

# Open Source Security

Ausarbeitung zum Referat

**Referenten, Matrikelnummer:**

Jan Suhr (223546)

Frank Reimann (176742)

Malwina Prokopczyk (221667)

Technische Universität Berlin

Fachbereich Informatik

Seminar „Information Rules 1“

Dozent: Prof. Dr. Lutterbeck / Dipl. Inf. Robert A. Gehring.

Januar 2004

Wir versichern hiermit, diese Arbeit selbständig und unter ausschließlicher  
Zuhilfenahme der in der Arbeit aufgeführten Hilfsmittel erstellt zu haben.

# Inhaltsverzeichnis

1. Einleitung .....	2
1.1 Sondierung des Themenbereiches .....	2
1.2 Gliederung.....	2
1.3 Begrifflichkeiten.....	3
2. Überlegungen und Ausarbeitung des Referates – Arbeitsbericht .....	4
3. Eine Geschichte zur Einführung .....	6
4. Probleme proprietärer Software .....	8
4.1 Absichtliche Sicherheitsprobleme.....	8
4.2 Aus der Geschichte lernen.....	9
4.3 Fehlerberichte .....	9
5. Technische Aspekte.....	11
5.1 Patches.....	11
5.2 Ist proprietäre Software (CSS) sicherer, weil der Quellcode nicht öffentlich ist oder ist Open Source Software (OSS) sicherer, weil der Quellcode offen ist?.....	12
6. Ökonomische Aspekte.....	17
6.1 Herstellerinteressen und ihre Problematik .....	17
6.2 Lock-In und seine negative Auswirkungen auf die Sicherheit .....	18
6.3 Sicherheitsinteressen von proprietären Herstellern und Benutzern .....	19
7. Ergebnis des Referats.....	21
7.1 Sicherheit.....	21
7.2 Literatur .....	22
8. Diskussion.....	23
9. Schluss.....	24
A Literaturverzeichnis .....	25

# 1. Einleitung

## 1.1 Sondierung des Themenbereiches

Spätestens seit dem Beginn der „Erfolgsstory von Linus Towalds und Linux“ (Moody 2001) ist der Begriff der freien Software immer mehr in das öffentliche Interesse geraten und hat zunehmend an Popularität gewonnen. Gab es vor einigen Jahren lediglich die durch entsprechende Vermarktung bekannten so genannten *proprietären* Produkte, wie beispielsweise das Betriebssystem Microsoft Windows oder Büroanwendungen, wie Microsoft Office etc, so wird mittlerweile *freie Software* oder *Open Source Software* eingesetzt. In der vorliegenden Ausarbeitung wird jedoch nicht zwischen freier und Open Source Software unterschieden sondern lediglich Open Source Software behandelt werden. Auf das abstrakte Konzept von Immaterialgütern und deren Eigentumsrechte soll an dieser Stelle nicht eingegangen werden (Vgl. Grassmuck 2002). Hier sollen lediglich die wichtigsten Eigenschaften proprietärer als auch Open Source Software dargestellt werden und auf Unterschiede in Bezug auf Sicherheitsaspekte aufmerksam gemacht werden. Der Software-Anwender hat also in vielen Fällen die Möglichkeit sich zwischen proprietärer und Open Source Software zu entscheiden. Insbesondere im Bereich von Internetdiensten ist der Leistungsumfang von Open Source Systemen meistens nicht schlechter als der von proprietären Alternativen. Allerdings werden im Folgenden nicht der Leistungsumfang, sondern lediglich die Sicherheitsaspekte der beiden Entwicklungsmodelle betrachtet werden. Auch sollen keine konkreten Produkte miteinander verglichen werden, sondern abstrakte Entwicklungsmodelle, um sich allgemeinen Aussagen annähern zu können.

## 1.2 Gliederung

Im Folgenden soll die Gliederung der Ausarbeitung vorgestellt werden. Das soll dazu dienen, sich leichter im Text zurecht finden zu können und deutlich machen, wo die Referenten ihre Schwerpunkte gesetzt haben und welche Kapitel bzw. welche Aufgaben bezüglich der Ausarbeitung bewältigt wurden. So wurde zu Beginn in Kapitel 1 das Thema des Referats ausführlich dargestellt, wichtige Begrifflichkeiten geklärt und ferner – in Kapitel 2 – ein Bericht über die Arbeitsweise während der Referatsausarbeitung und die dabei auftauchenden Schwierigkeiten und Entscheidungen verfasst. Diese Aufgabe wurde von Jan Suhr übernommen, der auch größtenteils den Aufbau des Referates erarbeitet hat. Das Kapitel 3,

nämlich die Erzählung einer fiktiven Geschichte fiel in den Aufgabenbereich von Malwina Prokopczyk. Anschließend werden in Kapitel 4 grundlegende Probleme von proprietärer Software dargelegt. Dieses Thema hat Jan Suhr vertiefend dargestellt. Im 5. Kapitel werden technische Aspekte konträr diskutiert, um nachfolgend Thesen formulieren zu können, um diese zum Schluss des Referates zur Diskussion zu stellen. Dieses Thema wurde weitestgehend von Frank Reimann behandelt und von diesem schriftlich verfasst. Ähnlich wurde auch in Kapitel 6 verfahren, was weitestgehend von Malwina Prokopczyk erarbeitet wurde und von ihr in schriftlicher Form vorliegt. In Kapitel 7, 8 und 9 werden die Ergebnisse dargestellt und nochmals kurz reflektiert, was zum Aufgabenbereich von Jan Suhr gehörte. Am Ende der Ausarbeitung befindet sich ein Literaturverzeichnis.

### **1.3 Begrifflichkeiten**

Mit dem bereits eingeführten Begriff - *Open Source Software* – sind Systeme gemeint, deren Quellcode uneingeschränkt einsehbar ist und an dessen Erstellung freiwillige Entwickler mitwirken können. Alle übrigen Systeme werden als *proprietär* bezeichnet. Eine Software deren Quellcode einsehbar ist, deren Entwicklung aber ausschließlich durch ein Unternehmen betrieben wird, ist ein Grenzfall. In der Praxis existieren jedoch wenige dieser Beispiele, so dass diese Betrachtung vernachlässigt werden kann.

Mit dem Begriff der *Sicherheit* sind primär Aspekte gemeint, die sich auch als „*passive Sicherheit*“ bezeichnen ließen. Dazu gehört die Abwesenheit von Sicherheitslöchern, Buffer-Overflows, Hintertüren etc. „*Aktive Sicherheit*“ definiert die Leistungsmerkmale (sog. „*Features*“), die eine Software aufweisen kann. Auf diese wird lediglich Bezug genommen, wenn ausdrücklich von aktiver Sicherheit gesprochen wird.

## **2. Überlegungen und Ausarbeitung des Referates – Arbeitsbericht**

Als Ausgangspunkt der Vorliegenden Ausarbeitung dienten drei Literaturempfehlungen der Seminarleiter Herr Prof. Dr. Lutterbeck und Herr Robert A. Gehring. Wir haben uns bemüht eine abwechslungsreiche und informative Bibliographie zu erstellen, um einzelne Thesen multiperspektivisch betrachten zu können, und nicht etwa zu einer einseitigen Sichtweise zu gelangen. Für die Erstellung der Bibliographie wurde eine Literaturrecherche durchgeführt, wobei auf die üblichen Suchmaschinen, Bibliotheksinformationssysteme und Dokumentendatenbanken von Fachzeitschriften zurückgegriffen wurde. Die so erschlossenen Dokumente waren zum größten Teil über das Internet verfügbar, wobei diese Ergebnisse durch eine Recherche in der Bibliothek ergänzt wurden.

Nach intensiver Sichtung der einzelnen Aufsätze wurde eine theoretische Herangehensweise beschlossen, wobei wichtige sicherheitsrelevante Eigenschaften der beiden Entwicklungsmodelle gegenübergestellt werden sollten, um zu einem allgemeinen Ergebnis zu gelangen. Empirische oder statistische Untersuchungen ließen wir weitgehend unberücksichtigt, da dessen Ergebnisse lediglich für die untersuchten Systeme zu der untersuchten Zeit gegolten hätten und nicht allgemeiner Natur gewesen wären. Damit das „Stoffmengenproblem“ (Wahl et al. 1991) bewältigt werden konnte, wurde auf Organisationshilfen zurückgegriffen, wie beispielsweise das Prinzip des „advance organizer“ (Wahl et al. 1991)(Ausubel 1974), um gedankliche Zusammenhänge dieser Fülle an Informationen schneller erfassen zu können oder auch das Prinzip des SOL (Herold, Landherr, 2001), wobei jeder Teilnehmer der Referatsgruppe eine Anzahl an Texten zugeteilt bekam, um diese vorzubereiten und zu einem späteren Termin zu referieren. Damit sollte die Stoffmenge bewältigt werden und die Möglichkeit zu gemeinsamer Reflexion gegeben werden. Die daraus entstandenen Resultate dienten als weitere Arbeitsgrundlage für die Strukturierung des Vortrages, da somit die Vorarbeit für eine Darstellung der unterschiedlichen Argumente vorbereitet war. Bei der Darstellung der Argumente wurde uns bewusst, dass viele Aspekte sehr ambivalent ausgelegt werden und wir uns somit aus Zeitgründen auf eindeutige und konkrete Argumente konzentriert haben. Damit einher gingen Probleme bei der Strukturierung und dem logischen Aufbau des Vortrages für das Plenum. Wir konzentrierten uns also auf eine klare Argumentation und bereiteten zur Auflockerung

eine fiktive Begebenheit vor, wobei es sich um Sicherheitsproblematik in Anlehnung an die behandelte Thematik handelte.

Ein weiterer Faktor für eine übersichtliche Darstellung war der Entschluss für eine thematische Gliederung, wobei Argumente und Gegenargumente direkt auf einer Folie gegenübergestellt wurden, sofern beide Standpunkte vertreten wurden.

Unsere bisherige Ausarbeitung verteilten wir in Form eines Handouts nach dem Referat, welches wir ausgehend von den bereits vorbereiteten Folien anfertigten, so dass das Handout und die Folien die gleiche Gliederung aufweisen. Dadurch lässt sich ohne weiteres zu einer bestimmten Folie die entsprechende Stelle im Handout finden.

Der Vorsatz, sich mit dem Vortragen des Referates abzuwechseln, um eine gewisse Vortragsdynamik zu erzielen, ließ sich nicht verwirklichen, da wir keine eigenständigen Kapitel hatten.

### **3. Eine Geschichte zur Einführung**

Um den genauen Unterschied zwischen Angriff und Verteidigung im Aspekt der Softwaresicherheit kennen zu lernen, wird folgende Einleitungsgeschichte dargestellt. Wir nehmen an, dass ein Betriebssystem wie Windows 2000 1.000.000 Fehler hat, die insgesamt eine MTBF (mean time before failure) von 1.000.000.000 haben. MTBF bedeutet mittlere Zeit bis zum Auftreten des Fehlers. Paddy arbeitet für die Irish Republican Army. Seine Aufgabe ist es, in den Computer der British Army einzubrechen und die Informantenliste in Belfast zu holen. Brian ist ein Sicherheitsspezialist und sein Job ist es, solche Angreifer wie Paddy zu stoppen. Das heißt also, dass er früher als Paddy alle Fehler im Betriebssystem ausfindig machen sollte.

Brian verfügt im Gegensatz zu Paddy über den Quellcode, hat den Zugang zu Insider-Informationen und viele Mitarbeiter, die ihm helfen. Er testet Software 10.000.000 Stunden pro Jahr und findet mit Hilfe von Mitarbeitern und zusätzlichen Informationen ungefähr 100.000 Fehler pro Jahr. Paddy arbeitet tagsüber, deswegen kann er nur 1.000 Stunden im Jahr dem Testen von Software widmen.

Nach einem Jahr findet Paddy einen Fehler, während Brian schon 100 gefunden hat. Die Wahrscheinlichkeit, dass Brian genau den Fehler gefunden hat, den Paddy ausnutzen will, beträgt 10%. Nach zehn Jahren könnte es sein, dass Brian diesen Fehler findet. Dann hat Paddy aber schon neun andere Fehler gefunden. Es ist sehr unwahrscheinlich, dass der Verteidiger die gleichen neun Fehler ausfindig macht.

Diese Kurzgeschichte zeigt, in welcher schwieriger Situation sich die Verteidiger befinden und dass die Seite der Angreifer bei weitem überlegen ist. (Anderson 2001)

<b>Name:</b>	<b>Paddy, IRA</b>	<b>Brian, BA</b>
<b>Windows 2000:</b>	1.000.000 Fehler	
<b>MTBF insgesamt:</b>	1.000.000.000 Stunden	
<b>Ausstattung:</b>	-	Quellcode, Mitarbeiter, Insider-Informationen
<b>Arbeitet im Jahr:</b>	1000 h	10.000.000 h
<b>MTBF/Fehler:</b>	1000 h	10 h (wg. Ausstattung)
<b>Bugs/Jahr:</b>	1	100.000
<b>Erfolgswahrsch.:</b>	90%	10% (=1Mil./100.000)

## 4. Probleme proprietärer Software

Proprietäre Software beinhaltet als elementare Eigenschaft das Geheimhaltungsprinzip. Die Algorithmen und Wirkungsweisen sind nicht ohne weiteres ersichtlich, sondern werden verborgen gehalten. Setzt man proprietäre Systeme ein, so ergeben sich damit grundlegende Sicherheitsprobleme, die in diesem Kapitel betrachtet werden sollen. Darüber hinaus soll ein Blick auf das Verhältnis zwischen Benutzer und Hersteller geworfen werden, da sich dies auf sicherheitsrelevante Fehlerberichte auswirken kann.

### 4.1 Absichtliche Sicherheitsprobleme

Ein elementares Problem von proprietärer Software ist, dass es nicht ausgeschlossen werden kann, dass die eingesetzte Software absichtlich, z.B. aus politischen Gründen, unsicher gemacht wurde. Dies kann bedeuten, dass Hintertüren („backdoors“) oder anderer bösartiger Code („malicious code“) eingebaut wurden, (Federrath, Pfitzmann 1999) um dadurch beispielsweise Zugriff auf fremde Systeme zu bekommen oder an bestimmte Informationen zu gelangen. Außerdem ist nicht ersichtlich, *wie* bestimmte Funktionen implementiert sind, was für die Sicherheit eines Systems eine entscheidende Rolle spielt.

Bei der Verwendung eines proprietären Systems, muss also dem Hersteller vertraut werden. Es ist aber fraglich, ob eine ausreichende Vertrauensgrundlage existiert. Insbesondere kann diese Vertrauensgrundlage durch staatliche Interessen eingeschränkt werden. Beispielsweise möchten US-amerikanischen Behörden unbekannt aber existierende Sicherheitslücken für ihre eigenen Interessen ausnutzen. Hersteller sollten demnach Sicherheitslücken nur korrigieren, wenn diese öffentlich bekannt würden. (Anderson 2002)

Ein weiteres Beispiel für unsicher gemachte Software stellt der Fall des sog. „NSAKey“ dar. Hierbei wurde bekannt, dass es einen zweiten kryptographischen Schlüssel gibt, mit dem auf Windows Betriebssystemen möglicherweise sicherheitsrelevante Module ohne Zutun des Benutzers installiert werden können. Der verwendete Name NSAKey erinnerte dabei an den US-amerikanischen Geheimdienst NSA. Microsoft dementierte einen Zusammenhang mit der NSA, jedoch konnte keine eindeutige Klärung erreicht werden. (Kettmann, Glave 1999)

Absichtlich unsicher gemachte Systeme erlauben nicht nur eine Ausnutzung durch die veranlassenden Instanzen, wie z.B. staatlichen Behörden oder Hersteller. Sie können ebenfalls als Einfallstor für andere böswillige Angreifer dienen. (Lutterbeck et al. 2000) Es wird später

näher darauf eingegangen werden, dass unbekannte Sicherheitslücken durchaus sicherheitsproblematisch sein können und es damit das Prinzip des "Security-by-obscurity" zu vermeiden gilt.

## **4.2 Aus der Geschichte lernen**

Die Kryptologie ist im Vergleich zur Informatik eine alte Wissenschaft. Es wird dort seit langer Zeit das Prinzip verfolgt, dass Geheimnisse eine Sicherheitslücke darstellen und daher zu vermeiden sind. Insbesondere die Teile, die sich nicht ohne weiteres ändern lassen, sollten nicht geheim gehalten werden müssen. Daher werden in der Kryptologie Algorithmen veröffentlicht, um von möglichst vielen Experten auf Sicherheitsmängel begutachtet zu werden, ehe sie eingesetzt werden. Bei deren Verwendung müssen lediglich die Schlüssel, aber nicht der komplette Algorithmus, geheim gehalten werden. (Diffie 2003)(Anderson 2002)

Beispielsweise weist die DVD-Technologie ein kryptographisches Verfahren ("Content Scrambling System" (CSS)) auf, das das Abspielen von DVD's ausschließlich mit lizenzierten Abspielgeräten erlauben soll. Dieses Verfahren wurde geknackt und dessen Schwachstellen veröffentlicht. Da das Verfahren im Nachhinein nicht geändert werden konnte, lässt sich seit dem das CSS in jedes beliebige Abspielgerät implementieren, welches damit alle DVD's abspielen kann. (Warren et al. 2000)

## **4.3 Fehlerberichte**

Proprietäre Software wird von Herstellern an ihre Kunden verkauft. Dadurch lässt sich ihr Verhältnis durch einen Vertrag beschreiben, ohne dass eine weitere Bindung zwischen ihnen besteht. Dies könnte der Grund dafür sein, dass - nach Anderson - die Anwender von proprietärer Software weniger Fehlerberichte (sog. „Bugreports“) schreiben als Benutzer von Open Source Software. (Anderson 2002) Diese Bugreports sind für die Sicherheit einer Software wichtig, da deren Entwickler durch sie auf bisher unbekannte Fehler aufmerksam gemacht werden, und diese dadurch korrigieren können. Da Open Source Software häufig auf freiwillige Tester angewiesen ist, ist es nicht verwunderlich dass deren Entwickler stark mit den Benutzern kooperieren müssen, um diese nicht zu enttäuschen (Lutterbeck et al. 2000) Für den Benutzer ist es sicherlich eine erfolgreiche Erfahrung, wenn ein aufgetretenes Problem auf Grund seines Berichtes in kurzer Zeit korrigiert wird, und er die Software dadurch fehlerfrei verwenden kann. Dies ist bei proprietärer Software nicht möglich, da der

Benutzer meistens nicht mit dem zuständigen Entwickler in Kontakt treten kann, sondern sich an den Support-Bereich wenden muss, der die Anfrage lediglich weiterleiten kann. Da in Firmen eine Weisungshierarchie besteht, können Reibungsverluste auftreten, so dass die Korrektur länger dauert als bei Open Source Software. (Payne 2002) (Lutterbeck et al. 2000)  
Für die Auseinandersetzung mit weiteren Problemen der Fehlerbehebung siehe Abschnitt 5.1.

## 5. Technische Aspekte

### 5.1 Patches

Wenn ein sicherheitsrelevanter Fehler in einer Software auftritt, so ist der Anwender stark eingeschränkt, falls er proprietäre Software einsetzt, die nur in binärer Form ausgeliefert wird. Er muss schlichtweg mit seiner “verwundbaren” Software warten bis der Hersteller einen Patch oder ein Update zur Verfügung stellt. Der Hersteller proprietärer Software kann sogar entscheiden, ob es ihm überhaupt wert ist ein Update für dieses spezielle Problem zu veröffentlichen. Der Anwender ist der Gnade des gewählten Anbieters ausgeliefert. Auf der anderen Seite haben Nutzer der freien Software eine Vielzahl an Auswahlmöglichkeiten. Sogar dann, wenn der Anbieter keinen offiziellen Patch herausgibt. Oftmals arbeiten Benutzer zusammen um einen eigenen zu entwickeln. Die Problemlösung kann mit Hilfe des frei verfügbaren Quelltextes unabhängig erarbeitet werden. (Payne 2002)

Open Source Software vermeidet in Bezug auf Patches die Abhängigkeit vom Hersteller.

Hier können Firmen, Behörden, andere Organisationen oder gar Privatpersonen selbst eigene Verbesserungen in Bezug auf die Sicherheit durch den Quelltext erarbeiten und auf die Software anwenden. (Beispiel: NSA Security-Enhanced Linux 2000) Bei proprietärer Software kann der Anbieter nicht nur bestimmen wann und ob ein Patch, bzw. Update den Nutzer erreicht, sondern er bestimmt auch wie er ein Sicherheitsproblem behebt. Dabei kommt es nicht selten vor, dass ein Patch das Problem nicht grundlegend beseitigt sondern nur einzelne Auswirkungen behebt. Als Folge kann das gleiche Sicherheitsproblem erneut auf eine andere Art und Weise ausgenutzt werden und es muss gegebenenfalls erneut ein Patch erstellt werden.

Ein weiteres Argument für eine bessere Versorgung mit Patches bei Open Source Software ist, dass sich die offizielle Bestätigung eines Sicherheitsloches negativ auf den Aktienwert eines Unternehmens auswirken kann, welches mit proprietärer Software gegen die Konkurrenz antritt. (Payne 2002) Sicherheitslöcher verkaufen sich nicht so gut wie Features und können auch dem Image einer Firma schaden und damit das Vertrauen der Nutzer oder potentiellen Käufer in das Produkt schwächen. Man kann das zum Beispiel bildhaft mit einem Autohersteller vergleichen, der verkündet, dass sein aktuelles Auto-Modell 2000 Fehler weniger als das letzte Modell hat und die Bremsen jetzt auch funktionieren. Da ist es naheliegend, dass sich der Kunde fragen wird, wie viele Fehler denn noch im aktuellen

Modell vorhanden sind, wenn das letzte Modell in einem so schlechten Zustand überhaupt verkauft wurde.

Nach empirischen Informationen hat sich gezeigt, dass "Sicherheitslöcher" bei Open Source Software schneller beseitigt werden, als bei proprietärer Software. (Payne 2002) (Lutterbeck et al. 2000)

Es wurden auch weniger Sicherheitslücken in Open Source Software gefunden. (Wheeler 2003)

Ein grober Fehler ist es Updates auf einem Server bereit zu stellen, auf dem das "Einfallstor" für den Schädling nicht beseitigt wurde. Wenn der schützende Patch nicht auf dem Server eingespielt wurde, der dieses Update für andere verfügbar machen soll, dann kann es sein, dass das schädliche Programm den Server befällt und ein Vertriebs des Updates unmöglich wird.

So ist es Microsoft mit einem Wurm passiert, der Windows-Server befallen hat, die aktuelle Windows-Patches bereitstellen sollten. (Brunnstein 2003)

Beispielsweise wurden Microsoft-Server, welche für das Windows-Update zuständig sind im Juli des Jahres 2001 vom "Code-Red" Wurm befallen. Wer in diesem Zeitraum die Seite besuchte wurde mit dem Text "Welcome to <http://www.worm.com>! Hacked By Chinese!" begrüßt. (Leyden 2001)

## ***5.2 Ist proprietäre Software (CSS) sicherer, weil der Quellcode nicht öffentlich ist oder ist Open Source Software (OSS) sicherer, weil der Quellcode offen ist?***

Zur Unterstützung dieses Gedankens kann man sagen, dass sich Sicherheitslöcher schwieriger in einem binären Programmcode finden lassen als in Quelltextform. Dieser Schutz ist jedoch nicht verlässlich und lässt sich mit verschiedenen Techniken fast aufheben. Bei Programmen mit frei verfügbarem Quelltext ist es üblich in den Quellen nach bestimmten Funktionen zu suchen, von denen bekannt ist, dass diese oftmals zu Sicherheitsproblemen führen. Eine ähnliche Technik lässt sich bei Programmen anwenden, die im Binärkode vorliegen. Der Programmcode wird in eine maschinennahe Programmiersprache (Assembler) zurückübersetzt (disassembliert). Diese Technik heißt Reverse Engineering. Nun können Systemaufrufe zu den problematischen Funktionen gefunden und zurückverfolgt werden.

Ähnlich dazu enthalten externe Referenzen auf dynamisch verknüpfte Module auch diese Information. Wie man sieht gibt es Unterschiede beim Finden von sicherheitsrelevanten Fehlern in Quelltexten von Programmen oder im Binärcode von Programmen. Es ist aber nicht bedeutend schwerer Fehler im Binärcode zu finden. Wenn man auf dieses Argument aufbaut, so scheint es auf jeden Fall sogar ein bedeutender Vorteil für Open Source Software zu sein, weil sogar ein Anfänger, der den Quellcode prüft, potentielle Probleme im Programmcode für sich beseitigen kann. Im Kontrast dazu ist es extrem schwierig ähnliche Probleme in Programmen, die nur im Binärcode vorliegen, zu beheben. (Payne 2002)

“Es ist einfach unrealistisch, sich auf Geheimhaltung bei der Sicherheit von Computersystemen zu verlassen. Wahrscheinlich wird man die exakten Funktionsprinzipien eines Programms vor einer breiten Öffentlichkeit geheim halten können, aber wird man verhindern können, dass der Code per Reverse-Engineering von ernsthaften Gegnern geknackt wird? Wahrscheinlich nicht.“ (Diffie 2003)

Die Hersteller investieren aber teilweise einige Energie darin ihre Systeme inkompatibel zu machen (siehe Lock-In Effekt) und Reverse Engineering zu erschweren, um der Konkurrenz keine Einblicke in die Codebasis und die verwendeten Protokolle zu ermöglichen. Das geht bis zur Verwendung kryptografischer Verfahren. (S.4) (Anderson 2001)

Unserer Meinung nach ist ein Schutz jedoch nur ausreichend gut, wenn der Aufwand zum Beseitigen des Schutzes mehr Kosten verursacht als die zu schützenden Informationen wert sind.

Hier zeigt sich die Problematik. Während es einem Anbieter reicht der Konkurrenz z.B. den Zugriff auf den Quelltext selbst oder Protokolldetails zu erschweren können andere Interessengruppen erheblich mehr Aufwand beim Umgehen der Sicherheitsbarriere leisten um an Informationen zu kommen, auf die z.B. über das Programm zugegriffen wird oder die mit Hilfe des Protokolls übertragen werden und die erheblich mehr wert sind. Damit relativiert sich diese Art des Schutzes in Bezug auf Sicherheit. Es gibt auch Beispiele aus der Praxis, bei denen kryptografische Verfahren durch Implementierungsfehler und geschickte Angriffe ausgehebelt werden konnten. (z.B. GnuPG (Lau 2003), Microsoft Xbox (Xbox Linux Project 2003))

Es reicht also nicht kryptografische Verfahren zu verwenden, sondern sie müssen auch fehlerfrei implementiert werden um Datensicherheit zu gewährleisten. Bei Open Source Software lässt sich das überprüfen. Jeder Experte kann den Quellcode einsehen wenn es nötig ist. (Lutterbeck et al 2000)

Open Source Software erlaubt es, durch die Offenlegung des Quelltextes, von den Anwendern an ihre individuellen Anforderungen angepasst zu werden. Natürlich können dadurch auch die Anforderungen an "aktive Sicherheit" besser erfüllt werden. (Payne 2002)

Andererseits haben die Anwender in der Regel nicht die nötigen Ressourcen und das Know-How um die Qualität der Software sicherzustellen, so dass dies zu (passiven) Sicherheitsproblemen führen kann. (Microsoft 2003)

Der Quellcode muss durch qualifizierte Experten überprüft werden. Alleine der Umstand, dass eine Software Open Source Software ist, macht sie nicht sicherer. (Payne 2002)

Die Änderungen können persönlich, durch eine Benutzergruppe oder durch die eine beauftragte Firma geschehen. Die Anpassbarkeit der Software ist eine Option und keine Verpflichtung. Deswegen ist man im Falle der Nutzung dieser Option auch selbst für die Sicherheit verantwortlich. Wir können darin keinen Nachteil des Open Source Prinzips erkennen.

Ein Vorteil von Open Source Software ist die Möglichkeit eines sogenannten Peer Review. Dabei handelt es sich um nachträgliches Betrachten des Quellcodes. Hier bekommen unabhängige Sicherheitsexperten die Möglichkeit korrigierend einzugreifen. Durch den Prozess des Peer Review werden viele Sicherheitslücken wirklich gefunden und damit Fehler beseitigt. (Lutterbeck et al 2000)

Microsofts Erfahrungen mit der eigenen Shared Source Lizenz besagen jedoch, dass bisher keine neue Sicherheitslücken durch Offenlegung innerhalb dieses Programmes gefunden wurden und Bug-Reports eher aus anderen Quellen stammen. Qualifizierte Personen sind ohne intellektuellen Ansporn oder Bezahlung nicht bereit solche Arbeiten durchzuführen. (Needham 2002)

Ein populäres Beispiel für die Stärke des Peer Review Prozesses in Bezug auf die Sicherheit ist die unterstellte Schwierigkeit für einen Angreifer Hintertüren (eine sogenannte "Back Door") in ein Open Source Programm unbemerkt einzuschleusen. Eine "Back Door" ist bösartiger Code, der, wenn er einmal in ein legitimes Programm eingeschleust wurde, es z.B. einem Angreifer einfach und unbemerkt ermöglicht die existierenden Sicherheitsmechanismen zu umgehen. So ist es z.B. möglich mit einer Hintertür Personen bei einem Betriebssystem als Administrator anzumelden, ohne dass man das Passwort angeben muss, wenn man sich von einer bestimmten Internet Adresse (IP Adresse) mit dem System verbindet. Entsprechend den Fans ist es bei Open Source nahezu unmöglich für einen

Angreifer solchen Code in einem Programm zu verstecken, wenn jede beliebige Anzahl von Leuten Zugriff bekommen und den Quellcode einsehen kann. (Payne 2002)

So ist z.B. eine Hintertür im Linux Kernel noch vor der Veröffentlichung gefunden worden. (Poulsen 2003)

Ein weiteres aufschlussreiches Beispiel war die absichtliche Einbettung einer Hintertür in die Borland/Inprise Interbase Datenbank durch autorisierte und legitimierte Programmierer der Firma 1992. Der Zweck dieser Hintertür war nicht böse, hätte aber missbraucht werden können, falls unautorisierte Personen darauf gestossen wären, weil damit der Zugriff auf alle Installationen der Datenbank möglich war. Interessanterweise wurde diese Angelegenheit 9 Jahre nach dem Hinzufügen der Hintertür entdeckt und beseitigt, jedoch nur kurz nachdem das Produkt als Open Source veröffentlicht wurde. (CERT Advisory CA-2001-01) Da Zugriff zum Quellcode erforderlich ist um eine Hintertür zu installieren, ist proprietäre Software für eine lange Zeit verwundbar. Deswegen gab es viele Spekulationen um die mögliche Installation einer Hintertür in Microsoft Code während eines erfolgreichen Einbruchversuches in das Netzwerk des Unternehmens im Jahr 2000. (siehe auch: Probleme proprietärer Software) Ein anderer Faktor, der es besonders schwer macht böse Code einzuschleusen, liegt in der Natur des Entwicklungsprozesses, der oft bei diesen Projekten verwendet wird. Viele Projekte nutzen das Concurrent Versioning System (CVS) Programm um Zugriffe und Veränderungen zu kontrollieren, so dass diese reguliert und zurückverfolgt werden können. (Payne 2002)

Gutes Peer-Review kann auch bei OSS durchgeführt werden, falls die notwendigen Kapazitäten vom Softwareanbieter bereit gestellt werden (Payne 2002), vorausgesetzt der Anbieter hat Interesse an dieser Art der Sicherheitsüberprüfung.

Die meisten Fehler werden in neuem Code entdeckt. Deswegen erlaubt OSS einem Angreifer neuen Code einfach zu erkennen, da er den Quellcode der aktuellen Version mit der letzten Version vergleichen kann. Damit lassen sich Codestücke in denen noch "Sicherheitslücken" enthalten sind leichter herausfinden. (Anderson 2002)

Vorteile ergeben sich wiederum beim Basar-Modell als Methode der Open Source Software Entwicklung durch starke Modularisierung. Diese verbessert die Übersicht über die Software (die vorhandene Codebasis) und Fehler lassen sich besser eingrenzen, sowie beheben. Ausserdem wird der Evaluierungsprozess erleichtert. (Lutterbeck et al 2000)

Durch die starke Modularisierung ist auch die Verteilung des Quellcodes auf eine grosse Entwicklergemeinde möglich. Es stehen vor allem technische Interessen im Vordergrund, was der Sicherheit der Software zugute kommt. (siehe auch ökonomischer Aspekt der Sicherheit)

Weitere Vorteile sind auch die kurzen Veröffentlichungszyklen. Durch häufige Veröffentlichungen wird erreicht, dass Fehlerberichte (sogenannte Bug-Reports) auf den aktuell von den Entwicklern bearbeiteten Code passen. Bei proprietärer Software sieht das meistens anders aus. Dort findet man lange Veröffentlichungszyklen vor und die Entwickler arbeiten meist schon auf einer ganz anderen Codebasis. Wenn lange nach der letzten Veröffentlichung Bug-Reports eingesendet werden, so muss sich der Entwickler darauf verlassen, dass dieser Teil der Software bereits einer Korrektur unterzogen wurde oder es besteht die Gefahr, dass Fehler in die Software unbeabsichtigt einprogrammiert werden, da sich der Quellcode schon stark verändert hat. (Challet, Le Du 2003)

Open Source Software propagiert auch ein Paradigma der aus der Software Technik. Es handelt sich dabei um die Wiederverwendbarkeit. Bereits entwickelte und getestete Software kann in Form von Software-Bibliotheken in neue Projekte einfließen. Die dort neu entwickelte Software baut auf vorhandener bewährter Software auf und kann damit nicht nur sicherer sondern auch schneller entwickelt werden. (Lutterbeck et al 2000)

Software ist von einzelnen Compilern und Entwicklungswerkzeugen abhängig. Auch diese stellen Sicherheitsprobleme dar. (Lutterbeck et al 2000)

Bei OSS kann der Compiler leichter mit einer Hintertür versehen werden, weil der Zugriff auf den Quellcode des Compilers leichter ist. (Payne 2002)

Mit einem böse manipulierten Compiler können auch korrupte Programme erzeugt werden.

Wenn der Nutzer seine Compiler-Software aus den offiziellen Softwarequellen bezieht gelten jedoch dieselben Schutzmassnahmen (siehe oben) wie für andere Software auch. Wie man aber an der Bedeutung der Compiler und Entwicklungswerkzeuge erkennt ist hier besondere Vorsicht geboten, da ein korrupter Compiler viele Programme erzeugen kann, die böse Code enthalten und dann verbreitet werden.

## **6. Ökonomische Aspekte**

Die meisten Menschen verbinden die Sicherheit oft nur mit dem technischen Aspekt der Softwareentwicklung. Nur wenige wissen, dass die Software-Sicherheit von den ökonomischen Aspekten sehr stark abhängt. (Anderson 2001) Software-Systeme können in zwei große Bereiche geteilt werden, wobei einer „Marketing Architecture“ und der andere „technical architecture“ heißt. Mit dem ersten befassen sich Leute wie Marketing Manager („marketect’s“), die genau wissen, wie man am besten ein Produkt vermarkten kann. Für die „tarchitect’s“ ist es das wichtigste, dass Software schnell und fehlerfrei funktioniert. Sicherheit ist so ein Bereich, wo „marketect“ und „tarchitect“ zusammenarbeiten müssen. (Hohmann 2003)

### **6.1 Herstellerinteressen und ihre Problematik**

Wenn wir an proprietäre Software denken, haben wir vor unseren Augen große Unternehmen, die um jeden Preis auf dem Markt ihre Produkte verkaufen wollen, damit sie davon sehr gut profitieren können. Für ein Unternehmen ist es besonders wichtig, die Produktionskosten so niedrig wie möglich zu halten und die Innovationen so schnell wie möglich auf dem Markt zu bringen. Man nennt das Time-To-Market- Druck.(Payne 2002) Sehr oft benachteiligt deswegen ein Software-Unternehmen die Sicherheit und Qualität des Produkts. Diese Software-Produkte missbrauchen das Vertrauen der Kunden und somit bestätigen sie, dass eine proprietäre Software ein unsicheres, aber gut vermarktbares Produkt bedeutet. Wenn man aber genau den Markt betrachtet, findet man sehr schnell heraus, dass passive Sicherheit im Gegensatz zur aktiven sich sehr schwer vermarkten lässt.(Payne 2002) Das ist der nächste Punkt, wo die großen und innovativen Unternehmen wieder bei der Entwicklung von unterschiedlichen Features profitieren.

Open Source Software wird nach rein technischen Aspekten entwickelt.(Payne 2002) Open Source Anhänger haben keine Absicht mit ihren Produkten auf dem Markt zu gehen, um die zu verkaufen und Gewinne zu machen. Das bedeutet, dass sie unter keinerlei ökonomischen Druck gesetzt werden, was sich wieder positiv auf die Sicherheit auswirken kann. Aber wenn jemand nur an technischen Aspekten denkt, kann er auch die eigentlichen Interessen des Benutzers nicht beachten.(Microsoft 2002) Sehr viele Kunden interessieren sich für gefertigte Benutzeroberflächen, und dafür, dass sie leicht und verständlich sind. Manche Leute erwarten etwas mehr als die Versicherung von der Seite des Herstellers, dass eine Software sicher ist. So will man am besten auch selber in der Lage sein, den Quellcode zu sehen oder auch zu

überprüfen. Letztendlich will der Benutzer die sicherste Software wählen können. Durch Offenlegung des Quellcodes ist es möglich die Software miteinander zu vergleichen und somit einen Wettbewerb, mit einem positiven Einfluss auf die Sicherheit, zu bewirken. (Lutterbeck et. al.). Das ist aber leider nur bei Open Source zutreffend, deswegen ist der Vergleich mit proprietärer Software nicht möglich. Andererseits ist es auch verständlich, dass die großen Unternehmen den Quellcode und damit die Fehler im System für sich behalten wollen. So eine Veröffentlichung von Sicherheitslücken könnte einen sehr negativen Einfluss auf den Aktienwert des Unternehmens haben.(Payne 2002) Das beweist uns aber, dass die Herstellerinteressen nicht immer mit den Interessen der Kunden übereinstimmen, was zu Problemen der Software-Sicherheits-Garantie führen kann.

## **6.2 Lock-In und seine negative Auswirkungen auf die Sicherheit**

Normalerweise wird der Wert eines Software-Produkts nach seinen Eigenschaften wie Sicherheit, Qualität, technische Innovationen und Benutzerfreundlichkeit bestimmt. Es ist aber eine Tatsache, dass die Kunden genau die Software-Produkte sehr schätzen, die am meisten verkauft wurden. Die Sicherheitsmanager entscheiden sich öfters für Produkte und Dienste von großen, bekannten Unternehmen, obwohl das keine optimale Lösung für die Firma ist. Falls etwas schief gehen sollte, können sie sich immer noch auf den bekannten Hersteller berufen und somit werden sie wahrscheinlich nicht gefeuert werden.(Anderson 2001) Der ursprüngliche Wert einer Ware wird jetzt anders bestimmt. Je mehr Leute eine bestimmte Software haben, desto hochwertiger ist sie.(Anderson 2001) Das haben auch die großen Unternehmen erkannt. Wenn eine Firma auf dem Markt ihre starke, dominante Position beibehalten will, muss sie nicht nur neue Kunden gewinnen, sondern auch die „alten“ behalten. Und somit kommen wir zu dem Lock-In Effekt, der die Benutzer von den Produkten einer bestimmten Firma abhängig macht. Das heißt, wenn man sich für die Produkte anderer Firmen als bislang entscheidet, muss man mit sehr hohen Switching-Kosten rechnen.(Anderson 2001) Die meisten Benutzer können sich das nicht leisten, deswegen bleiben sie auch bei der proprietären Software, die sie bisher benutzt haben. Somit verpassen die meisten Benutzer eine Chance die Software, die sinnvoller ist, zu benutzen. Ein kommerzielles Software-Produkt, für das man unglaublich viel bezahlen muss, ist mit einem Produkt der Open Source Software, das man umsonst haben kann, gleich.(Anderson 2001) Die bekannten Hersteller der proprietären Software versuchen ihre Produkte so zu manipulieren, damit sie mit Produkten anderer Firmen inkompatibel bleiben, was die

Switching-Kosten und damit auch den Lock-In erhöht. Letztendlich fühlen sich die Hersteller der proprietären Software nicht dazu verpflichtet, Software ohne Sicherheitslücken herzustellen.(Anderson 2001) Man könnte sagen, dass in dieser Situation die Nutzung von Open Source Software für den Benutzer günstiger ist. Die Entwickler haben keine Interessen an Marktanteilen und Gewinnen. Die kommerziellen Hersteller behaupten, dass nur die proprietären Software-Entwickler sich um die Interessen des Kunden kümmern.(Microsoft 2003) Damit ist die Interoperabilität der Produkte von proprietärer Software garantiert. Je mehr zufriedene Benutzer, desto größer ist der Umsatz.(Microsoft 2003) Bei der Open Source Software hat man das Problem, dass sehr viele Entwickler an einem Projekt engagiert sind. Dabei kann jeder was verändern, was möglicherweise zu Inkompatibilitäten führen kann.(Microsoft 2002) Während der Entwicklung der Software wird ein Projekt auf mehrere Projekte bzw. Aufgaben aufgeteilt (sog. Forking), deswegen wird die Interoperabilität kaum bewahrt. Außerdem aus den lizenzrechtlichen Gründen dürfen einige offene Standards nur in proprietärer Software und nicht in Open Source Software implementiert werden.(Microsoft 2002) Man kann sagen, dass Inkompatibilität Lock-In positiv beeinflusst, was nicht unbedingt ein Vorteil für die Kunden ist. Beide Aspekte sind für den Anwender problematisch und von den Herstellern der proprietären Software erwünscht.

### **6.3 Sicherheitsinteressen von proprietären Herstellern und Benutzern**

Jeder Benutzer interessiert sich für eine sichere Software, der er auch vertrauen hat. Es wird von den Herstellern erwartet, dass ihre Produkte den Kundeninteressen entsprechen. In vielen Fällen ist es leider nicht so. Sehr oft hat die Marketingpolitik der proprietären Hersteller eine andere Aufgabe als den Interessen des Benutzers zu dienen.(Anderson 2002) Für ein renommiertes Unternehmen ist es sehr wichtig die Konkurrenten zu eliminieren. Für einen Benutzer bedeutet die Sicherheit, das Abwehren von Hackern, für die Hersteller kann die Sicherheit ein ökonomisches Monopol und sehr hohe Gewinne bedeuten.(Anderson 2002) Für mehrere Kunden bedeutet die Sicherheit die anonyme Nutzung der Anwendungen. Das trifft in Open Source Software zu.(Federrath, Pfitzmann 1999) Die vertraulichen Informationen über den Benutzer sind für die Hersteller der proprietären Software relevant, was natürlich nicht in dem Interesse des Benutzers liegt.(Federrath, Pfitzmann 1999) Die Anhänger der offenen Software behaupten, die bewusste Verwendung von Sicherheitslöchern durch kommerzielle Hersteller. Von dieser Tatsache will auch die US Regierung profitieren.

Die Sicherheitslöcher sollten erst bei ihr gemeldet werden, so dass sie von Geheimdiensten und Strafverfolgungsbehörden ausgenutzt werden können. Nach der Veröffentlichung von Löchern, können die Patches hergestellt werden.(Anderson 2002) Die Entwickler der kommerziellen Software unterstreichen, dass ihre Interessen mit den Benutzerinteressen übereinstimmen.(Microsoft 2002) Wenn man aber die aktuellen Fakten betrachtet, kommt man auf die gegensätzliche Meinung, dass die Hersteller der proprietären Software kaum gemeinsame Interessen mit den Benutzern haben.

## 7. Ergebnis des Referats

### 7.1 Sicherheit

Wir haben uns bemüht zu zeigen, dass Softwaresicherheit von unterschiedlichen Faktoren abhängt, und nicht eines der beiden Entwicklungsmodelle per Definition sicherer ist. Vielmehr muss jede einzelne Software individuell betrachtet werden, um die Sicherheit zu vergleichen. Insbesondere spielen Technik, Interessen, Ressourcen und Qualifikation eine ausschlaggebende Rolle:

- **Interessen:** Für sichere Software ist es nötig, dass schon während der Entwicklung die Sicherheit ein wichtiges Kriterium darstellt, und nicht vernachlässigt wird. Die Interessen der Hersteller und Anwender können unterschiedlich sein. (Payne 2002) Außerdem können staatliche Interessen Hersteller beeinflussen. (Anderson 2002)
- **Technik:** Die Sicherheit von Software ist von ihrer inneren Qualität abhängig, die bei der Entwicklung durch entsprechende softwaretechnische Methoden gesteigert werden kann. (Lutterbeck et al. 2000) (Challet, Le Du 2003)
- **Ressourcen:** Um die Sicherheit einer Software zu erhöhen, müssen bei deren Entwicklung viele Ressourcen, meistens in Form von Entwicklern, eingesetzt werden, wodurch die Entwicklungskosten steigen. (Anderson 2002) (Payne 2002)
- **Qualifikationen:** Software muss von Sicherheitsexperten auf Schwachstellen untersucht werden, da gewöhnliche Entwickler für diese Aufgabe nicht qualifiziert genug sind. (Payne 2002)

Open Source Software bietet elementare Voraussetzungen, diese vier wichtigen Kriterien zu erfüllen.

- **Interessen:** Der Interessenkonflikt zwischen Entwicklern und Anwendern ist minimal. Es besteht kaum eine Abhängigkeit des Benutzers vom Hersteller, und durch den Einblick in den Quellcode kann dieser auf Hintertüren überprüft werden. (Payne 2002)
- **Technik:** Die softwaretechnische Entwicklung steht im Vordergrund und wird nicht durch wirtschaftliche Interessen dominiert. Das Open Source Entwicklungsmodell erfordert es, die Software nach qualitativen Kriterien zu gestalten. (Payne 2002)

- **Ressourcen:** An Open Source Software arbeiten häufig viele freiwillige Entwickler. Daher kann durch Open Source die Anzahl der Entwickler erhöht werden, ohne zusätzliche Kosten zu verursachen.
- **Qualifikationen:** Durch eine hohe Anzahl von Entwicklern erhöht sich die Wahrscheinlichkeit, dass qualifizierte Sicherheitsexperten an der Entwicklung teilnehmen. (Diffie 2003) Insbesondere dieser Punkt ist aber bei jeder Software individuell zu prüfen.

*Open Source Software ist nicht per Definition sicherer, sie bietet aber elementare Voraussetzungen wesentliche, sicherheitsrelevante Voraussetzungen zu erfüllen.*

## 7.2 Literatur

**Tabelle 1: Thematische Zuordnung der Literatur**

<b>These</b>	<b>pro Open Source / kontra proprietär</b>	<b>kontra Open Source / pro proprietär</b>
<b>Probleme propriet. Software</b>	Lutterbeck et al., Diffie, Federrath, Pfitzmann, Anderson	-
<b>Patches</b>	Lutterbeck et al., Payne, Brunstein, Wheeler	-
<b>geheimer Quellcode</b>	Lutterbeck et al., Diffie	Payne, Anderson
<b>Offener Quellcode</b>	Lutterbeck et al., Challet, Le Du	Payne, Needham
<b>Anpassbarkeit</b>	Payne	Microsoft
<b>Herstellerinteressen</b>	Federrath, Pfitzmann, Payne, Anderson	Microsoft
<b>Lock-In</b>	Anderson	Microsoft

Die meisten verwendeten Arbeiten bezogen eher Position für Open Source Software und nur wenige, oftmals zweifelhafte Berichte, vertraten die proprietäre Software. Dieses Ungleichgewicht ist in Tabelle 1 übersichtlich dargestellt.

## 8. Diskussion

Am Ende des Referates stellten wir folgende Punkte zur Diskussion:

- Sind unter sicherheitsrelevanter Betrachtung, kommerzielle Distributoren von Open Source Betriebssystemen (z.B. SuSE, Redhat) mit den Herstellern proprietärer Betriebssysteme vergleichbar? Haben diese Distributoren Interessen, die zu denen der Anwender konträr sind? Besteht für den Anwender eine Abhängigkeit z.B. wegen des Veröffentlichens von Patches?
- Im Gegensatz zu den proprietären Entwicklern ist die Motivation der Open Source Entwickler eine intrinsische, da diese meistens freiwillig teilnehmen. Daher ist die Entwicklerzufriedenheit bei Open Source Software sicherlich höher anzusiedeln als bei proprietärer Software. Kann diese Entwicklerzufriedenheit eine Rolle bei der Softwarequalität und damit bei der Sicherheit spielen?
- Ließen sich durch eine gesetzliche Haftungsregelung für Softwarehersteller die Interessen der Hersteller an die ihrer Benutzer angleichen und damit eine höhere Sicherheit erzielen?
- Nehmen an der Entwicklung von Open Source Software wirklich viele Entwickler teil, oder werden diese Projekte nur von wenigen Entwicklern getragen? Was würde dies für die Sicherheit bedeuten?
- Könnte die Verwendung und Weiterentwicklung des Shared Source Modell von Microsoft dazu führen, den Sicherheitsvorteil zu Gunsten proprietärer Software zu wenden?

## 9. Schluss

In dem Referat und der dazugehörigen, vorliegenden Ausarbeitung wurde zu Beginn die Motivation dargelegt, sich mit der Sicherheit von Open Source Software und proprietärer Software zu befassen und diese miteinander zu vergleichen. Ferner wurden elementare Probleme von proprietärer Software aufgezeigt und unterschiedlichste Aspekte der Systemsicherheit aus Sicht der beiden Entwicklungsmodelle gegenübergestellt. Dabei überzeugte die Open Source Software, so dass wir zu dem Ergebnis kamen: *Open Source Software ist nicht per Definition sicherer, sie bietet aber elementare Vorraussetzungen wesentliche sicherheitsrelevante Vorraussetzungen zu erfüllen.*

## A Literaturverzeichnis

- Anderson 2001** ANDERSON, Ross: Why Information Security is Hard - An Economic Perspective. Paper, first presented at the Applications Security conference in December 2001. 2001. - online: <http://www.cl.cam.ac.uk/ftp/users/ria14/econ.pdf>
- Anderson 2002** ANDERSON, Ross: Security in Open versus Closed Systems - The Dance of Boltzmann, Coase and Moore. Paper, first presented at the "Open Source Software: Economics, Law and Policy" conference in Toulouse (France), 20<sup>th</sup>-21<sup>st</sup> June, 2002. 2002. - online: <http://www.cl.cam.ac.uk/ftp/users/ria14/toulouse.pdf>
- Ausubel 1974:** AUSUBEL, David P.: Psychologie der Unterrichts. Beltz, Weinheim/Basel, 1974.
- Brunnstein 2003** BRUNNSTEIN, Klaus: Gestaltbarkeit und Beherrschbarkeit von Informatiksystemen (GBI). Vorlesung, Universität Hamburg, Fachbereich Informatik, Sommersemester 2003
- CERT<sup>®</sup> Advisory CA-2001-01 2001** CERT Coordination Center: Interbase Server Contains Compiled-in Back Door Account. In: *CERT (2001)*. - *CERT<sup>®</sup> Advisory*, January 10<sup>th</sup>, 2001, online: <http://www.cert.org/advisories/CA-2001-01.html> [January, 6<sup>th</sup>, 2004]
- Challet und Le Du 2003** CHALLET, Damien; LE DU, Yann: Closed Source versus Open Source in a Microscopic Model of Software Bug Dynamics. Paper, first presented at arXiv.org e-Print archive on June 19<sup>th</sup>, 2003 (version 1) and latest released on July 25<sup>th</sup>, 2003 (version 3). 2003. - online: <http://arxiv.org/abs/cond-mat/0306511>
- Diffie 2003** DIFFIE, Whitfield: Proprietär vs. Open Source: Eine Frage der Sicherheit. In: *ZDNet (2003)*. - *ZDNet IT-Manager/Enterprise Tech*, February 7<sup>th</sup>, 2003, online: <http://www.zdnet.de/itmanager/tech/0,39023442,2130055,00.htm>
- Federrath und Pfitzmann 1999** FEDERRATH, Hannes; PFITZMANN, Andreas: Stand der Sicherheitstechnik. In: KUBICEK et al (Hrsg): „*Multimedia@Verwaltung*“, *Jahrbuch Telekommunikation und Gesellschaft 1999*, Hüthig, S. 124-132. 1999. - Universität Regensburg, April 2003, online: [http://www-sec.uni-regensburg.de/publ/1999/FePf\\_99JahrbTK.pdf](http://www-sec.uni-regensburg.de/publ/1999/FePf_99JahrbTK.pdf)
- Grassmuck 2002:** GRASSMUCK, Volker: Freie Software. Zwischen Privat- und Gemeineigentum. Bundeszentrale für politische Bildung, Bonn, 2002.
- Herold, Landherr, 2001:** HEROLD, LANDHERR: SOL, Selbstorganisiertes Lernen, ein Systemischer Ansatz für Unterricht. 2001.
- Hohmann 2003** HOHMANN, Luke: Beyond Software Architecture - Creating and Sustaining Winning Solutions. Addison-Wesley Longman (1. Februar 2003), Amsterdam, 2003. - ISBN 0-201-77594-8
- Kettmann, Glave 1999** KETTMANN, Steve; GLAVE, James: MS Denies Windows 'Spy Key' In: *Wired News*, 1999, online: <http://www.wired.com/news/technology/0,1282,21577,00.html> [January, 6<sup>th</sup>, 2004]
- Lau 2003** LAU, Oliver: GnuPG 1.2.4 räumt mit fehlerhaften ElGamal-Code auf. In: *heise online (2003)*. - *heise newsticker*, December 28<sup>th</sup>, 2003, online: <http://www.heise.de/newsticker/data/ola-28.12.03-004/> [January, 6<sup>th</sup>, 2004]

- Leyden 2001** LEYDEN, John: Code Red bug hits Microsoft security update site. In: *The Register (2001)*. - *The Register Virus News*, July 20<sup>th</sup>, 2001, online: <http://www.theregister.co.uk/content/56/20545.html> [January, 6<sup>th</sup>, 2004]
- Lutterbeck et al. 2000** LUTTERBECK, Bernd; GEHRING, Robert; HORNS, Axel H.: Sicherheit in der Informationstechnologie und Patentschutz für Softwareprodukte - Ein Widerspruch?. Kurzgutachten, erstellt im Auftrag des Bundesministeriums für Wirtschaft und Technologie, vorgelegt von der Forschungsgruppe Internet Governance Berlin, Dezember 2000, Kap. 6, S. 110-127. 2000. - online: <http://ig.cs.tu-berlin.de/forschung/IPR/LutterbeckHornsGehring-KurzgutachtenSoftwarePatente-122000.pdf>
- Microsoft 2002** MICROSOFT Corporation: Betriebssysteme im Unternehmen: Microsoft® Windows und OSS. Zusammenhänge und Investitionsfolgen. Wettbewerbspapier Microsoft, Microsoft Deutschland, Stand Februar 2002. 2002. - online: <http://www.microsoft.com/germany/library/resourcesmod/mswinoss.pdf>
- Microsoft 2003** MICROSOFT Corporation: Open Source versus Open Standards. Gegensätzliche Konzepte und die Bedeutung von Interoperabilität. Microsoft Deutschland, März 2003. - online: [http://www.microsoft.com/germany/library/resourcesmod/open\\_source\\_vs\\_open\\_standards.pdf](http://www.microsoft.com/germany/library/resourcesmod/open_source_vs_open_standards.pdf)
- Moody 2001**: MOODY, Glyn: Die Software-Rebellen. Die Erfolgsstory von Linus Torvalds und Linux. Linux New Media AG, München 2001.
- Needham 2002** NEEDHAM, Roger: Security and Open Source. In: *Microsoft Japan (2002)* – Microsoft Japan, Shared Source / Security, May 13<sup>th</sup>, 2002, online: <http://www.microsoft.com/japan/sharedsource/Security/Needham.msp>
- Payne 2002** PAYNE, Christian: On the security of open source software. In: *Information Systems Journal*, January 2002, Vol. 12, Issue 1, S.61.
- Poulsen 2003** POULSEN, Kevin, SecurityFocus: Linux kernel backdoor blocked. In: *The Register (2003)*. - *The Register Security*, November 7<sup>th</sup>, 2003, online: <http://www.theregister.co.uk/content/55/33855.html> [January, 6<sup>th</sup>, 2004]
- Wahl et al. 1991**: WAHL, D. et al.: Erwachsenenbildung konkret. Beltz, Weinheim, 1991.
- Wheeler 2003** WHEELER, David A.: Why Open Source Software / Free Software (OSS/FS)? Look at the Numbers!. September 2003. 2003. - online: [http://www.dwheeler.com/oss\\_fs\\_whv.html](http://www.dwheeler.com/oss_fs_whv.html)
- Xbox Linux Project 2003** Xbox Linux Project: MechInstaller released!. In: *Xbox Linux Project Sourceforge Homepage (2003)*. – *Xbox Linux Project News Archive*, August 11<sup>th</sup>, 2003, online: [http://xbox-linux.sourceforge.net/news\\_archive.php](http://xbox-linux.sourceforge.net/news_archive.php) [January, 6<sup>th</sup>, 2004]

Wenn nicht anders angegeben, wurden die URL's am 14. Dezember 2003 überprüft.