. Während das oben genannte Verfahren durchaus im Sinne der FOSS-Bewegung war, nämlich dem Nutzer eine Modifikation zu erlauben, folgten weitere lizenzrechtliche Auseinandersetzungen zwischen dem ursprünglichen Autor des »netfilter«-Moduls Patrick McHardy und mehreren Herstellern eingebetteter Systeme. Da diverse Hersteller von ebendiesen Systemen nicht den Veröffentlichungspflichten der GPL-Lizenz nachkamen, mahnte Patrick McHardy diversen dieser Firmen ab, und verlangte teilweise horrende Summen für den Fall der Wiederholung.

Diese Summen sollten jeweils an Ihn als Inhaber des entsprechenden Urheberrechts fließen[8]. Auch wenn genaue Zahlen nicht bekannt sind, wird vermutet dass rund 50 solcher Verfahren angestrengt wurden.

Problematik

Auch wenn McHardy - wohl auch aufgrund der öffentlichen Berichterstattung - seine Klagen bzw. Abmahnungen teilweise zurückzog [9], zeigt sich hier doch wie sehr die Nutzung von FOSS-Software von dem Wohlwollen aller beteiligten Autoren abhängig ist. Zwar ist es durchaus im Sinne der FOSS-Gemeinde, dass die Lizenzvereinbarungen auch eingehalten werden, jedoch stellt sich dieser Fall fundamental anders dar. Durch die Abmahnungen und damit auch verbundenen Kosten entsteht eine erhebliche rechtliche Unsicherheit, inwiefern FOSS-Software in kommerziellen Produkten genutzt werden kann. Dabei dient die Androhung einer Strafzahlung an einen einzelnen Softwareentwickler sicherlich nicht der Weiterentwicklung des gesamten Projektes.

2.2 Sicherheitsimplikationen in FOSS

2.2.1 Stackoverflow - Copy & Paste

Eine Option, die sich durch die Verbreitung von Open-Source-Software weitaus mehr verbreitet hat, und auch durchaus gewünscht ist, ist die Wiederverwendung fremden Codes. Während dies einer möglichst effektiven Softwareentwicklung grundsätzlich dient, ergeben sich erhebliche Schwierigkeiten, wenn Code ungeprüft übernommen wird. Hier soll daher als Beispiel aufgezeigt werden, wie von dem populären Entwicklerforum »Stackoverflow« Code mit schwerwiegenden Sicherheitslücken übernommen wird und somit auch in fertige Softwareprodukte einfließt.

Problematik

Während das reine Übernehmen von Code an sich kein Problem darstellt, kann dies spätestens dann problematisch werden, wenn der kopierte Code vom Übernehmenden nicht ausreichend geprüft bzw. weiterentwickelt wird. Insbesondere in sicherheitskritischer Software, in denen ein tiefes Verständnis der zugrundeliegenden Bibliotheken zur korrekt Nutzung notwendig wäre, können die Auswirkungen einer fehlerhaften Übernahme katastrophal sein. Beispielhaft lässt sich hier die Übernahme von Code mit potentiellen SQL-Injections[10] nennen. Eine hierzu angestellte regelmäßige - leider rein quantitative und damit nur begrenzt aussagekräftige - Untersuchung[11] zeigt hier trotz der allgemeinen Bekanntheit über solche Sicherheitslücken nach wie vor monatlich Fragen mit potentiell verwundbarem Code im hohen dreistelligen Bereich, teilweise sogar vierstellig. In einer weiteren Untersuchung aus dem Jahr 2017 zeigte sich, das rund 4000 sicherheitsrelevante Code-Snippets mit Sicherheitslücken von Stackoverflow in ca. 195.000 Android-Applikationen auftreten[12][1]. Anhand der schieren Anzahl der Applikationen zeigt sich, das eine ungeprüfte Übernahme von Code teilweise enorme Ausmaße annehmen kann. Auch wenn es sich bei den in [12] untersuchten Applikationen nicht um FOSS-Produkte handelt, so ist hier klar zu erkennen, dass bereits die reine Veröffentlichung von unsicherem Code oder Code-Ausschnitten eine problematische Nutzung durch Dritte nach sich ziehen kann.

2.2.2 Heartbleed - Herkunft

Während die Sicherheitslücke Heartbleed[13] in der Verschlüsselungs-Bibliothek OpenSSL sowohl aus Sicht der Auswirkungen, wie auch der technischen Ursachen sicherlich allgemein bekannt ist, stellt sich hier insbesondere die Frage nach der Herkunft einer solch gravierenden Sicherheitslücke. Inwieweit diese Lücke nicht auch in einer kommerziellen Bibliothek hätte auftreten können ist dabei eine rein spekulative Frage und soll daher hier aussen vor gelassen werden.

[1] Es wurden 1,3 mio. Applikationen auf die Existenz von 4019 sicherheitsrelevanten Code-Ausschnitten untersucht. 15,5% dieser Applikationen enthielten einen Codeausschnitt, und davon 97,9% mindestens einen mit einer Sicherheitslücke

Herkunft & Vorgeschichte

Im Februar 2012 wurde mit dem Request for Comments (RFC) 6520 [14] der Vorschlag für ein sog. Heart**beat**-Verfahren bei der IETF veröffentlicht. Ziel dieser Protokollerweiterung war es, durch regelmäßige Nachrichten mit frei wählbarem, nicht protokollrelevantem Nutzinhalt, eine TLS-Verbindung aufrecht zu erhalten. Im Zuge des Vorschlages wurde vom ursprünglichen Autor eine Implementierung dieser Erweiterung als Patch beim OpenSSL-Projekt eingereicht. Auch wenn das Feature kaum Nutzung erfuhr, war damit entsprechender Code in der OpenSSL-Bibliothek vorhanden, der den entsprechenden Fehler enthielt[15]. Erst rund zwei Jahre später (April 2012) wurde die in diesem Code vorhandene Sicherheitslücke - ein klassischer Out-Of-Bounds-Read[16] - entdeckt, mit fatalen Folgen. Hier zeigt sich, das auch öffentlicher Code nicht unbedingt einer uneingeschränkten Kontrolle unterliegt. Insbesondere wird hier die Tatsache, dass sich Code mit relativ geringen Hürden in ein Projekt einbringen lässt, zum Problem. Ob ein kaum bis nicht genutztes Feature in einem großen Softwareprojekt nötig ist, welches zusätzliche Komplexität und damit einhergehende Anfälligkeit für Sicherheitslücken schafft, lässt sich insbesondere an diesem Beispiel hinterfragen.

2.3 Organisatorische Probleme & Verfügbarkeit

2.3.1 NPM-Verfügbarkeit

Der Node-Package-Manager (NPM) bildet zusammen mit NodeJS ein relativ komplexes Softwareentwicklungs-Ökosystem für die Programmierung mit (auch serverseitigem) Javascript. Aufgrund des sehr minimalistischen Funktionsumfanges der Sprache an sich (es gibt in dem Sinne keine Standardbibliothek) ist eine Vielzahl an Bibliotheken (sog. Packages) entstanden, welche oft auch kleine Funktionalitäten einfach installierbar und zugänglich machen. Das System basiert dabei auf einem zentralen Repository, welches von der Firma NPM Inc. [17] zur Verfügung gestellt wird. Im Jahr 2013 entschied ein Entwickler aufgrund von Differenzen mit der Firma NPM Inc., seine Pakete aus dem Repository zu entfernen. Unter anderem hatte er das Paket »left-pad« [18], welches einen String mit einem Padding versieht, entwickelt. Zwar ist die Funktionalität dieses Paketes in wenigen Zeilen auch problemlos von jedem Entwickler zu realisieren, wurde jedoch aufgrund der einfachen Verfügbarkeit in mehreren Tausend anderen Javascript-Projekten genutzt. Unter anderem basierte auch der Kompilierprozess des eigentlichen Javascript-Interpreters, NodeJS auf der Verfügbarkeit ebendiesen Paketes. Die Folgen der Löschung des Paketes waren ein kurzzeitiger Kollaps von diversen großen Softwareprojekten[19].

Abhilfe

Hier zeigt sich, dass die Verfügbarkeit von FOSS nicht unbedingt dauerhaft garantiert ist. Ist ein Softwareprodukt bspw. im Deployment-Zyklus auf die Verfügbarkeit eines FOSS-Moduls angewiesen, sollten hier die entsprechenden Besonderheiten freier Software stärker berücksichtigt werden. So wären durch ein automatisches Erhalten der letzten Version die Auswirkungen erheblich geringer ausgefallen. Zwar wird regelmäßig die Entwicklung von FOSS-Projekten eingestellt, jedoch ist dies üblicherweise ein schleichender Prozess, sodass bei den Nutzern ein entsprechender zeitlicher Puffer existiert, um die Entwicklung einer Alternative voranzutreiben.

3 Fehler vermeiden & Fazit

3.1 Fazit für die FOSS-Entwicklung

Grundsätzlich lässt sich bei allen angesprochenen Beispielen sagen, dass insbesondere die Art des Softwareentwicklungsprozesses in FOSS-Projekten nicht ausschließlich Vorteile mit sich bringt. So fehlen teilweise Strukturen bzw. Konzepte, welche Ziele ein Open-Source-Projekt hat. Eine Qualitätskontrolle ist bei FOSS-Projekten zwar durch die öffentliche Entwicklung grundsätzlich einfacher zu realisieren, bzw. durch die Öffentlichkeit an sich schon gewährt, jedoch sind hier oft keine klaren Strukturen festgelegt. Diese begünstigen - wie beim Heartbleed-Bug etwa, das ungenutzter Code in Software einfliesst, ohne dass dieser einer ausreichenden Qualitätskontrolle unterlag. Auch die Frage nach der zu wählenden Lizenz sollte mit allen Implikationen bedacht werden. Wie im Kapitel 2.1.2 beschrieben, kann eine unklare Formulierung, welche Rechte der einzelne Beitragende hat hier durchaus dazu führen, dass die Nutzung der Software starken Unsicherheiten unterliegt. Der Fall der mangelden Verfügbarkeit eines einzelnen NPM-Paketes zeigt auch, dass u.U. die Entwickler von FOSS-Software nicht immer überschauen können, welche Nutzung Ihre Software erfährt. Hier konnte ein einzelner Entwickler ein gesamtes Ökosystem kurzfristig lahmlegen, was sicherlich nicht unbedingt böser Wille war. Viel mehr zeigt sich, dass der Entwickler sich der Relevanz seiner Software für das Software-Ökosystem und deren Nutzer nicht bewusst war.

3.2 Fazit für die FOSS-Nutzung

Auf der Seite der Nutzer von FOSS-Software zeigt sich immer wieder, dass die Lizenzrechtlichen Fragen nach wie vor unzureichend geklärt werden. Dies kann einerseits darin zeigen dass die Firmen, welche FOSS nutzen nicht beachten, dass den Kunden damit teilweise auch Rechte eingeräumt werden, welche den legitimen Zielen der Firmen u.U. zuwider laufen. So ist es - wie in 2.1.1 beschrieben - durchaus verständlich, dass ein Hersteller von Consumer-Geräten ein Interesse an einer unmodifizierten Firmware hat, um beispielsweise den Wartungsaufwand zu reduzieren. Gleichzeitig zeigt der in 2.1.2 beschriebene Fall, das viele Firmen nach wie vor die Vorgaben der entsprechenden Lizenzen schlicht ignorieren. Über die Frage, ob in diesem Fall ein anderes Verhalten des Entwicklers angemessener gewesen wäre lässt sich zwar sicherlich diskutieren, doch wäre den rechtlichen Auseinandersetzungen durch eine Beachtung der Lizenzen ohnehin jede Grundlage genommen worden. Als vollkommen anderer Problemkomplex zeigt sich die Qualitätsfrage bei der Nutzung von FOSS oder auch nur kleiner Code-Ausschnitte. Hier sollte für alle Nutzer verbindlich

geregelt sein, welche Maßnahmen für Sicherstellung einer dauerhaften Verfügbarkeit von Software Dritter getroffen werde. Auch kann es hilfreich sein, bei der Übernahme von Code eine entsprechende Dokumentation vorzunehmen, und diesen explizit auf seine korrekte Verwendung zu prüfen.

Literatur

[1] URL: https://opensource.google.com/ (besucht am 06. 09. 2018).

[2] URL: https://opensource.microsoft.com/resources (besucht am 06. 09. 2018).

[3] URL: https://developer.apple.com/opensource/ (besucht am 06. 09. 2018).

[4] URL: https://www.gnu.org/licenses/gpl-2.0.txt (besucht am 06. 09. 2018).

[5] URL: https://www.heise.de/newsticker/meldung/FSFE-wirft-AVM-GPL-Verletzung-vor-1263357.html (besucht am 06. 09. 2018).

[6] URL: https://fsfe.org/activities/ftf/avm-gpl-violation.de.html (besucht am 06. 09. 2018).

[7] URL: https://fsfe.org/activities/ftf/kg-avm-vs-cybits.pdf (besucht am 06. 09. 2018).

[8] URL: https://opensource.com/article/17/8/patrick-mchardy-and-copyright-profiteering (besucht am 06. 09. 2018).

[9] URL: https://www.heise.de/newsticker/meldung/Linux-Klaeger-McHardy-zieht-Antrag-gegen-Elektronik-Hersteller-zurueck-3988556.html (besucht am 06. 09. 2018).

[10] URL: https://www.cs.montana.edu/courses/csci476/topics/sql_injection.pdf (besucht am 06. 09. 2018).

[11] URL: https://laurent22.github.io/so-injections/ (besucht am 06. 09. 2018).

[12] URL: https://saschafahl.de/papers/stackoverflow2017.pdf (besucht am 06. 09. 2018).

[13] URL: https://nvd.nist.gov/vuln/detail/CVE-2014-0160 (besucht am 06. 09. 2018).

[14] URL: https://tools.ietf.org/html/rfc6520 (besucht am 06. 09. 2018).

[15] URL: https://git.openssl.org/gitweb/?p=openssl.git;a=blobdiff;f=ssl/d1_both.c;h=0a84f957118afa9804451add380eca4719a9765e;hp=89338e9430f2865f21a70da0e
hb=4817504d069b4c5082161b02a22116ad75f822b1;hpb=84b6e277d4f45487377d0159e82c356d
(besucht am 06. 09. 2018).

[16] URL: https://www.theregister.co.uk/2014/04/09/heartbleed_explained/ (besucht am 06. 09. 2018).

[17] URL: https://www.npmjs.com/about (besucht am 06. 09. 2018).

[18] URL: https://github.com/stevemao/left-pad/blob/master/index.js (besucht am 06. 09. 2018).

[19] URL: https://www.theregister.co.uk/2016/03/23/npm_left_pad_chaos/ (besucht am 06. 09. 2018).