Proseminar *Entwicklung sicherer Software*
Wintersemester 2008/2009

Sicherheit in Open-Source-Software

Chancen und Risiken

Sebastian Kürten
kuerten@inf.fu-berlin.de

Betreuer: Dipl. Medieninf. Martin Gruhn

10.03.2009

Zusammenfassung

Zunächst werden Argumente präsentiert, die für eine erhöhte Sicherheit in Open-Source-Software (OSS) sprechen. Es wird auf den typischen Entwicklungsprozess von OSS und auf dessen Stärken eingegangen. Da OSS auch potentiellen Angreifern Vorteile bietet, wird außerdem diskutiert, welche Vorteile diesen gegeben sind und welche Möglichkeiten Anwender haben, sich dagegen zu wehren. Hierbei wird auf die besondere Freiheit und Flexibilität von OSS eingegangen. Desweiteren werden Probleme bezüglich des Vertrauens gegenüber den Herstellern einer Software erläutert.

Gliederung

1. Einleitung

Open-Source-Software (OSS) kann unterschiedliche Vorteile gegenüber Closed-Source-Software (CSS) haben. Gerade weil viele dieser Vorteile nicht im Bereich der Sicherheit angesiedelt sind, ist es besonders interessant, wie sicher die Verwendung solcher Software überhaupt ist. Dieser Frage wird in dieser Ausarbeitung nachgegangen.

2. Definitionen

Open-Source-Software

Unter *Open-Source-Software* wird hier Software verstanden, deren Quelltext verfügbar ist. Darüberhinaus beschränken sich die Ausführungen auf Software, die nicht nur verwendet werden darf, sondern für die es ebenfalls erlaubt ist, diese zu verändern und sogar veränderte Versionen der Software zu vertreiben. Ob die Software kostenlos ist oder nicht, soll hierbei keine Rolle spielen.

Sicherheit

Unter *Sicherheit* soll hier primär verstanden werden, wie *verletzlich* eine Software ist. Sicherheit wird im weiteren also in etwa gleichgesetzt mit der Anzahl von Schwachstellen in einer Software. Ein besonderer Fokus liegt auf dem *Schutz* eines Anwenders vor unautorisierter Handlung Dritter mit der von ihm verwendeten Software.

3. Warum sollte Open-Source-Software sicher sein?

Der Entwicklungsprozess von OSS unterscheidet sich grundlegend von dem proprietärer Software. Während letztere klassischerweise von einem Unternehmen entwickelt wird, entsteht OSS meist durch die Zusammenarbeit von Freiwilligen auf der ganzen Welt. Im folgenden soll betrachtet werden, warum dieser Prozess sich positiv auf die Sicherheit von OSS auswirken kann.

3.1. Das Viele-Augen-Prinzip

Idealerweise hat eine freie Software sehr viele Anwender und insbesondere tragen auch viele dieser Anwender zur Entwicklung bei. Eric Raymond bezeichnet folgendes als Linus' Law[5]: Je mehr Leute sich den Quelltext ansehen, desto wahrscheinlicher wird die Entdeckung von Bugs. Ist ein Bug erst einmal identifiziert, so wird jemand eine Korrektur dafür finden und bereitstellen.

Die große Menge der Anwender beansprucht eine Software auf unterschiedlichste Weise, wodurch die meisten Bugs entdeckt werden[5]. Wurden Bugs korrigiert,

wird zeitnah eine neue Version der Software herausgegeben, so dass die Anwender nun als Tester für diese fungieren können. Dadurch wird das Testen in hohem Maß parallelisiert[5]. Die Anwender können die Entwickler meist mit reichhaltigen Fehlerberichten versorgen, welche das Reproduzieren einer problematischen Situation und das Beheben eines Bugs erleichtern können[5].

Durch diese Vorgänge enthält die Software immer weniger Bugs und langfristig wird die Zahl der Bugs sehr gering werden. Dadurch wird die Software sicherer, da mit der Anzahl der Bugs auch die der Schwachstellen abnimmt[5].

3.2. Das Peer-Review-Verfahren

Eine wichtige Praktik zur Überprüfung der Güte des Quelltextes stellt das sogenannte Peer-Review-Verfahren dar. Hierbei werden Änderungen am Quelltext von anderen Entwicklern überprüft[4]. Ein besonders strenge Handhabung dieses Verfahrens ist es, wenn diese Überprüfung stattfinden muss, bevor Änderungen in eine Software übernommen werden. Dieses Verfahren wird allerdings nicht in allen Projekten angewendet. Ein Beispiel[4] hierfür ist jedoch der Linux-Kernel, bei dem jede Änderung von vertrauenswürdigen Entwicklern als überprüft signiert werden muss. Das hierfür nötige Vertrauen haben sich viele Entwickler durch jahrelange Mitarbeit am Projekt erarbeitet.

4. Welche Risiken bestehen?

Der Entwicklungsprozess von OSS bringt jedoch nicht nur Vorteile mit sich. Im folgenden wird betrachtet, welche Risiken sich durch diesen ergeben.

4.1. Fälschliche Annahme von Kontrolle

Damit das Viele-Augen-Prinzip seine Wirkung entfalten kann, müssen, der Komplexität einer Software angemessen, viele Leute tatsächlich mit dem Quelltext interagieren[6]. Ist dies in einem Projekt nicht der Fall, können Anwender fälschlicherweise dennoch annehmen, dass diese Kontrolle des Quelltextes stattfindet[3]. Wenn die Mehrzahl der Anwender diese Sicherheit annehmen und infolgedessen nur sehr wenige die Qualität einer Software überprüfen, wird die Vorrausetzung für die positiven Effekte des Viele-Augen-Prinzips genommen, nämlich dass möglichst viele Freiwillige sich mit der Funktionsweise der Software auseinandersetzen. So kann es gut passieren, dass sich die Anwender in einer Sicherheit wiegen, die nicht im erwarteten Umfang existiert[3]. Unterschiedliche Faktoren können außerdem die tatsächliche Anzahl der kontrollierenden Augenpaare reduzieren, wie beispielsweise, dass eine Software wenig genutzt wird, sie ein Nischenprodukt ist, es einfach wenige Entwickler gibt oder eine wenig verbreitete Programmiersprache verwendet wird[8].

4.2. Fragliche Qualifikation der Entwickler

Selbst wenn genug Leute am Quelltext arbeiten, ist fragwürdig, ob diese die nötige Qualifikation haben, um sichere Software zu schreiben, denn oft gibt es keine Auswahl der Programmierer auf Basis ihrer Fähigkeiten[2]. Dies ist stark projektabhängig. So wie es Projekte gibt, die von Hobbyentwicklern geführt werden, die gerade gelernt haben zu programmieren, so gibt es andere Projekte, in denen professionelle Entwickler mit jahrelanger Erfahrung tätig sind. Jedoch können selbst die Fähigkeiten von professionellen Entwicklern im Bereich der Sicherheit in Frage gestellt werden, denn auch diese sind nicht unbedingt gleichzeitig Sicherheitsexperten[6,9].

4.3. Kontraproduktivität der Transparenz

Die Augen der Angreifer

Während die Einsicht in den Quelltext und die Nutzung der Kommunikationsplattformen einerseits von den Entwicklern genutzt werden, um eine Software weiterzuentwickeln und Bugs zu beheben, können Angreifer diese Transparenz nutzen, um gezielt nach Schwachstellen zu suchen. Der Quelltext kann dabei eine große Hilfe darstellen[1].

Bekanntwerden von Bugs

Die meisten OSS-Projekte behandeln die Verarbeitung von bekannt gewordenen Schwachstellen öffentlich[1]. Zudem verfolgen viele einschlägige OSS-Entwickler das Prinzip der „vollkommenen Veröffentlichung“[7]: Wird eine Schwachstelle entdeckt, so wird unter Umständen ein passender Exploit veröffentlicht, um die Ausnutzbarkeit zu demonstrieren. Dadurch kann sich gemäß Linus' Gesetz eine größtmögliche Zahl von Leuten dem Problem annehmen. Neben den Entwicklern profitieren von diesem Verhalten aber insbesondere die Angreifer[7]. Während ein fertiger Exploit diesen auf dem Silbertablett serviert wird, ist auch das Wissen über das Vorhandensein einer Schwachstelle schon sehr viel wert und erleichtert die Erarbeitung eines Exploits erheblich[1]. Eine mögliche Maßnahme hiergegen ist, dass Projekte die Diskussionen um Schwachstellen in private Bereiche verlagern. Die Entwickler der Software haben immer noch Zugriff auf diese wichtigen Informationen, potentielle Angreifer werden aber weitestgehend ausgeschlossen. Diese Methode wird jedoch nicht von allzu vielen Projekten angewendet. In manchen ist dies aber durchaus üblich und es ist die Tendenz zu erkennen, dass immer mehr Projekte zu dieser Strategie wechseln[1].

Veröffentlichung von Patches

Jedoch selbst wenn Schwachstellen in privaten Bereichen behandelt werden, so muss letztendlich ein Patch veröffentlicht werden, um eine Schwachstelle zu beheben. Bei OSS kann ein Angreifer anhand der vorgenommenen Änderungen am Quelltext genau erkennen, welcher Teil einer Software problematisch war. Diese Information ist natürlich ähnlich hilfreich, wie die Kenntnis der Schwachstelle selbst[1]. Da es mehr oder weniger lange dauert, bis alle Anwender einen Patch ein-

gespielt haben oder auf eine neue Version umgestiegen sind, ergibt sich unweigerlich ein Zeitraum, in dem Angreifer ein leichtes Spiel haben. Hier ist die Geheimhaltung des Quelltextes bei CSS ein Vorteil, denn anhand eines Patches ist kaum erkennbar, ob und wo eine Sicherheitslücke geschlossen wurde[1]. Wenn niemand weiß, dass es eine Schwachstelle in einer proprietären Software gibt, ist es auch weniger wahrscheinlich, dass diese ausgenutzt wird[4].

4.4. Böswillige Entwickler

Natürlich können auch Leute zu einem Projekt beitragen, die böse Absichten verfolgen. Diese sind selbst Angreifer und können versuchen, bewusst eine Hintertür einzubauen oder eine Schwachstelle, welche sich zur Verwendung als solche eignet. Bei OSS-Projekten, in denen es relativ leicht ist mitzuwirken, wird begünstigt, dass derartige Leute Erfolg haben können[8]. Jedoch spricht die Natur von OSS auch dagegen, denn die jeweilige Schwachstelle kann durch andere Entwickler entdeckt werden und in der Regel ist der Übeltäter dadurch enttarnt[4]. Durch solches Verhalten setzt der Entwickler das Vertrauen der anderen aufs Spiel und verbaut sich somit die Möglichkeit, weiter am Projekt mitarbeiten zu können. Insbesondere in Projekten mit Peer-Review-Verfahren wird es recht schwer, eine Hintertür einzuschleusen[4]. Daneben sei erwähnt, dass es auch in Firmen proprietärer Software solche Leute geben kann. Hier ist es jedoch besonders unwahrscheinlich, dass eine absichtlich eingefügte Hintertür gefunden wird, da wenige Firmen ihren Code einer internen Kontrolle unterziehen[8].

4.5. Böswillige Distributoren

Die wenigsten Nutzer von OSS kompilieren ihre Software selbst[4]. Hierfür gibt es sicherlich Ausnahmen und insbesondere auch Projekte, bei denen besonders viele Anwender ihre Software selbst aus den Quelltexten erstellen. Unter Annahme der ersten Möglichkeit werden jedoch in irgendeiner Form vorkompilierte Binärdateien verwendet, die entweder von den Entwicklern einer OSS selbst oder von einem anderen Distributor erstellt wurden. An dieser Stelle ist zusätzliches Vertrauen in denjenigen notwendig, der diese Binärdateien aus den Quellen erstellt. Denn grundsätzlich gibt es keine Garantie dafür, dass der zu einer Binärdatei präsentierte Quelltext mit dem übereinstimmt, aus dem das Kompilat erstellt wurde[3]. Ein gutes Beispiel[4] hierfür sind diverse Linux-Distributionen, bei denen so gut wie alle Softwarepakete in kompilierter Form heruntergeladen werden. Zwar können derartige Pakete digital signiert werden, sodass zumindest die Identität des Produzenten verifizierbar ist. Jedoch stellt dies noch nicht sicher, dass nicht dieser selbst Modifikationen am Quelltext vorgenommen hat, bevor dieser kompiliert wurde[3].

5. Welche Chancen hat der Anwender?

5.1. Kontrolle der Software und des Entwicklungsprozesses

Da der Entwicklungsprozess meist öffentlich einsehbar ist, können Anwender nachvollziehen, in welcher Weise mit sicherheitsrelevanten Defekten in einer Software umgegangen wird[8]. Es lässt sich überprüfen, wie schnell in der Vergangenheit Patches verfügbar gemacht wurden und welche Qualität diese haben. Darüberhinaus lässt sich auch einschätzen, wie gut ein Projekt im Allgemeinen organisiert ist und welche Mechanismen eingesetzt werden, um sichere Software zu produzieren. Daneben kann man sich auch durch eigenhändige Überprüfung des Quellcodes von der sicheren Implementierung eines Features überzeugen[2]. Dies kann hilfreich sein, wenn es in einer bestimmten Umgebung besonders auf die Sicherheit eines bestimmten Teils einer Software ankommt.

5.2. Modifikation der Software

Eins der Probleme, das unweigerlich zu Unsicherheit führt, ist die Komplexität einer Software[7]. Wenn diese verringert werden kann, sinkt auch die Fehleranfälligkeit[2]. Wenn also ein Anwender nicht den vollen Funktionsumfang einer Software benötigt, so kann er die nicht benötigte Funktionalität herausnehmen[1]. Dadurch wird der Quelltext verkürzt und die Komplexität verringert.

Stellt man bei der Überprüfung einer Software Mängel oder Verbesserungsmöglichkeiten fest, so kann man diese außerdem durch Modifikation beheben. Hierzu kann entweder versucht werden, die entsprechenden Änderungen in das Projekt selbst einfließen zu lassen oder selbst eine modifizierte Version der Software herzustellen[4]. Ein gutes Beispiel[4] hierfür ist das sogenannte Security-Enhanced Linux, welches vom amerikanischen Geheimdienst NSA spezifiziert und implementiert wurde. Hierbei handelt es sich um eine modifzierte Version des Linux-Kernels, der besonderen Sicherheitsansprüchen gerecht wird. Das Projekt wird seit der Veröffentlichung von der Community gepflegt und weiterentwickelt. Eine derartige Modifikation wäre bei CSS selbst für eine einflussreiche Organisation wie den NSA kaum möglich gewesen.

6. Zusammenfassung

Das Viele-Augen-Prinzip und das Peer-Review-Verfahren können sich positiv auf die Sicherheit einer Software auswirken. Fraglich ist jedoch einerseits, wie viele Leute an einer Software arbeiten und andererseits, wie fähig diese sind, sicheren Programmcode zu verfassen. Hier kommt es stark darauf an, wer in einem konkreten Projekt tatsächlich tätig ist. Durch den öffentlich einsehbaren Entwicklungsprozess ist es zumindest möglich, sich als Anwender ein Bild davon zu verschaffen.

Ohne Zweifel profitieren aber nicht nur die Anwender und Entwickler vom offenen Quellcode und Entwicklungsmodell sondern auch die Angreifer. Die Offenheit ist also sowohl Vor- als auch Nachteil. Was hier überwiegt ist schwer abzuschätzen und nicht abschließend geklärt. Ein klarer Vorteil hingegen ist die Möglichkeit, angepasste Versionen einer Software erstellen zu können, damit diese in bestimmten Umgebungen oder Funktionen besonders sicher ist.

Besonders wichtig ist allerdings, dass Nutzer sich der Risiken bewusst sind und sich auch einen Einblick verschaffen, wie hoch der Sicherheitsstandard innerhalb einer Software ist. Denn die Sicherheit von OSS steht und fällt mit einer möglichst großen Anzahl kritischer Augen, die auf die Software im Ganzen, aber insbesondere auch auf deren Design und Quelltext gerichtet sind. Man sollte sich hier nicht in einer Sicherheit wähnen, die möglicherweise gar nicht vorhanden ist.

Literatur

[1] Ford, R., 2007. *Open vs. Closed: Which Source is More Secure?*
ACM Queue, Volume 5, Issue 1, pp. 32-38

[2] Hoepman, J.-H. , Jacobs, B., 2007. *Increased Security Through Open Source*
Communications of the ACM, Volume 50, Issue 1, pp. 79-83

[3] Levy, E., 2000. *Wide Open Source*
Security Focus (http://www.securityfocus.com/news/19)

[4] Payne, C., 2002. *On the security of open source*
Information Systems Journal, Volume 12, Issue 1, pp. 61-78

[5] Raymond, E. S., 2002. *The Cathedral and the Bazaar*
O'Reilly (http://www.catb.org/~esr/writings/cathedral-bazaar/)

[6] Viega, J., 2004. *Open Source Security: Still a Myth*
O'Reilly (http://www.onlamp.com/pub/a/security/2004/09/16/open_source_security_myths.html)

[7] Viega, J., McGraw, G., 2002. *Building Secure Software - How to Avoid Security Problems the Right Way*
Addison-Wesley

[8] Wheeler, D., 2003. *Is Open Source Good for Security?*
(http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/open-source-security.html)