

Open Source Security

Information Rules 1 WS 2003/2004
7. Januar 2004



GRUPPE 12:
Wolfram Riedel
Dincel Atac
Birol Kayapinar

Lehrveranstalter:
Prof. Dr. iur. Bernd Lutterbeck,
Dipl.-Inform. Robert Gehring,
Dipl.-Inform. Frank Pallas

Inhaltsverzeichnis

| | | |
|----------|--|-----------|
| 1 | Einleitung | 2 |
| 2 | Sicherheitsaspekte der Softwareentwicklung | 3 |
| 2.1 | Softwareentwicklung in der Open Source Community | 3 |
| 2.2 | Viele-Augen-Theorie | 5 |
| 2.3 | Peer Review | 5 |
| 2.4 | Code Audit | 6 |
| 2.5 | die Binärpaket-Problematik | 7 |
| 2.6 | Softwareentwicklung in der Closed Source Company | 7 |
| 2.7 | Mängel aus ökonomischen Gründen | 8 |
| 2.8 | Produkthaftung | 9 |
| 2.9 | Vergleich der Entwicklungsmodelle | 9 |
| 3 | Angriffe und Gegenmaßnahmen | 11 |
| 3.1 | Security by Obscurity | 11 |
| 3.2 | Angriffe im Closed Source Bereich | 12 |
| 3.3 | Angriffe im Open Source Bereich | 12 |
| 3.4 | Open Source Security Tools | 14 |
| 3.4.1 | Kerberos | 14 |
| 3.4.2 | OpenSSL | 15 |
| 3.4.3 | Webservices | 15 |
| 3.5 | Empirischer Vergleich: Apache vs. IIS | 17 |
| 4 | Open Source in der Politik | 18 |
| 4.1 | e-Government mit Open Source Software | 18 |
| 4.1.1 | Common Criteria | 18 |
| 4.1.2 | Webservices für e-Government als Open Source | 19 |
| 4.2 | e-Voting mit Open Source Software | 20 |
| 4.2.1 | Sicherheitsanforderungen eines "Electronic Voting Systems" | 20 |
| 4.2.2 | Die Praxis | 21 |
| 5 | Fazit | 22 |

1 Einleitung

Ist Open Source Software sicherer als Closed Source Software?

Parallel mit der Entwicklung des Internets entstand eine Entwicklungsmethode für Software, die die Offenheit der Programmtexte propagiert. Pionier einer solchen Bewegung ist Richard M. Stallman, der die so genannte Free Software¹ als analog zu Free Speech (freie Rede) betrachtet.

Der Begriff und die Definition des Open Source² entstanden aus dem Bedürfnis, das Konzept der offenen Programmquellen auch für kommerzielle Verwendung interessanter zu machen und auf eine breitere Basis zu stellen, obwohl diese auch mit Free Software prinzipiell möglich gewesen wäre.

Der Entwicklung von kommerzieller Software ist ein eigener Industrie-Zweig gewidmet. Die Firmen machen den Quellcode traditionell allerdings nicht verfügbar, um das Stehlen von Code und dessen Funktionsweise zu verhindern. Diese Software nennt man zur Unterscheidung Closed Source.

Die Art wie man Software entwickelt – das sei an dieser Stelle betont – ist nicht notwendigerweise gebunden an die Offen- oder Verborgenheit des Sourcecodes. Kommerzielle Software kann sowohl Open Source sein genauso wie Closed Source allein keine Lizenzgebühren bedingt (z.B. Freeware). Zwei Entwicklungsmodelle sind jedoch sehr üblich und am interessantesten für den Vergleich: Die Softwareentwicklung in einer Internet-Community mit offenem Source und die Herstellung von Closed Source zur kommerziellen Vermarktung.

Sicherheit beruht immer auf einer Kette von Maßnahmen. Schon das Versagen eines einzelnen Gliedes führt oft zur Kompromittierung des gesamten Systems. Sicherheit von muss daher schon ganz am Anfang bei der Entwicklung von Software beginnen.

Softwaresicherheit lässt sich seitens der Entwicklung in drei Bereiche einteilen: Prüfung während der Entwicklung (Software Auditing), Compiler-Techniken, die mögliche Fehler begrenzen (Vulnerability Mitigation) und Gefahreneindämmung und Schadensbegrenzung durch Betriebssysteme (Behavior Management). (Cowan 2003)

Um die Sicherheit von Open Source Software zu untersuchen, ist es sinnvoll, auf den ersten dieser drei Aspekte einzugehen, denn dort sind die größten Unterschiede zu Closed Source Software zu erwarten. Wir werden daher als erstes die beiden gegensätzlichen Ansätze des offenen und verborgenen Quellcodes und deren übliche Entwicklungsmethoden näher untersuchen.

Sicherheitsprobleme treten fast immer erst beim Benutzer während der Anwendung von Software auf, auch können sie gerade erst durch Benutzungsfehler und falsche Konfiguration entstehen. Deswegen wollen wir im zweiten Teil die Sicherheitsprobleme beider Ansätze anhand bereits stattgefundenener, sicherheitsbezogener Vorkommnisse und Gefahren vergleichen.

Regierungen und Behörden weltweit setzen offene Software zunehmend ein. Als dritten Aspekt der Sicherheit möchten wir daher feststellen inwiefern sich das Konzept Open Source – und eine möglicherweise dazugewonnene Sicherheit

¹<http://www.gnu.org/philosophy/free-sw.html>

²<http://www.opensource.org/docs/definition.php>

– auf den Einsatz in demokratischen Gesellschaften, insbesondere auf die informationstechnologische Infrastruktur von Regierungen und bei der Durchführung von Wahlen, auswirkt.

2 Sicherheitsaspekte der Softwareentwicklung

2.1 Softwareentwicklung in der Open Source Community

Die Kontrolle über den Quellcode ermöglicht die Kontrolle über das Programm. Eine grundsätzliche Eigenschaft von Open Source Software ist die Freiheit, die gesamten Innereien auf ihre Funktionsweise überprüfen zu können. Theoretisch ist man jederzeit in der Lage zu wissen oder nachvollziehen zu können was eine Software tut und wie sie im Anwendungsfall, z.B. auch auf unvorhergesehene Eingaben, reagieren wird.

In der Praxis gibt es ein Hindernis, die die theoretisch mögliche Sicherheit von Open Source etwas einschränkt. Nicht jeder Benutzer von Computern ist gemäß seiner geistigen Fähigkeiten und zeitlichen Ressourcen in der Lage, selber Sourcecode zu lesen, zu verstehen oder zu verbessern. Privatnutzer ohne jegliches technisches Interesse werden von dieser Möglichkeit überhaupt keinen Gebrauch machen, aber auch informatisch versierte Anwender können als Einzelperson faktisch nicht den gesamten Quellcode aller Programme anschauen, die sie verwenden möchten.

Quellen von nicht-trivialen Programmen sind zudem riesige Werke, bestehend aus vielen tausenden Zeilen von Code. Allen verwendeten Source zu lesen ist für einen Einzelnen ein unmögliches Unterfangen. Doch vermag die Open Source Entwicklung zu funktionieren und obendrein stabile und zumeist auch sichere Software hervorzubringen. Wie das geschieht wollen wir uns nun genauer anschauen.

Ein Open Source Projekt beginnt mit einem Initiator oder einer Gruppe von Leuten, die Interesse oder Bedarf an der Entwicklung einer bestimmten Software haben. Die freiwilligen Teilnehmer sind motiviert aus wissenschaftlichem oder sportlichem Ehrgeiz oder weil sie konkrete Funktionalität vermissen. Zur Teilnahme an der Entwicklung bedarf es nur relativ einfacher und kostengünstig beschaffbarer Hilfsmittel wie z.B. Internetzugang und einiger Softwaretools, die es bereits als Open Source gibt.

Der Source und seine Funktionalität steht bei der gesamten Entwicklung im Mittelpunkt. Der Quellcode steht jedem z.B. per CVS³ oder als gepacktes Dateiarchiv zum Download zur Verfügung. Über Mailinglisten werden Vorgehensweisen und Probleme des Projekts diskutiert. Benutzer des Programms können sich an der Diskussion beteiligen und eigene Vorschläge und Berichte von Fehler einbringen. Die Meldung von Fehlern geschieht bei größeren Projekten oft über spezielle Bugtrackingsysteme⁴, die die Erfassung von Fehlermeldungen erleichtern und den Entwicklern helfen, die Übersicht zu behalten.

Ein wichtiger Punkt der Open Source Philosophie, der auch durch den offenen Code erst möglich wird, ist die weitgehende Wiederverwendung von Code. Anstatt ein neues Projekt von Null auf anzufangen schaut man sich zuerst nach bereits vorhandener Software um und passt diese gegebenenfalls an die eigenen

³<http://www.cvshome.org/>

⁴<http://www.bugzilla.org>

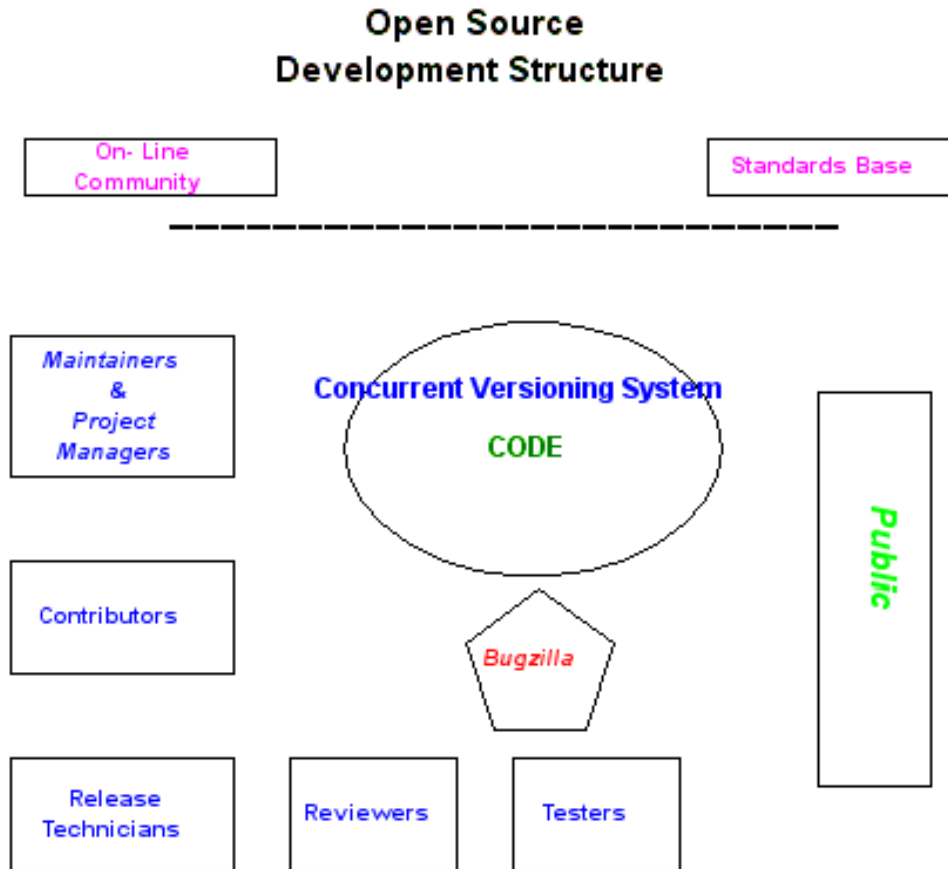


Abbildung 1: Entwicklung in der Open Source Community (Adelstein 2003)

Bedürfnisse an bevor man komplett neuen Code schreibt. Die wiederverwendeten Codeteile sind meistens bereits von den größten Fehlern bereinigt, sodass sie weniger Debuggingarbeit nach sich ziehen. Werden darin dennoch Fehler gefunden, so können entsprechende Hinweise oder Patches an das andere Projekt zurückfließen. Besonders oft verwendete Codeteile werden entsprechend oft begutachtet und erlangen schnell die Reife für einen produktiven Einsatz.

Die Sourcen werden kontinuierlich veröffentlicht. Genauer gesagt werden so genannte Release-Versionen bestimmt, die von Entwicklern und Testern als stabil eingestuft wurden. Aber es steht auch der aktuellste – und möglicherweise noch fehlerbehaftete – Sourcecode jederzeit zur Verfügung. Jeder kann ihn selber ausprobieren, nach Fehlern durchsuchen und so zur Stabilität der Software beitragen.

Die Open Source Philosophie beinhaltet neben der Verwendung von offenen Quellcodes auch die der offenen Standards für Schnittstellen und Dateiformate. Sie ermöglichen größtmögliche Interoperabilität zwischen Programmen und zu anderen Plattformen.

Kommerzielle Firmen verwenden nicht selten Open Source Software und sind deshalb auch manchmal in ihre Entwicklung involviert. Dann bezahlen sie ihre Mitarbeiter dafür, dass sie wie andere Entwickler der Community an Projekten mitarbeiten, oder sie stellen ältere Versionen ihrer Closed Source Software unter eine Open Source Lizenz.

Die Qualität von Sourcecode kann entscheidend zur Sicherheit beitragen. Code, der übersichtlich und leicht verständlich ist, offenbart Fehler meist schon beim Schreiben. Ausführliche Kommentare sind essentiell, um nicht selbst geschriebenen Code zu verstehen und Fehler in fremden Codezeilen zu finden. Open Source Entwickler halten sich deshalb gegenseitig dazu an, gut lesbaren Code zu schreiben.

2.2 Viele-Augen-Theorie

Die Sicherheit von Software korreliert vermutlich stark mit der Anzahl ihrer Fehler. Ihre Beseitigung steht daher an höchster Stelle um Sicherheitsmängel zu vermeiden.

Eric S. Raymond, der selber Open Source Software gemeinschaftlich entwickelt hat, schließt aus seinen Erfahrungen, dass je mehr Augen einen Quellcode begutachten, desto schneller Fehler gefunden und eliminiert werden können. "Given enough eyeballs, all bugs are shallow." (Raymond 2000)

Seine Annahme funktioniert natürlich nur, wenn die Personen, die den Quellcode anschauen, auch etwas von der Materie verstehen, allerdings ist anzunehmen, dass nur interessierte Personen genau dies tun. Bei gängigen Programmen, die von vielen Entwicklern aktiv weiterentwickelt werden, erscheint deshalb die Wahrscheinlichkeit tatsächlich recht hoch, ein Stück Software mit relativ wenigen Fehlern und Sicherheitslücken in Händen zu halten.

Die Offenheit von Quellcode ist somit keine Garantie für fehlerfreie Software, aber sie ist die entscheidende Grundlage, ohne die eine solche Entwicklergemeinde gar nicht erst möglich ist. Ob eine Open Source Software tatsächlich von ihren Fehlern befreit wird hängt also ganz deutlich von der Größe und Zusammensetzung seiner solchen Projektgemeinde ab. (Köhntopp/Pfitzmann 2000)

Um die Anzahl der verwendeten Augen nicht ganz vom Zufall abhängen zu lassen, bedient man sich des Peer Reviews und des Code Audits.

2.3 Peer Review

Mit Peer Review beschreibt man die gründliche Prüfung seitens einer oder mehrerer beaufsichtigender Personen, aber möglichst unabhängiger Fachleute. Im Sinne der Open Source Softwareentwicklung wäre dies also das Nachprüfen von neu geschriebenem Code durch mindestens den Maintainer eines Open Source Projekts, im Bestfall aber durch Software-Entwickler auf der ganzen Welt. Der Entwicklungsansatz der Open Source Communitys erscheint das am besten geeignete Peer-Review-Verfahren zu einer adäquaten Strategie zu sein. (Gehring 2001)

Der Linux-Kernel von Linus Torvalds ist ein bedeutendes Stück Open Source Software und als Betriebssystem durchaus sicherheitsrelevant. Er wird nach dem Verfahren des Peer-Review entwickelt. Der Kernel ist dabei in Subsysteme aufgeteilt, denen jeweils ein Maintainer zugewiesen ist. Die Maintainer erhalten Code und Patches von den teilnehmenden Entwicklern und begutachten die

Beiträge. Nur von ihnen gebilligter Code wird an den Maintainer für den Kernel weitergeleitet, der sich den Code ebenfalls anschaut bevor er ihn dann in das offizielle Repository aufnimmt. (OSDL 2003)

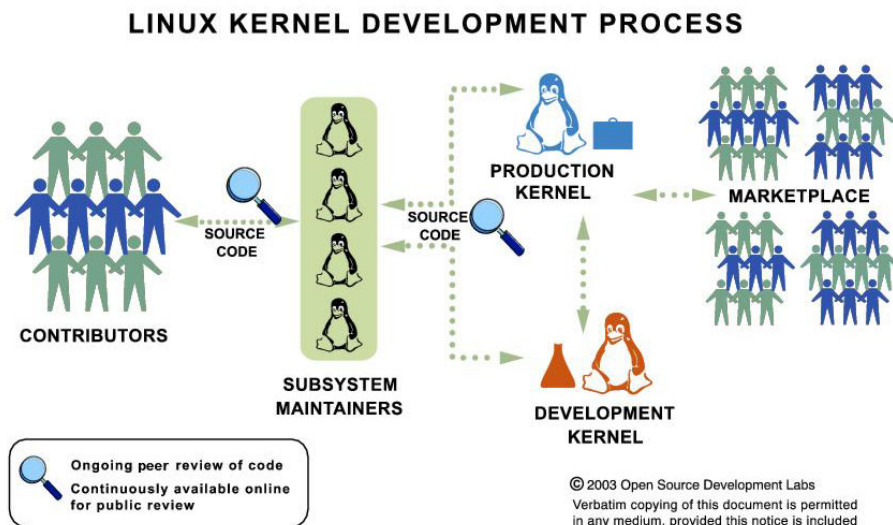


Abbildung 2: Struktur des Linux-Kerneldevelopment (ODSL 2003)

2.4 Code Audit

Der so genannte Code Audit ist ein Ansatz, Software nachträglich abzusichern. Programmierer überprüfen aktiv bereits existierenden Programmcode ein- oder mehrmals auf mögliche Fehler und Sicherheitslücken. Das OpenBSD-Projekt⁵ hat nach dieser Methode sein BSD-Derivat sehr erfolgreich abgesichert und kann sich rühmen, innerhalb von sieben Jahren nur eine einzige Remote-Sicherheitslücke in der Standardinstallation gehabt zu haben.

Der Ansatz des Code Audit ist also sehr erfolgversprechend, jedoch auch sehr zeit- und kostenintensiv und somit sehr unattraktiv für Hersteller von Closed Source Software. Die hohen Kosten dieser zusätzlichen Kontrolle können in sicherheitsrelevanten Bereichen aber durchaus berechtigt sein. Es mag Firmen oder Behörden geben, die aus Angst vor Hintertüren die von ihnen benötigte Software komplett neu schreiben lassen, da sie den bereits existierenden Open Source Programmen und erst recht vorkompilierten Programmpaketen kein Vertrauen schenken. (Robichaux, zitiert bei Lawton 2002)

⁵<http://www.openbsd.org>

2.5 die Binärpaket-Problematik

Open Source Software nicht stets selber zu kompilieren ist sehr gebräuchlich und hat entscheidende Vorteile. Die Übertragungszeiten bei Downloads sind wesentlich kürzer, ebenso verringern sich Aufwand und Wartezeit bei der Installation. Allerdings birgt diese Bequemlichkeit auch eine Gefahr für die Sicherheit. Man muss dem Distributor bzw. dem Ersteller des Binärprogramms zusätzliches Vertrauen entgegenbringen. Wenn man Binärpakete benutzt geht die Möglichkeit verloren, lückenlos nachvollziehen zu können, ob das binäre Programm tatsächlich aus dem dazugehörigen veröffentlichten Sourcecode entstanden ist.

Aus einer paranoiden Betrachtungsweise heraus kann ein Programm erst dann als "sicher" gelten wenn man seinen Quellcode überprüft und es eigenhändig mit einem Compiler kompiliert, dessen Quellcode man zuvor auf die gleiche Weise überprüft und kompiliert hat.

Ein Blick zum Closed Source Sektor macht allerdings deutlich, dass das Problem für Open Source Software rein praktischer Natur und eher gering einzustufen ist. Es lässt sich dort relativ simpel umgehen, indem man nur Software verwendet, die man auch selbst kompiliert hat. Mit Closed Source Software ist man auf Gedeih und Verderb der Ehrlichkeit des Herstellers ausgeliefert.

2.6 Softwareentwicklung in der Closed Source Company

Herstellerfirmen von Closed Source Software sind oft durch klare Hierarchien strukturiert, Management und Marketing bestimmen die Vorgehensweise. Es gibt einen festgelegten Ablauf in der Produktentwicklung, ein Budget und einen Zeitplan.

Man beginnt mit der Marktforschung, die den Bedarf einer bestimmten Zielgruppe ermitteln soll und erstellt dann eine Anforderungsanalyse, mit deren Umsetzung kleine Entwicklerteams betraut werden. Am Ende der Erstellung steht eine mehr oder minder lange Testphase mit hauseigenen und/oder externen Testern.

Kommerzielle Hersteller produzieren Software, um damit auf einem Markt Gewinne zu erzielen. Dabei kann die Funktionalität und Fehlerfreiheit von Software ein gewisser Wettbewerbsvorteil sein, jedoch müssen diese Aspekte gegebenenfalls hinter ökonomischen Interessen zurückstehen. Im Zweifelsfall wird die Erhöhung der Anzahl der neuen Features, die auf der Verkaufspackung angegeben werden kann, Vorrang haben vor dem Bereinigen aller Fehler, die mit der Einführung der neuen Features hinzugekommen sind. Da Closed Source Produkte vom Kunden praktisch nie vorher ausprobiert werden dürfen – sofern es nicht gerade um Shareware handelt – suggerieren zusätzlich ihre hohen Preise bei Entscheidern die erhoffte Sicherheit.

Und so ist der Benutzer von Closed Source Software immer vollständig abhängig von der Firma, die sie produziert hat. Er muss dem Hersteller sein Vertrauen schenken und kann nur drauf hoffen, dass alle sicherheitskritischen Fehler beseitigt wurden (oder noch werden, falls man sie erst später entdeckt) und dass der Hersteller sich oder dritten keine Hintertür eingebaut hat. Neue Features landen grundsätzlich in neuen Versionen, die natürlich erneute Kosten für den Kunden bedeuten.

Um einen Nachteil von Closed Source Software auszugleichen, insbesondere den, nicht überprüfen zu können was eine Software tut, gibt es den so genannten

Closed Structured Development Method

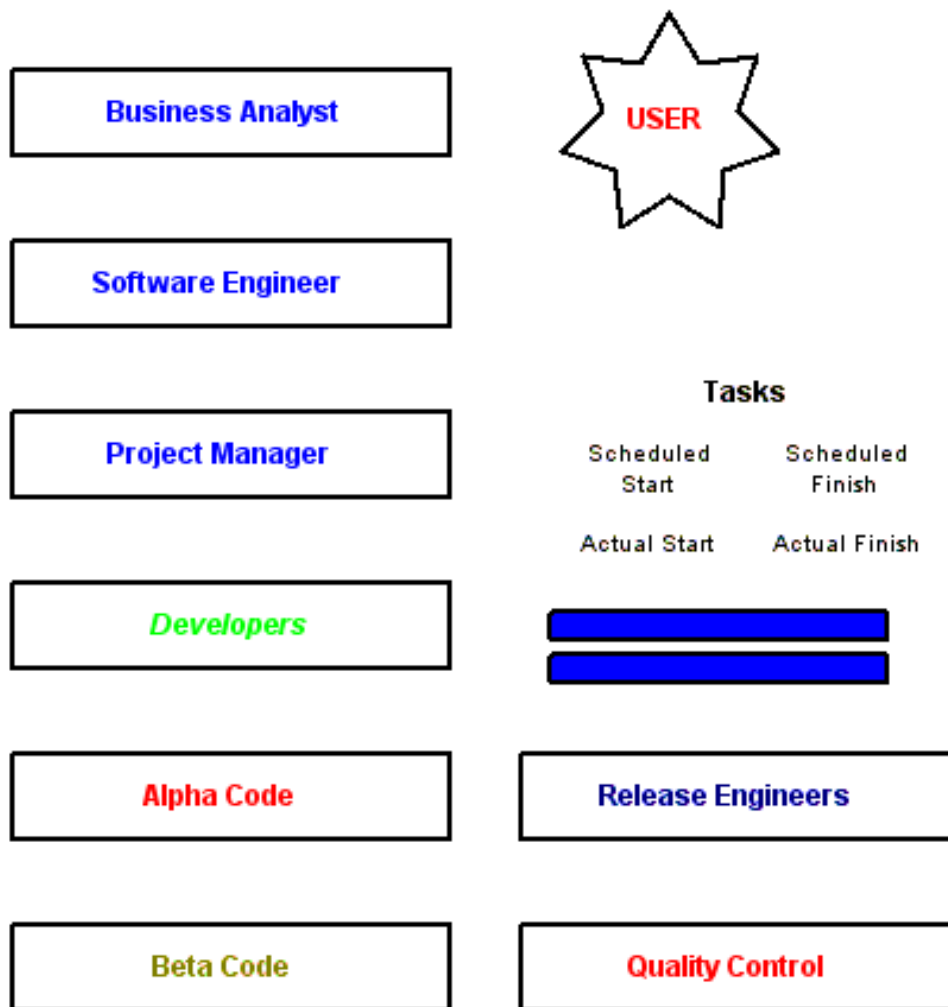


Abbildung 3: Entwicklung in der Closed Source Company (Adelstein 2003)

Shared Source. Firmen wie z.B. Microsoft erlauben dabei wenigen handverlesenen Firmen und Regierungen, Einblick in Teile ihres Windows-Quellcodes zu nehmen. Dadurch wird allerdings nur diesem einem Nachteil entgegengewirkt und das auch nicht vollständig, denn eine Garantie, dass das entsprechende Closed Source Programm tatsächlich den vorgelegten Code enthält, gibt es für die Einblick nehmenden Parteien nicht.

2.7 Mängel aus ökonomischen Gründen

Closed Source wird oft von Grund auf neu geschrieben, da normalerweise kein großer Fundus wie bei Open Source besteht aus dem man sich bedienen kann. Infolgedessen kann das gesamte Projekt potentiell mit "Kinderkrankheiten" behaftet sein. Oft genug muss der Endnutzer ausbaden was durch eine zu kurze Testphase nicht bereinigt wurde.

Kommerzielle Firmen haben nur begrenzte Mittel für Tests und Fehlerbeseitigung zur Verfügung, da ihr Produkt nicht beliebig teuer werden kann. Hinzu kommt, dass Tests nicht immer unter realistischen Bedingungen stattfinden kön-

nen und mitunter zu zeitaufwendig sind. (Gehring 2001)

Jeder gefundene Fehler erzeugt bei der Beseitigung allerdings zusätzliche Kosten. Es ist ökonomisch nachvollziehbar wenn der Hersteller sich gegen die Offenlegung der Sourcen gegenüber seinen Testern entscheidet, weil sie mit Sourcen in der Hand mehr Fehler finden. Ebenso ziehen Hersteller in Erwägung, bereits bekannte Sicherheitslücken nicht zu schließen, wenn die dabei eingesparten Kosten den eigenen potentiellen Schaden einer nachträglichen Beseitigung aufwiegen. (Anderson 2002) Vor allem aus diesem Grund reagieren Hersteller von Closed Source oft erst dann mit der Beseitigung von Sicherheitslücken, wenn ihnen große Image- und Umsatzverluste drohen.

2.8 Produkthaftung

Produkthaftung ist ein gern verwendetes Argument von Softwarefirmen, die Closed Source befürworten. Sie geben an, dass es bei Open Source keinen Hersteller gibt, der für seine Produkte gerade zu stehen hat. Das ist jedoch nicht korrekt, auch kommerzielle Linux-Distributoren wie z.B. SuSE oder Redhat unterliegen einer Produkthaftung.

Entscheidend für die Nichtigkeit des Arguments ist jedoch, dass Softwarehersteller generell nicht für die Qualität und Fehlerfreiheit ihrer Produkte haften. Sie lizenzieren ihre Software praktisch immer unter der Bedingung von dieser Haftung ausgeschlossen zu sein. Um dies zu verdeutlichen sei hier auszugsweise aus der Macromedia Endbenutzer-Lizenzvereinbarung zitiert:

„DIE FIRMA MACROMEDIA UND IHRE ZULIEFERER SCHLIESSEN JEGLICHE GEWÄHRLEISTUNG UND ZUSICHERUNG [...] AUS, EINSCHLIESSLICH JEGLICHER HAFTUNG FÜR DIE ZUSICHERUNG MARKTÜBLICHER QUALITÄT [...] MACROMEDIA GARANTIERT NICHT, DASS DIE SOFTWARE FREI VON FEHLERN IST ODER UNTERBRECHUNGSFREI FUNKTIONIERT.“
(Macromedia EULA)

Allein um die gesetzliche Produkthaftung kommen sie in den betreffenden Ländern, z.B. auch Deutschland, nicht herum, jedoch erscheint es ein unmögliches Unterfangen zu sein, diese im Fall von fehlerhafter Programmierung geltend machen zu können. Somit bleibt vom Argument der Produkthaftung nicht mehr als eine Gewährleistung bei Materialfehlern übrig.

2.9 Vergleich der Entwicklungsmodelle

Der Kern jeder Softwareentwicklung besteht aus einem Zyklus für Design, Review und Patch. In der Designphase entwirft man die Struktur des Programms und implementiert es in Sourcecode. Beim Review wird die Implementierung daraufhin untersucht, ob es dem erwünschtem Design entspricht. In der Patchphase werden gegebenenfalls Korrekturen und Änderungen vorgenommen, neue Funktionswünsche können in die nächste Designphase miteinfließen. (Köhntopp/Köhntopp/Pfitzmann 2000)

Diesen Kern hat die Entwicklung von Open Source mit der von Closed Source gemeinsam. Jedoch ist die Ausprägung beider Ansätze leicht verschoben.

Die Schmieden von Closed Source Software verwenden mehr Energie auf den Design-Teil. Er unterteilt sich bei ihnen genauer betrachtet in Planung, Analyse, Design und Implementation. (Feller und Fitzgerald, zitiert bei Iannacci 2003)

Open Source Entwicklung beginnt dagegen meistens mit der Implementierung und kann aufgeteilt werden in Kodierung, Review, Vortest, Veröffentlichung der Entwickler-Version, paralleles Testen und Veröffentlichung der stabilen Version. (Jørgensen, zitiert bei Iannacci 2003) Es handelt sich hierbei aber keinesfalls um abgeschlossene Phasen. Die Erkenntnisse aus Reviews und Vortests fließen ebenso wieder in die Kodierung ein wie die Patches, die resultierend aus dem Testen einer Entwicklerversion geschrieben werden. (Iannacci 2003)

Figure 1: The Linux Development Process

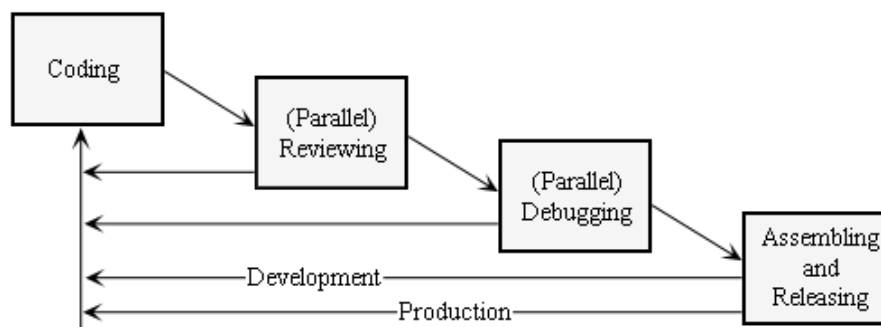


Abbildung 4: quasiparallele Arbeitsschritte der Open Source Entwicklungsmethode (Iannacci 2003)

Während die Arbeit der Softwareschmiede an einem Produkt mit der Fertigstellung der Implementation beendet wird – tatsächlich gibt es auch bei ihr Tests, jedoch nur in begrenztem Umfang – bleibt die Entwicklung in der Open Source Gemeinde ein fortwährender Prozess. Nicht nur Fehler werden kontinuierlich behoben, mit der Fertigstellung einer stabilen Version hat die Fortführung des Projekts mit einer neuen Entwicklerversion, die neue Features implementiert, meistens schon begonnen. Die Open Source Methode ist durch ihren andauernden Entwicklungsprozess und die breitere Basis an Testern schneller und effizienter.

Desweiteren unterscheiden sich auch die Motivationen unter denen Software entsteht. Auf Seiten des Closed Source steht das ökonomische Interesse einer kleinen, geschlossenen Gruppe, das allein nicht ausreicht um Software kontinuierlich von Fehlern zu befreien, weiterzuentwickeln und zu perfektionieren. Auf der Open Source Seite beschleunigt sich der Prozess durch die schnelle Reaktionsfähigkeit einzelner Leute und dem Willen und Bedarf vieler Interessierten zur fehlerfreien Funktionalität. Die Zahl der Helfer ist dabei nach oben offen.

„Closed source forces users to accept the level of security diligence that the vendor chooses to provide, whereas open source lets users

(or other collectives of people) raise the bar on security as high as they want to push it.“ (Cowan 2003)

3 Angriffe und Gegenmaßnahmen

Rechner in großen Netzwerken sind häufig ein Ziel von Angriffen mit den unterschiedlichsten Hintergründen. Das Betriebssystem und die darauf aufsetzende Software sorgen beide für sehr unterschiedliche Angriffspunkte und Konsequenzen bei erfolgreichen Attacken.

3.1 Security by Obscurity

Security by Obscurity lautet ein Prinzip der Sicherheit in Closed-Source-Software. Sie entsteht offensichtlich, indem man den Sourcecode nicht veröffentlicht und somit die im Programm möglicherweise vorhandenen, angreifbaren Schwachstellen verschleiert.

Tatsächlich ist es für einen Angreifer leichter, offenen Source zu verstehen und anzugreifen als ein binäres Programm, jedoch ist auch Software im Binär-code nicht vor Angriffen sicher. Durch Beobachtung von Ein- und Ausgabeverhalten, direkten Zugriff auf Speicherbereiche des laufenden Programms und Disassemblierung sind weitreichende Manipulationen möglich, die immer wieder neuentdeckte Angriffspunkte enthüllen können.

So kann diese Art der Sicherheit als eine erste Hürde für einen Angreifer angesehen werden, jedoch kann eine solche Software nicht als sicher angesehen werden, solange faktisch vorhandene Sicherheitslücken nicht geschlossen wurden.

Geschickt eingesetzt kann eine temporäre Verschleierung einen wertvollen Zeitvorteil bedeuten wenn eine noch nicht öffentlich bekannte Sicherheitslücke geschlossen werden soll, jedoch mag es aus den bereits genannten ökonomischen Gründen leider auch dazu führen, dass man die Fehlersuche und -beseitigung mit Vorsatz vernachlässigt.

In einer idealen Welt sind die Angriffsmöglichkeiten von Open und Closed Source Software symmetrisch verteilt. Gutartige Verbesserungen können an offenen Quellen ebenso leicht vorgenommen werden wie hinterhältige Angriffsversuche. Entsprechend dazu wäre Software mit verborgenem Source für gute und schlechte Intentionen gleichermaßen verborgen. Da unsere Welt allerdings nicht ideal ist, müssen wir uns von diesem Gedanken verabschieden. Closed Source Software lässt sich auf mannigfaltige Weisen manipulieren.

„We can expect attackers to search for, find and exploit phenomena that break the symmetry between open and closed models.“

(Anderson 2002)

Es entstehen somit Asymmetrien, die sich auf Art und Anzahl der Angriffe auswirken.

3.2 Angriffe im Closed Source Bereich

Im Bereich Closed Source sind die so genannten Würmer ein gravierendes Problem. Einmal in Umlauf gebracht verschicken sich diese virenartigen Skript - Programme (halb-) automatisch unter Ausnutzung einiger "Features" des bei Microsoft Windows mitgelieferten E-Mail-Programms Outlook durch das Internet.

Auf den "besuchten" Rechnern können sie schwere Datenverluste verursachen. Jedoch sind die Betroffenen auch meistens mitschuld an der Anfälligkeit von Software, da sie Fehler bei der Bedienung machen.

Laut einer Umfrage des Hamburger Forschungs- und Beratungsinstituts Media Transfer AG, welches 1.250 Personen, die Computer an den Arbeitsplätzen bereitstellen, befragte, zu dem Ergebnis kam, dass viele Unternehmen, obwohl sie enorme Schäden einzubüßen hatten, immer noch keine entsprechenden Sicherheitsvorkehrungen getroffen haben.

Dieses unterstreichen folgende Zahlen: 39,5 Prozent haben entsprechende Sicherheitsvorkehrungen in Form einer neuen Software getroffen. Fast 40 Prozent aller Unternehmen mit Computerarbeitsplätzen erlauben den Mitarbeitern nach wie vor, E-Mailanhänge "nach Lust und Laune" zu öffnen. Nur 46,7 Prozent aller befragten erlauben das Öffnen von Dateianhängen erst nach einem entsprechenden Virencheck mit neuester Anti-Viren Software. Zunächst Rücksprache mit der Technik halten müssen die Mitarbeiter in 7,7 Prozent aller befragten Firmen. 6,3 Prozent aller Firmen erlauben keinerlei Attachments mehr zu öffnen.

Ihre Sicherheitsvorkehrungen für ausreichend halten 60,5 Prozent aller befragten Unternehmen, bzw. man nehme auch für die Zukunft eine Viren- Attacke in Kauf. (MediaTransfer AG, zitiert bei Tietz 2000)

Attacken in Closed Source Systemen zeichnen sich dadurch aus, dass ihre zu Grunde liegenden Sicherheitslücken allein der Hersteller schließen kann und dies leider oft nicht oder erst sehr spät tut. Outlook-Würmer existieren in zahlreichen Varianten, die oft dieselbe Schwachstelle immer wieder ausnutzen.

Die Leidtragenden finden sich damit ab oder versuchen durch Umgehung der fehlerhaften Funktionen den nötigen Schutz zu erlangen. Dritte, also andere Closed Source Firmen, verdienen an den Sicherheitslücken, indem sie Software verkaufen, die die böartigen Schädlinge durch Observierung des Systems entdecken und dann zu beseitigen versuchen, dies ist meistens eine Behandlung der Symptome.

3.3 Angriffe im Open Source Bereich

Die Server des Debian-Projekts waren Ende November 2003 Ziel eines Angriffs. Es handelt sich um einen Einzelfall, kein Massenphänomen wie bei Outlook-Würmern, aber er illustriert sehr gut die üblichen Methoden bei Angriffen auf Open Source Software, in diesem Fall Linux-Rechner.

Am 19. November benutzte ein unbekannter Angreifer ein abgehorchtes Passwort, um Zugang zu einem der Debian-Rechner zu erhalten. Dann verwendete er einen Exploit für den Linux-Kernel, der eine bereits bekannte Sicherheitslücke ausnutzte:

„Ein Integer-Überlauf im brk-Systemaufruf wurde ausgenutzt, um Kernelspeicher (»change page protection bits«) zu überschreiben. Dadurch erhielt der Angreifer komplette Kontrolle über den Kernelspeicher und so war es ihm möglich, jeden Wert im Speicher zu ändern.“

So bekam er Root-Rechte und die Kontrolle auf den gesamten Rechner. Von dort aus erlangte er nach und nach Zugriff auf insgesamt vier Rechner. Noch am selben Tag gab es Ungereimtheiten in den Log-Einträgen der betroffenen Rechner, tags darauf fanden die Verantwortlichen die endgültige Bestätigung, dass auf den Rechnern eingebrochen wurde.

Bereits am 21. November gab es eine öffentliche Ankündigung über den Vorfall und die nötigen Maßnahmen, die ergriffen werden mussten. Die Rechner wurden vom Netz genommen, Abzüge der Festplatten zur forensischen Analyse gegeben und die betroffenen Rechner an den nachfolgenden Tagen für den Serverdienst wiederhergestellt. Die Analyse ergab unter anderem, dass der Angreifer mittels eines sogenannten Root-Kits seine Spuren verschleierte und Hintertüren installierte:

„[Das SucKIT Root-Kit] enthält einen Passwortschnüffler und Versteckfähigkeiten (d.h. Werkzeuge, um Prozesse und Dateien zu verbergen), die direkt in den Kernel installiert werden [...] Desweiteren haben [drei der betroffenen Rechner] »Advanced Intrusion Detection Environment« (Paket aide) installiert, um Dateisystemänderungen zu überwachen.“

Der geschilderte Vorfall beruht vor allem auf Nachlässigkeiten und Informationsdefiziten an denen zu arbeiten man sich nach dem Angriff vorgenommen hat. Die Sicherheitslücke war bei der ursprünglichen Entdeckung als nicht schwerwiegend angesehen worden. Die Linux-Entwickler hatten sie zwar für zukünftige Versionen behoben, jedoch hatten weder sie noch die gängigen Linux-Distributoren vor ihr gewarnt. Bereits wenige Tage nach Bekanntwerden des Einbruchs auf den Debian-Servern wurden Patches bzw. eine neue Linux-Version zur Verfügung gestellt. (Debian-Untersuchungsbericht 2003)

Schwachstellen von Open Source Systemen sind leicht auszukundschaften, da der Source zur Verfügung steht. Eine Sicherheitslücke ist jedoch in der Regel schnell geschlossen wenn sie einmal entdeckt wurde. Da sehr viele Sicherheitsprobleme bereits bei der Entwicklung lokalisiert und gelöst werden, müssen Angreifer trotz der offenen Sourcen teilweise lange suchen bis sie eine bisher unentdeckte Schwachstelle finden und sehr trickreich vorgehen wenn sie sie erfolgreich einsetzen wollen.

Intrusion Detection Systeme wie aide⁶ oder snort⁷ leisten gute Dienste bei der Erkennung von Rechner-Kompromittierungen. Sie speichern unter anderem Prüfsummen von den installierten Programmen ab und vergleichen diese Regelmässig mit dem aktuellen Zustand. Dem Abhören von Passwörtern ließe sich durch konsequentere Verschlüsselung entgegenwirken. Mehr zu Open Source Tools und Verschlüsselung im nächsten Abschnitt.

⁶<http://sourceforge.net/projects/aide>

⁷<http://www.snort.org>

3.4 Open Source Security Tools

Nachdem wir jetzt verschiedene Attacken auf beide Systeme kennen gelernt haben, wollen wir uns nun mit einigen Open Source Tools beschäftigen, die solche Angriffe verhindern sollen.

Zu den wichtigsten Anwendungsgebieten von Open Source Security Tools gehört die Verschlüsselung zum kontrollierten Umgang von Geschäftsgeheimnissen und persönlichen Daten.

3.4.1 Kerberos

Da wäre als erstes Kerberos. Kerberos bietet sichere und einheitliche Authentisierung in einem ungesicherten TCP/IP Netzwerk aus sicheren Hostrechnern. Die Authentisierung übernimmt eine "vertrauenswürdige dritte Partei".

Diese dritte Partei ist ein besonders geschützter Kerberos 5 Netzwerkdienst. Durch Kerberos werden insbesondere Angriffe durch passives "Sniffing" unterbunden, aber auch "Spoofing", "Dictionary Attacks", "Replay" und andere Angriffe werden erschwert.

„In der Regel gibt es eine physikalisch geschützte Maschine, auf der ein Kerberos Dienst (und nichts anderes) läuft, auf den Klienten und Server zur Authentifizierung zurückgreifen. In einem ersten Schritt fragt ein "Principal" (z.B. der Benutzer an einer Workstation) den Kerberos Dienst nach einem Ticket für einen so genannten "Ticket Granting Service" (TGS). Dieses Ticket wird mit dem geheimen Schlüssel des Principals verschlüsselt. Da der geheime Schlüssel nicht auf der Workstation des Principals gespeichert ist, muss er den geheimen Schlüssel eingeben, um das Ticket zu verwenden.

Dies kann per Tastatureingabe, oder z.B. mit einer Smartcard geschehen. Zu keiner Zeit wird dabei ein Passwort im Klartext über das Netz gesandt. Wenn der Benutzer nun die Autorisierung erhalten möchte, einen Netzwerkdienst (z.B. "telnet") zu verwenden, bittet der Telnet-Klient unter Vorlage des ersten Tickets beim Kerberos Server um ein Ticket für den Telnet-Dienst. Dieses Ticket wird dann zusammen mit einem Authenticator dem Server vorgelegt. Mit diesem Ticket darf sich der Benutzer dann am entfernten Hostrechner anmelden. Kerberos Tickets haben eine begrenzte Lebensdauer, nach deren Ablauf sie nicht mehr gültig sind, außerdem sind sie an die IP-Adresse eines bestimmten Rechners gebunden.

Dies dient dem Zweck, dass Tickets, die von einer unbefugten Instanz abgehört wurden, nicht mehr nach Ablauf ihrer Lebensdauer und auch nicht auf einem anderen Host missbräuchlich verwendet werden können.

Da es vergleichsweise einfach möglich ist, eine IP-Adresse in einem IP-Paket zu fälschen und auch IP-Pakete abzufangen bietet dies jedoch nur eine leichte Erschwernis für einen hypothetischen Angreifer. Auch die begrenzte Lebensdauer von Tickets ist eine vergleichs-

weise leicht zu umgehende Einschränkung.

Die eigentlichen Stärken von Kerberos liegen in den kryptografischen Methoden. Praktisch jeder Netzwerkdienst lässt sich an Kerberos anpassen. Bei der Kerberos Distribution selbst werden schon angepasste Varianten von telnet, den bekannten r-Kommandos und ftp mitgeliefert, die zusätzlich über die Möglichkeit zur Verschlüsselung des Datenstroms verfügen.

Angepasste Varianten vieler verbreiteter Netzwerkdienste, wie z.B. "Secure Shell" und cvs sind verfügbar. Außerdem gibt es auch Netzwerkdateisysteme, die Kerberos zur Authentifizierung verwenden (z.B. AFS und DFS). Wird in einem Netzwerk Kerberos zur Authentifizierung verwendet, ist es sinnvoll, Kerberos mit Hilfe von PAM oder einem an Kerberos angepassten login-Programm gleich in die lokale Anmeldung zu integrieren. Zum einen lässt sich darüber eine Vereinheitlichung der Passwörter und der Benutzerverwaltung für alle Hostrechner erreichen, zum anderen ist es für den Benutzer bequem, ohne Angabe eines weiteren Passworts auf alle für ihn freigegebenen Netzwerkressourcen zugreifen zu können ("Single Sign-On")."

(DFN-CERT 2003)

3.4.2 OpenSSL

Dann gibt es noch OpenSSL:

„Das OpenSSL Projekt hat ein gemeinschaftliches Ziel, eine robuste, einem kommerziellen Produkt in nichts nachstehende vollständige Secure Socket Layer (SSL v2/v3) und Transport Layer Security (TLS v1) Implementierungen auf Open Source Basis zu entwickeln. Dazu gehört ebenfalls eine universelle Kryptografische Bibliothek. Das Projekt wird durch eine weltweite Gemeinschaft von Volutären unterhalten welches das Internet für Kommunikation, Planung und Entwicklung benutzt. OpenSSL basiert auf der SSLeay Bibliothek von Eric A. Young und Tim J. Hudson. OpenSSL hat eine Apache -ähnliche Lizenz, welche grundlegend besagt: Freie Benutzung für kommerzielle und nicht - kommerzielle Anwendungen.“ (OpenSSL)

3.4.3 Webserver

Jedoch gehören zu den verbreitetsten Web-Servern der Apache Webserver und der Microsoft Internet Information Server.

„Der Microsoft Internet Information Server 4.0 (IIS 4.0) ist Bestandteil des Windows NT Server 4.0 Option Packs, das eine Reihe von zusätzlichen Dienstprogrammen und Anwendungen enthält, die mit dem IIS 4.0 zusammenarbeiten. Als typische Microsoft-Anwendung ist der IIS 4.0 eng mit den Ressourcen des Betriebssystems verzahnt. Der IIS 4.0 verwendet die Verzeichnisdatenbank von Windows NT, d.h. die Benutzerkonten werden mit Hilfe des Windows NT- Benutzermanager verwaltet. Außerdem werden vorhandene Windows NT-

Dienstprogramme, wie der Systemmonitor, die Ereignisanzeige und die SNMP-Unterstützung zur Verwaltung des Web-Servers genutzt.“ (BSI Sicherheitsstudie 2002)

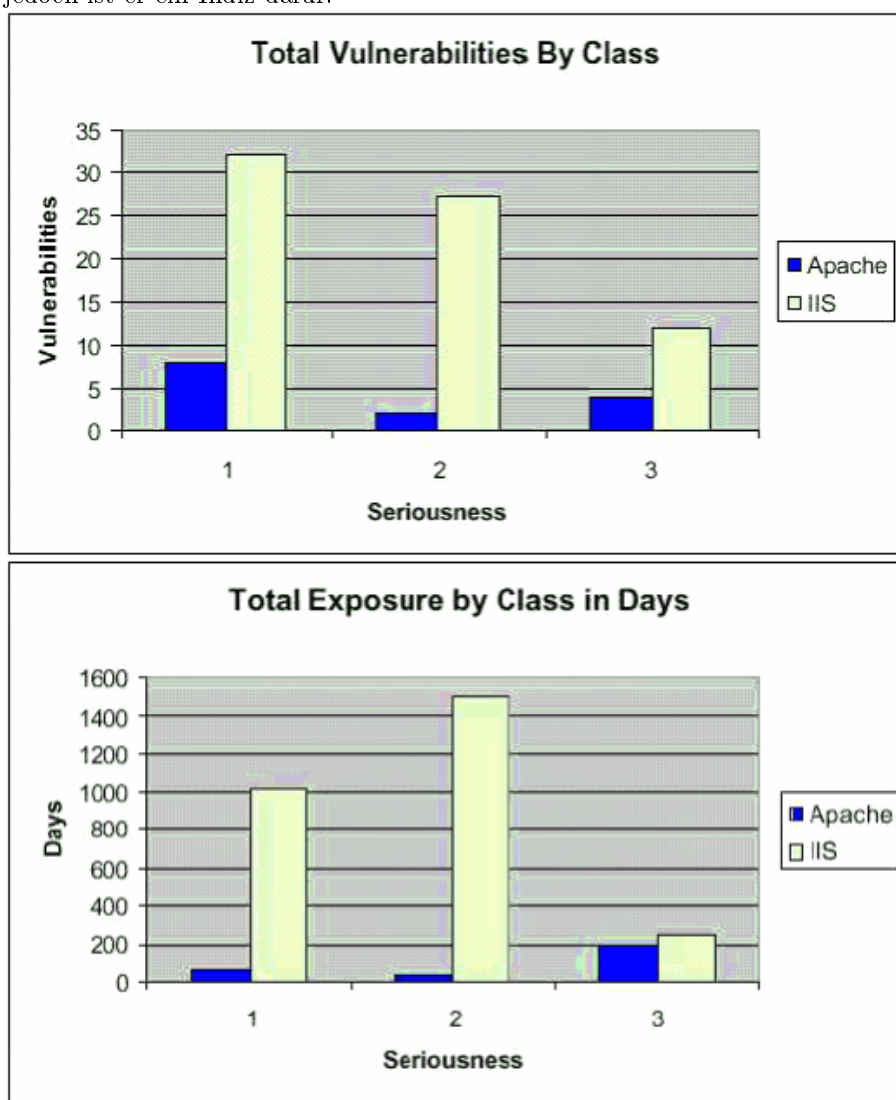
„Apache ist ein HTTP -Server, der sich u.a. durch seine freie Verfügbarkeit für viele Betriebssysteme auszeichnet. Für den Apache existieren zwei Programme, die jeweils den Server um SSL-/TLS -Funktionalität ergänzen.

Durch diese Ergänzung ist es möglich, verschlüsselte und authentifizierte Verbindungen zwischen dem HTTP -Server und einem HTTP-Client zu realisieren. Beide Programme setzen ein betriebsbereites SSL-/TLS-Programmpaket OpenSSL.“ (DFN-CERT 2002)

3.5 Empirischer Vergleich: Apache vs. IIS

Nun schauen wir uns noch einen Vergleich der beiden bekanntesten Open bzw. Closed Source Webserver an. Gemeint sind der Apache Webserver und der Microsoft Internet Information Server (IIS). Betrachtet man die Anzahl der Sicherheitslücken und die Länge der ermittelten Gefahrenaussetzung, geht der Apache Webserver als deutlicher Sieger hervor. Er hat nicht nur weniger Fehler und bedeutend weniger sicherheitskritische Fehler, entdeckte Fehler werden auch schneller beseitigt, davon sicherheitskritische am schnellsten.

Natürlich ist dieser Vergleich nur ein Einzelbeispiel und kein allgemeingültiger Beleg für die sicherheitstechnische Überlegenheit von Open Source Software, jedoch ist er ein Indiz dafür.



Fehlerklassen: 1 = hohes, 2 = mittleres, 3 = geringes Risiko

Abbildung 5: empirischer Vergleich (Ritchey, zitiert bei Gehring 2003)

4 Open Source in der Politik

4.1 e-Government mit Open Source Software

Open Source Software gewinnt mehr Anerkennung in der öffentlichen Verwaltung. Die Stadt München ist Vorläufer für die Einführung des alternativen Betriebssystems GNU/Linux in der öffentlichen Verwaltung in Deutschland. Das Bundesministerium des Innern hat in Zusammenarbeit mit IBM Deutschland einen Migrationsleitfaden herausgebracht, der die öffentlichen Verwaltungen des Bundes, der Länder und Gemeinden dabei unterstützen soll, Behörden von der "Monokultur" Microsoft auf das Betriebssystem Linux umzustellen. Bereits über 500 Behörden aus Bund, Ländern und Gemeinden hätten einen Antrag gestellt, diesem Vorhaben beizutreten.

Der Bundesinnenminister Otto Schily erhofft sich dadurch eine enorme Entlastung des Haushalts, weil teure Lizenz-Abkommen mit dem bisherigen Monopolisten Microsoft der Vergangenheit angehören würden. Abgesehen davon sei ein weiteres Ziel dabei "Monopolstrukturen aufzubrechen" und Wettbewerb in die Betriebssystem-Landschaft zu bringen. (Fuchs 2003)

„Dem der Biologie entlehnten Prinzip der Vermeidung von Monokulturen folgend gilt es, im Sinne der IT-Sicherheit auch die Vielfalt der Informationstechnik zu fördern.“

(Helmbrecht, Wirtschaftsinformatik 2003)

Da das ein gewagtes Vorhaben ist, Open Source Software in den öffentlichen Verwaltungen einzusetzen, mussten vertrauensvolle Instanzen her, die eine offizielle Sicherheitsprüfung der quelloffenen Software durchführen.

So eine Sicherheitsprüfung wurde in einer Zusammenarbeit zwischen der IBM Corporation, atsec Information Security GmbH⁸ und der SuSE Linux AG durchgeführt. Hierbei wurde der Linux Enterprise Server 8, eingesetzt auf IBM xSeries Servern, nach den international anerkannten Common Criteria (ISO 15408) evaluiert. Die Evaluation konnte in erstaunlich kurzer Zeit erfolgreich abschließen. Durch die Evaluierung wird bestätigt, ob das geprüfte Produkt tatsächlich über angemessene Sicherheitseigenschaften verfügt.

Dieser Erfolg veranlasste das Bundesamt für Sicherheit in der Informationstechnik (BSI) das weltweit erste Zertifikat für ein Open-Source-Betriebssystem auszustellen.

4.1.1 Common Criteria

Die Common Criteria sind eine Weiterentwicklung und Harmonisierung der europäischen ITSEC, des Orange Books (TCSEC), der Federal Criteria (FC) der USA sowie der kanadischen Kriterien (CTCPEC). Sie wurden im Dezember 1999 durch die internationale Standardisierungsorganisation (ISO) als internationale Norm ISO/IEC 15408 veröffentlicht. Die Common Criteria sind damit das aktuelle Kriterienwerk und werden vom BSI als Evaluierungsgrundlage empfohlen.

⁸einem beim Bundesamt für Sicherheit in der Informationstechnik (BSI) akkreditiertem Prüflabor

Die Anforderungen der Sicherheitskriterien gliedern sich einerseits in Anforderungen an die *“Bewertung der Korrektheit“* des zu evaluierenden Produkts oder Systems und andererseits in Anforderungen an die *“Bewertung der Wirksamkeit“* von dessen Sicherheitsfunktionen.

Bewertung der Korrektheit

- Anforderungen an den Umfang und den Detaillierungsgrad der Design-Dokumentation
- Anforderungen an die Strukturierung des Designs
- Anforderungen an den Umfang und die Tiefe der funktionalen Tests
- Anforderungen an die Entwicklungswerkzeuge
- Anforderungen an die Sicherheitsmaßnahmen in der Entwicklungsumgebung
- Anforderungen an die Verfahren der Produktauslieferung zum Anwender

Bewertung der Wirksamkeit

- Bei der *Bewertung der Wirksamkeit* der Sicherheitsfunktionen wird durch Analysen und Penetrationstests untersucht, ob es einem Angreifer möglich ist, die Sicherheitsfunktionen zu überwinden oder Schwachstellen auszunutzen. (BSI Informationen zu Fachthemen)

“Die große Resonanz auf die Erteilung des Zertifikats zeigt die steigende Bedeutung der Zertifizierung für den Bereich der IT-Sicherheit“
(Dr.Helmbrecht zitiert bei BSI Pressemitteilung 2003)

4.1.2 Webservices für e-Government als Open Source

Eines von vielen Projekten, die eine Open-Source Lösung für die öffentliche Verwaltung bieten soll, ist das *“Projekt Leopard“*. *Leopard* ist ein Projekt des Open Source Software Institute (OSSSI), das sich für den Einsatz freier Software im öffentlichen Verwaltungsbereich einsetzt.



Abbildung 6: Projekt Leopard (OSSSI)

Diese Open Source Lösung stellt Behörden einen Webservice auf der Grundlage von LAMP⁹ bereit. Behörden sollen damit standardisierte Webanwendungen erstellen können, um sowohl die Kommunikation der Behörden untereinander als auch die Verständigung mit den Bürgern zu verbessern.

⁹Linux, Apache, MYSQL, PHP/Perl/Python

4.2 e-Voting mit Open Source Software

Ein elektronisches Wahlsystem muss bestimmte Sicherheitsanforderungen erfüllen. *Testbarkeit durch Öffentlichkeit* ist ein Punkt in den Sicherheitsanforderungen, der mehr Vertrauen bei den Wählern schaffen würde.

Elektronische Wahlsysteme, deren Quellcode nicht öffentlicher Kontrolle oder Kritik ausgesetzt wird, legen den Verdacht eines ungenügenden Sicherheitsniveaus nahe. Ein Wahlsystem besitzt die Eigenschaft der Testbarkeit durch die Öffentlichkeit, wenn jede Software- und Hardwarekomponente als *Open Source* deklariert wird.

4.2.1 Sicherheitsanforderungen eines “Electronic Voting Systems“

- *Authentifikation* – Noch vor der Abgabe des gültigen Wahlscheins muss die Identität des konkreten Teilnehmers überprüft werden.
- *Autorisierung* – Die Wahl-Server müssen gegen unberechtigten Zugriff auf ihre Ressourcen geschützt werden. Um unautorisierten Zugriff auf Daten, Programme sowie auf andere Ressourcen zu vermeiden, muß anhand der Autorisierung ermittelt werden, wer auf eine Ressource wie zugreifen kann.
- *Integrität* – Unter Datenintegrität wird verstanden, daß alle im Wahlprotokoll transferierten und gespeicherten Daten nicht manipuliert werden können.
- *Korrektheit* – Führen alle Systemprozesse die Spezifikation des Wahlprotokolls ordnungsgemäß aus, dann werden alle gültigen Stimmen korrekt ausgezählt.
- *Verifizierbarkeit* – Das Wahlergebnis kann nicht gefälscht werden, wenn es überprüfbar ist.
- *Vertraulichkeit* – Die Forderung nach Vertraulichkeit verhindert die Preisgabe wahlbezogener Informationen gegenüber Dritten.
- *Nichtvermehrbarkeit* – Eine unzulässige Wahlscheinvermehrung muss unbedingt entdeckt werden können.
- *Nichtbeeinflussbarkeit* – Vor dem offiziellen Ende der Wahlzeit darf keine Administration über die technische Fähigkeit verfügen, Zwischenresultate zu ermitteln und zu publizieren.
- *Wahlgeheimnis* – Das Wahlgeheimnis, der Rechtsgrundsatz der geheimen Stimmabgabe, gewährleistet die Entschließungsfreiheit des Wählers.
- *Unmittelbarkeit* – Das Kriterium der Unmittelbarkeit umfaßt die Unverkaufbarkeit und Unerzwingbarkeit der Stimmabgabe.
- *Testbarkeit* – Die Testbarkeit des Wahlprotokolls beschreibt Maßnahmen zum Finden von Fehlern. Die Testbarkeit des Quellcodes ermöglicht erhöhte Sicherheit vor unzulässigen Modifikationen. Testbarkeit durch die Öffentlichkeit schafft mehr Vertrauen bei Wählern und trägt zu mehr Transparenz in der Demokratie bei.

- *Verfügbarkeit* – Der Netzzugang und die Wahl-Server muss für den Wähler während der Wahlen verfügbar sein.
- *Zuverlässigkeit* – Eine unzuverlässige Infrastruktur darf keinesfalls die Stimmabgabe behindern und niemals in einem inkonsistenten bzw. unbekanntem Zustand gelangen.
- *Wartbarkeit* – Die Wartung eines E-Wahlsystems umfaßt alle Tätigkeiten, die nach Abschluß der Entwicklungsarbeiten vorgenommen werden müssen. Sie wirken sich auf die Sicherheit aus.

(Schlifni 2001)

4.2.2 Die Praxis

Debian GNU/Linux diente in den Oktober-Wahlen 2001 im ACT¹⁰ als Plattform für ein elektronisches Abstimmungssystem. Das Consulting-Unternehmen *Software Improvements* erhielt den Zuschlag, um dieses OSS e-Voting System zu erstellen. Hier ein Zitat von Carol Boughton von *Software Improvements*:

„Online-Wahlen sind im höchsten Maße kritisch, nicht in dem Sinn, dass das Leben einer Person davon abhängt, doch müssen sie präzise, zuverlässig und an dem Tag verfügbar sein [...] Der wichtigste Grund, weswegen wir diesen Weg gingen, war die Transparenz und die Möglichkeit, den Wählern versichern zu können, dass alles mit rechten Dingen zugeht. Wenn man den Code einmal veröffentlicht hat, hat jeder die Möglichkeit, dieses zu überprüfen.“

(Boughton, zitiert bei Debian 2001)



Abbildung 7: Terminal bei den Wahlen im ACT

Im grossen und ganzen verliefen die Wahlen mit dem neuem System ohne Probleme ab. Den Sourcecode, welchen man für dieses elektronische Wahlsystem angefertigt hatte, kann man sich auf der offiziellen Seite¹¹ dieser elektronischen Wahl in ACT herunterladen.

¹⁰Australian Capital Territory, 2400 qm großes Gebiet um die Hauptstadt Canberra

¹¹<http://www.elections.act.gov.au/Elecvote.html>

5 Fazit

In der Softwareentwicklung ermöglicht Open Source sicherere Software als Closed Source. Der offene Quellcode ist dabei keine Garantie für sichere Software, aber die Grundvoraussetzung, die den Kreis der beteiligten Personen nicht aus ökonomischen Motiven begrenzt.

Open Source sorgt großflächig für Sicherheit. Kleine Firmen und gemeinnützige Organisationen, die sich keine oder nur eine teilweise Absicherung ihrer IT-Infrastruktur leisten könnten, erhalten mit Open Source einen sehr kostengünstigen Schutz für alle ihre Rechner. (Schlesinger 2003) Dabei ist nicht zu vernachlässigen, dass jeder nicht abgesicherte Rechner mit Verbindung zum Internet eine zusätzliche Gefahr für andere Internet-Teilnehmer darstellt, da ein böswillig erobertes Rechner als Ausgangspunkt für weitere Angriffe dienen kann. Aktive Communitys zeichnen sich insbesondere durch die schnelle Behebung von Fehlern und Sicherheitslücken in ihren Programmen aus.

Sehr überzeugend klingt die Vereinigung der beiden Ansätze von Open und Closed Source zu einer Hybridlösung. Als Basis dienen ein oder mehrere Open Source Softwareprojekte, die die Transparenz und die vielen Augen einer Community gesehen haben. Darauf aufbauend kann ein Hersteller geeignete Hardware und dazugehörige Dienstleistungen wie Installation, Konfiguration und Wartung anbieten.

Gegebenenfalls schmiedet der Hersteller eine grafische Oberfläche mittels der der Kunde die Open Source Programme einfacher bedienen kann. Der Source einer solchen Oberfläche ist bei weitem übersichtlicher als die Gesamtheit der damit kontrollierten Software und könnte somit Closed Source bleiben, auch damit der Hersteller seinen Wettbewerbsvorteil wahren kann. (Lawton 2002)

Es ist ein trauriger Fakt, dass nicht alle Fehler behoben werden, die auch entdeckt werden.

„In Security rapid response is everything.“ (Schlesinger 2003)

Dieses bedeutet, dass wenn ein Fehler aufgetreten ist, man ihn schnell erkennen sollte um weiteren Schaden zu verhindern. Da Closed Source Betreiber ihren Quellcode nicht freigeben und dadurch die Fehlersuche einschränken, ermöglicht das den Open Source Befürwortern in einem positiven Licht zu stehen. Natürlich können die Closed Source Betreiber auf ihr Recht der Geheimhaltung plädieren, aber dieses könnte man nur akzeptieren wenn es sich um private Angelegenheiten handelt. Ein Server ist alles andere als Privat, im Gegenteil es ist ausdrücklich erwünscht, dass viele Menschen darüber kommunizieren oder arbeiten.

Natürlich gibt es keine fehlerfreien Systeme, man kann immer nur darauf bauen, alle Möglichkeit der Absicherung getroffen zu haben, und ansonsten hoffen, vor einem Angriff verschont zu bleiben.

Und welche Tools man benutzt um sich abzusichern bleibt jedem selbst überlassen. Wahrscheinlich wird man sich nach den oben aufgeführten Argumenten für Open Source Tools entscheiden. Jedoch sollte man unbedingt eines nicht vergessen:

So unterschiedlich die Angriffsstrategien und potentiellen Schäden beim offenen und geschlossenen Quellcode auch sind, keine von beiden Entwicklungsme-

thoden kann man pauschal als die sichere bezeichnen. Es kann letztendlich eine Frage der Überzeugung sein ob man Open oder Closed Source Software benutzt.

Verschiedene Vorgänge in demokratischen Gesellschaften basieren auf Vertrauen und das wiederum ist durch die Transparenz gewährleistet, die man in einem System wiederfindet.

Open Source kann tatsächlich zu mehr Transparenz in Politik und Gesellschaft beitragen. Die gesamte Bevölkerung profitiert von niedrigen Lizenzkosten und vom freien Wettbererb unter den IT-Dienstleistern.

Weltweit gibt es einen Trend, in Behörden und sicherheitsrelevanten Bereichen zunehmend auf *Open Source* zu setzen. Gründe dafür sind unter anderem das Misstrauen in Microsoft-Produkte, da diese gerne "*nach Hause telefonieren*".

Abgesehen davon spielt die Unabhängigkeit der Länder eine große Rolle. Gesetzt den Fall, daß Microsoft pleite gehen würde, dann würde es vielen genauso ergehen, die auf Microsoft-Produkte gesetzt haben. Um dies zu unterstreichen sei hier John "Maddog" Hall, Präsident von *Linux International*, zitiert:

„Man sollte seine Geschäfte niemals von der Gnade oder dem Geschick anderer Firmen abhängig machen.“

(Hall zitiert bei Bochers 2003)

Mit Open Source hat der Staat den kompletten Source code und alle Rechte. Ihn zu verändern und weiter zu entwickeln wäre ein Schritt zu mehr Souveränität und Sicherheit seiner Geschäfte.

Literatur

- [Adelstein] Adelstein, Tom (2003) How to Misunderstand Open Source Software Development <http://www.consultingtimes.com/ossdev.html>
- [Anderson] Anderson, Ross (2002) Security in Open versus Closed Systems <http://www.ftp.cl.cam.ac.uk/ftp/users/rja14/toulouse.pdf>
- [Bochers] Borchers, Detlef (17.02.2003) "Man kann Moral und Ethik nicht mit Technologie regulieren" <http://www.heise.de/ct/aktuell/meldung/34598>
- [BSI] Bundesamt für Sicherheit in der Informationstechnik (2002) Microsoft Internet Information Server Sicherheitsstudie, Isabel Münch (Hg.) http://www.bsi.bund.de/literat/studien/sistudien/IIS_2003.pdf
- [BSI] Bundesamt für Sicherheit in der Informationstechnik - BSI-Informationen zu Fachthemen - IT-Sicherheitszertifizierung - <http://www.bsi.de/literat/doc/ancczer.htm>
- [Cowan] Cowan, Crispin (2003) Software Security for Open-Source Systems Security & Privacy Magazine, IEEE, Vol.1, Iss.1, Jan.-Feb. 2003, Pages:38-45 <http://ieeexplore.ieee.org/iel5/8013/26429/01176994.pdf?isNumber=26429&prod=STD&arnumber=1176994&arNumber=1176994&arSt=+38&ared=+45&arAuthor=Cowan%2C+C>.
- [Debian] Debian GNU/Linux (2001) - Nachrichten - Debian GNU/Linux soll in Wahl-Software benutzt werden, <http://www.debian.org/News/2001/20010727>
- [Debian] Debian (2003) Debian-Untersuchungsbericht nach Serverkompromittierungen <http://lists.debian.org/debian-news-german/debian-news-german-2003/msg00058.html>
- [DFN-CERT] DFN-CERT Services GmbH (2002) Das SSL-Apache Handbuch http://www.dfn-pca.de/certify/ssl/handbuch/sslapache1_3/ssl13.html
- [DFN-CERT] DFN-CERT Services GmbH (2003) Kerberos 5 <http://www.cert.dfn.de/infoserv/dib/dib-2002-02-Kerberos5/>
- [Fuchs] Fuchs, Richard (2003) Auf dem Linux Weg http://www.politik-digital.de/egovernment/open_source/linux.shtml
- [Gehring] Gehring, Robert (2001) Unsichere Software - Eine systematische Betrachtung
- [Gehring] Gehring, Robert (2002) Open Source Software - Sicherheit im Spannungsfeld von Ökonomie und Politik <http://ig.cs.tu-berlin.de/forschung/ITSec/>
- [Han] Han (2003) ct - Webservices für E-Government als Open Source <http://www.heise.de/newsticker/data/han-06.11.03-000/>
- [Helmbrecht] Helmbrecht, Udo (2003) Freie Software in der Sicherheitsstrategie der Behörde http://www.wirtschaftsinformatik.de/wi_heft.php?op=heft&heftid=140

- [Iannacci] Iannacci, Federico (2003) The Linux managing model First Monday, volume 8, number 12 (December 2003), http://firstmonday.org/issues/issue8_12/iannacci/index.html
- [Köhntopp-Köhntopp-Pfitzmann] Köhntopp, Kristian; Köhntopp, Marit; Pfitzmann, Andreas (2000) Sicherheit durch Open Source? Chancen und Grenzen
- [Lawton] Lawton, George (2002) Open Source Security: Opportunity or Oxymoron? Computer, Vol.35 Iss.3, Mar 2002, Pages:18-21 <http://ieeexplore.ieee.org/iel5/2/21332/00989921.pdf?isNumber=21332&prod=STD&arnumber=989921&arNumber=989921&arSt=18&ared=21&arAuthor=Lawton>
- [Macromedia] Macromedia Endbenutzer-Lizenzvereinbarung (Nur für Desktopbenutzung) <http://www.macromedia.com/de/shockwave/download/license/desktop/>
- [OSDL] OSDL (2003) OSDL Lauches Linux Kernel Awareness Initiative http://www.osdl.org/newsroom/press_releases/2003/2003_11_26_beaverton.html
http://www.osdl.org/newsroom/graphics/linux_dev_process_graphic.jpg
- [OpenSSL] OpenSSL <http://www.openssl.org>
- [Raymond] Raymond, Eric S. (2000) The Cathedral and the Bazaar <http://www.catb.org/esr/writings/cathedral-bazaar/>
- [Schlesinger] Schlesinger, Steve (2003) Open-Source Security: Better Protection at a Lower Cost <http://www.linuxworld.com/story/25001.htm>
- [Schlifni] Schlifni, Manhard (2001), Sicherheitsanforderungen für ein "Electronic Voting System" http://members.chello.at/manhard.schlifni/Webpub/Kapitel2/2_2.html
- [Tietz] Tietz, Thomas (2000) Trojaner-Info, 22.06.2000 http://www.trojaner-info.de/news_unternehmen.shtml