Open Source Software — Sicherheit im Spannungsfeld von Ökonomie und Politik

Dipl.–Inform. Robert A. Gehring

Technische Universität Berlin
Informatik und Gesellschaft

—DRAFT—
Version 0.8

Zusammenfassung Ob 'Open Source'–Software 'sicherer' bzw. 'zuverlässiger' als Software ist, die als proprietäre Software entwickelt wurde, ist durchaus umstritten. Oft arten die Diskussionen darüber in verbale 'Glaubenskriege' aus. Versucht man, die Faktenlage zu evaluieren, stellt man sehr schnell fest, daß es weitgehend an aussagekräftigen Daten fehlt. Der hier dokumentierte Vortrag[1] befaßt sich aus unterschiedlichen Perspektiven mit Aspekten der Sicherheit von 'Open Source'–Software.

Einige Beispiele aus der Praxis zeigen zu Beginn, inwiefern die mit dem Etikett 'Open Source' versehene Methode einen *nachweisbaren* Beitrag zur Verbesserung der IT–Sicherheit geleistet hat.

Im zweiten Teil des Vortrages wird auf die ökonomischen Randbedingungen der Software–Entwicklung eingegangen und diskutiert, inwiefern diese Auswirkungen auf den Entwicklungsprozeß selbst haben. Da der 'Open Source'–Entwicklungsprozeß anderen ökonomischen Randbedingungen unterworfen ist, als die proprietäre Softwareentwicklung, könnten sich daraus erfolgversprechende Ansätze für die Verbesserung von Softwarequalität und IT–Sicherheit ergeben.

Im dritten Teil schließlich wird in aller Kürze darauf eingegangen, inwiefern der TCPA–Ansatz für 'trusted computing' im Verbund mit dem Einsatz von 'Open Source'–Software geeignet ist –oder nicht geeignet– ist, die Sicherheit und Zuverlässig von IT–Systemen zu erhöhen.

Inhaltsverzeichnis

1 Einleitung

Sicherheit von Informationstechnologie scheint zumindest für die Apologeten und Antagonisten von ‘Open Source Software’[2] weitgehend Glaubenssache zu sein. Jedenfalls kann man diesen Eindruck gewinnen, wenn man die marktgängige Fachpresse — online und offline — zur Meinungsbildung heranzieht. Die sicherheitsbezogenen Meinungsäußerungen haben nichtzuletzt deswegen Konjunktur, weil klare Untersuchungen mit ‘harten’ Resultaten Seltenheitswert haben und Sicherheit vorrangig ein Marketinginstrument im Kampf «[M]ankind vs. Microsoft», um die Worte des SUN–Chefs Scott McNealy zu gebrauchen (Chai und Gao 2003), geworden zu sein scheint.

Vorliegender Aufsatz[3] verfolgt zwei Ziele. Zum Ersten wird ein Beispiel diskutiert, das die Frage nach der Sicherheit durch ‘Open Source’ unter bestimmten politischen Randbedingungen beantwortet, ein Aspekt der regelmäßig übersehen wird. Anschließend werden einige empirische Untersuchungen vorgestellt, die nach meiner Meinung methodisch hinreichend wissenschaftlich abgesichert sind, um einen partiellen Einblick in die Sicherheit und Qualität (was bekanntlich oftmals zusammenhängt) von ‘Open Source Software’ zu gestatten.

Der zweite Teil widmet sich den ökonomischen Randbedingungen der Softwareentwicklung und fragt nach deren Konsequenzen für die Sicherheit der so entstehenden Produkte. Dabei greife ich auf frühere Arbeiten zurück.[4] Zieht man ökonomische Faktoren als relevant in Betracht, kann man auch beim neuen Allheilmittel der IT–Sicherheit, der von der TCP Alliance propagierten ‘trusted computing platform’ nach deren ökonomischen Fundamenten fragen. Im vorliegenden Aufsatz stelle ich diese Frage im Hinblick auf ‘Open Source Software’. In Anbetracht der klaren pro ‘Open Source’–Positionierung des TCPA–Vorreiters IBM kann man diese Frage wohl nicht früh genug stellen. . .

2 Empirische Beispiele für Sicherheit mit/von/durch ‘Open Source’

Ob ‘Open Source’–Software ‘sicherer’ bzw. ‘zuverlässiger’ als Software ist, die als proprietäre Software entwickelt wurde, ist durchaus umstritten und der Streit wird zum Teil mit einer Art religiösen Eifer geführt. Ist IT–Sicherheitheit durch Open Source also eine Glaubenssache? Das sollte sie wohl nicht sein. Doch wie gewinnt man Gewißheit, welche Argumente stichhaltig sind? Wer kann zuverlässige Auskunft geben?

Wie bei vielen anderen Streitfragen könnte ein Blick in die Praxis Argumente liefern, die uns weiterhelfen. Im folgenden wird deshalb der Frage nach der Sicherheit mit/von/durch ‘Open Source Software’ in einem ersten Annäherungsversuch empirisch nachgegangen.

2.1 Open Source & Kryptographie

Open Source–Entwicklungen im Bereich der Kryptographie haben starke Verschlüsselungsverfahren weltweit verfügbar gemacht und, quasi als “Seiteneffekt” einen Wandel in der US–Kryptopolitik bewirkt.

2.1.1 Zur Erinnerung: Die Kryptodebatte

Das erste Beispiel weist starke Bezüge zu politischen Entwicklungen der vergangenen Jahre auf. Einige Leser werden sich noch der sogenannten 'Kryptodebatte'[5] der 90'er Jahre erinnern. Im Kern ging es um die Frage, ob der Staat es seinen Bürgern gestatten dürfe, starke Verschlüsselungsverfahren nach Belieben zu nutzen, und/oder ob eine Regulierung etwa in Form einer Schlüsselhinterlegung oder von Genehmigungsverfahren zwingend notwendig sei. Die Argumente gingen hin und her und die Staaten verfolgten durchaus unterschiedliche Wege. In den USA etwa unterlag der Export von Verschlüsselungsverfahren strikten Exportbeschränkungen. Verschlüsselungsverfahren wurden als 'Munition' eingestuft und nur nach umständlichen Genehmigungsprozeduren exportiert werden. Es kam sogar zu einigen Gerichtsverfahren[6]:

- Der Entwickler der Software PGP, Philip Zimmermann, wurde des illegalen Exportes von Verschlüsselungssoftware angeklagt, jedoch schließlich freigesprochen.
- Der U.S.–Wissenschaftler Daniel Bernstein hatte 1995 die U.S.–Regierung verklagt, da er wegen der Exportbeschränkungen im Kryptobereich den Quellcode seiner Verschlüsselungssoftware nicht im Internet verfügbar machen durfte. Darin sah er eine Einschränkung der von der U.S.–Verfassung garantierten Redefreiheit, wollte er doch den Quelltext für Zwecke der Lehre seinen Studenten zur Verfügung stellen — wie andere Texte auch. Bernstein bekam 1997 in erster und 1999 in zweiter Instanz Recht. Beide Richterinnen befanden das Krypto–Exportverbot insofern für verfassungswidrig als es auf Software im Quellcode angewandt wurde.

Auf Dauer ließen sich die Exportbestimmungen im SOftwarebereich nicht durchsetzen. Die 'Open Source'–Community spielte dabei neben den Gerichtsprozessen zu Fragen der Redefreiheit[7] eine Schlüsselrolle, wie wir im folgenden sehen werden.[8]

2.1.2 SSLeay/OpenSSL

In Australien fand eine Entwicklung im Bereich der Kryptographie statt, die den Stein zur Deregulierung im Kryptobereich entscheidend ins Rollen gebracht hat. Eric A. Young begann 1995 mit der Entwicklung einer Softwarebibliothek namens SSLeay, die das von Netscape entwickelte SSL/TLS (s.u.) preiswert und ohne die US–Exportbeschränkungen für Verschlüsselungssoftware zur Verfügung stellen sollte (Hudson und Young 1998). Sein eigentlicher Antrieb war, seine Programmierfähigkeiten an einer praktischen Aufgabe auszuprobieren. Young wurde dabei sehr bald von Tim J. Hudson unterstützt. SSLEay wurde mit einer 'Open Source'–Lizenz versehen und im Internet veröffentlicht wurde.[9] Es fanden sich weitere Entwickler, die Code beisteuerten und Portierungen für diverse Softwareplattformen und –pakete vornahmen. Inzwischen haben Young und Hudson die aktive Weiterentwicklung aufgegeben[10] und diese Aufgabe ist auf die OpenSSL–Entwickler[11] übergegangen.

OpenSSL stellt mittlerweile alle notwendigen kryptographischen Verfahren, d.h. Verschlüsselungsverfahren, Schlüsselverwaltung, Zertifikatsverwaltung, Hashverfahren usw., zur Implementierung von Secure

Socket Layer (SSL Versionen 2 und 3) und Transport Layer Security (TLS Version 1) zur Verfügung. Sowohl SSL als auch TLS kommen zur Absicherung der Kommunikation von Webservern und Webbrowsern (https–Protokoll) zum Einsatz, sind jedoch nicht darauf beschränkt.

2.1.3 Gesinnungswandel der U.S.–Regierung

Die Implikationen der weltweiten Verfügbarkeit von starken Verschlüsselungsverfahren als Open Source haben sich auf gar nicht so lange Sicht als gravierend erwiesen.

Wie Solveig Singleton in einem Bericht für das konservative Cato–Institut in den USA feststellte, waren durch die SSLeay–Entwicklung in Australien die U.S.–Exportbeschränkungen im Prinzip sinnlos geworden.

> «Export controls can be used to stop hard-to-transport items like missiles or military planes from leaving the country. But they cannot stop the spread of a few lines of of code (an encryption program can be contained in as few as three lines), technology that can be transported instantaneously over phone lines at almost no cost. Nor can they stop the movement of capital abroad to software developers located in other countries. [. . .] Encryption using products like SSLeay, SSL source code available free from a web site in Australia, enables the creation of strong encryption products from weaker prod-ucts. Stronghold, a UK product, combines SSLeay with Apache, a leading Web server in the public domain, to create a 128 bit Web server.» Singleton (1998, S. 22 f.)

Exportbeschränkungen im Bereich der Softwarekryptographie führten nicht dazu, die Ausbreitung starker Kryptographie verhindern zu können. Stattdessen mußten U.S.–Unternehmen zusehen, wie Konkurrenten in Ländern mit einer liberaleren Kryptopolitik die Nachfrage eines Marktes bedienten, der ihnen selbst verschlossen blieb. Irgendwann erkannte das auch die U.S.–Administration und schließlich hob U.S.–Präsident Bill Clinton im September 1999 die Exportbeschränkungen für starke Verschlüsselungsverfahren weitgehend auf.[12]

Whitfield Diffie, einer der zivilen Erfinder der asymmetrischen Kryptographie, stellt zusammen mit Susan Landau in der Bewertung der Ereignisse Open Source als eine der treibenden Kräfte hinter dem grundlegenden Wandel der US–Kryptopolitik am Ende des 20. Jahrhunderts dar.[13]

> «What forces drove the U.S. Government from complete intransigence to virtually complete capitulation in under a decade? Most conspicuous is the Internet, which created a demand for cryptography that could not be ignored and at the same time made it more difficult than ever to control the movement of information but more subtle forces were also at play. One of these was the *open–source* movement.» Diffie und Landau (2001, S. 14 f.)

2.1.4 Diskussion

Als Ergebnis (a) einer Debatte um den Quellcode von Software und (b) einer Open Source–Entwicklung ist starke Kryptographie nunmehr weltweit verfügbar und hat entscheidend dazu beigetragen, die Kommunikation zwischen Webservern und Webbrowsern sicherer zu machen. Die “Macht des Faktischen” hat die U.S.–Kryptopolitik ad absurdum geführt und die U.S.–Administration zum Umdenken bewegt. “Sicherheit durch Open Source” ist zumindest in diesem Beispiel belegt.

2.2 Sicherheit von Webservern

Im zweiten Beispiel geht es um die Sicherheit zweier konkurrierender Produkte im Segment der Webserver von denen eines als ‘Open Source’–Software entwickelt wird, das andere klassische als proprietäre Software.

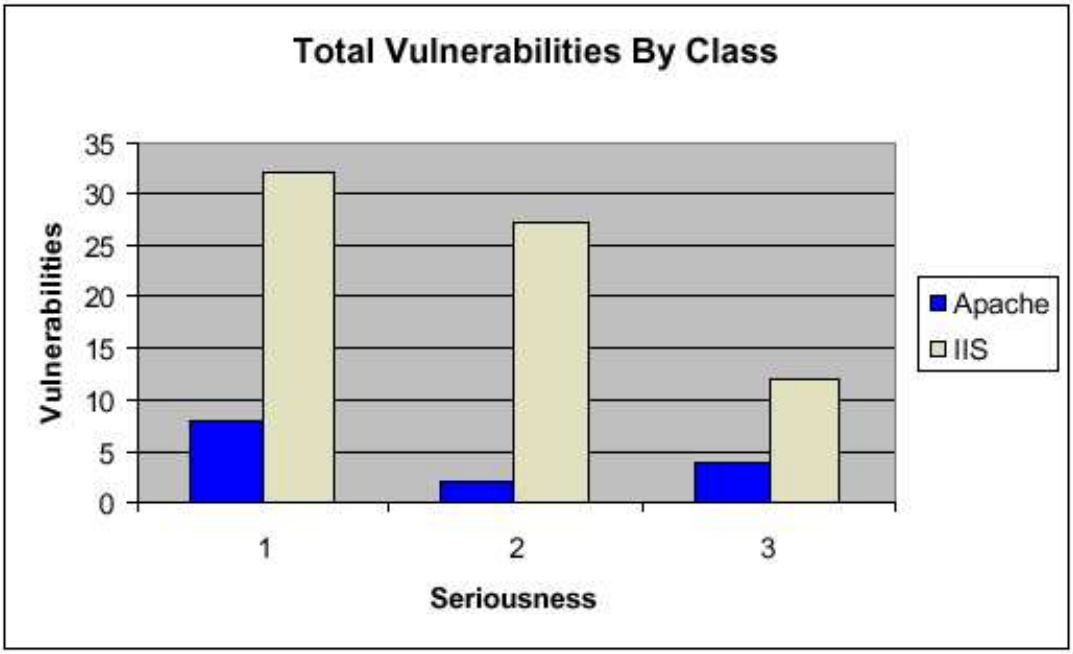
2.2.1 Apache vs. MS Internet Information Server

Ritchey (2001) hat die Sicherheit eines der umfangreichsten ‘Open Source’–Softwarepakete untersucht und mit einem unmittelbaren proprietären Konkurrenzprodukt verglichen: ‘Apache’–Webserver (OSS) vs. ‘Microsoft Internet Information Server’ (closed source). Dazu hat er auf veröffentlichte Daten der auf Sicherheitsfragen spezialisierten Website SecurityFocus zurückgegeriffen.

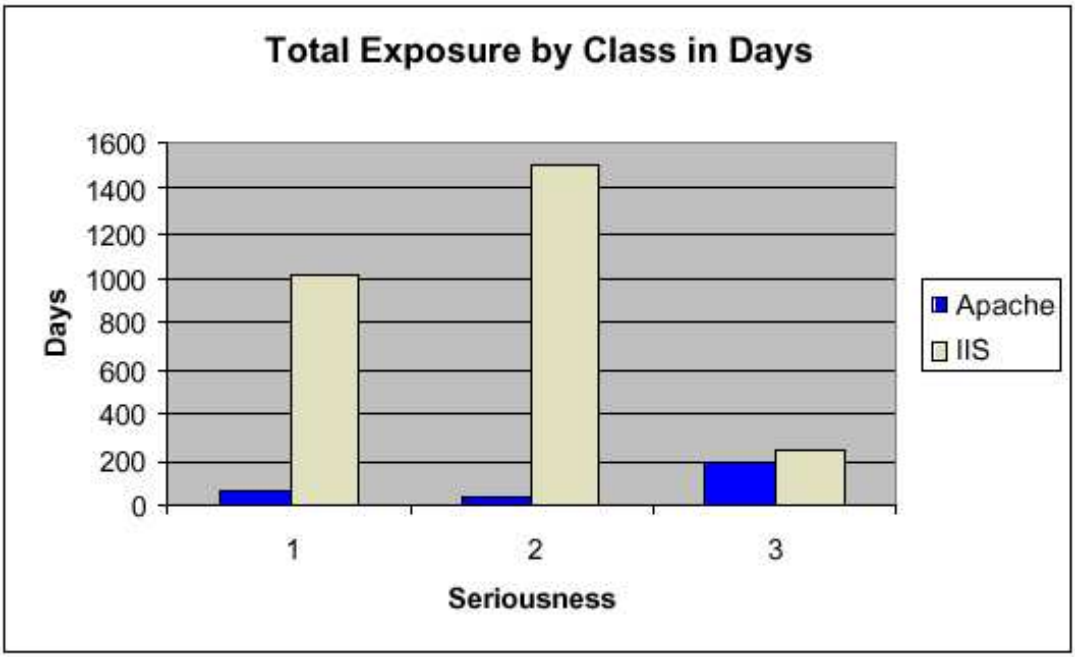
Der gesamte Datenbestand für die Jahre 1996–2001 wurde dahingehend untersucht, ob und welche Hinweise auf Sicherheitsprobleme der genannten Softwarepakete gespeichert sind. Je nach Schwere wurden die dokumentierten Sicherheitslücken in eine von drei Klassen (1 = hohes Risiko, 2 = mittleres R., 3 = geringes R.) eingeteilt. Die gewonnenen Informationen wurden hinsichtlich zweier Kriterien untersucht:

- Zeitverzögerung bis zur Beseitigung des Sicherheitsproblems (‘exposure time’, S.2)
- Schwere des Sicherheitsproblems (’severity’, S.3).

Quelle: Richey (2001, S. 6)



Quelle: Richey (2001, S. 6)



Betrachtet man Ritcheys Daten im Detail, fällt die große Schwankungsbreite bezüglich der Sicherheitsrisiken auf. In manchen Jahren scheint es keine, in anderen Jahren erheblich viele Vorfälle gegeben zu haben. Eine einleuchtende Erklärung gibt es dafür nicht.

2.2.2 Ergebnis der Untersuchung

Um zu einem Urteil zu gelangen, aggregiert Ritchey die Daten über den gesamten Untersuchungszeitraum und stellt fest:

> «Apache is the clear winner with a significantly smaller number of and average exposure to vulnerabilities. For the most severe vulnerabilities, Apache had roughly 4 times less security exposure than IIS. Apache did nearly as well on category two vulnerabilities, though it did not do as well on the minor vulnerabilities. IIS had approximately 1/2 the exposure to minor vulnerabilities than Apache. For all classes of vulnerabilities, Apache's vulnerability exposures lasted 1/2 the time of IIS's.» Ritchey (2001, S. ???)

Ritcheys Vergleich der Webserver fällt eindeutig zugunsten des nach dem 'Open Source'–Modell entwickelten Apache und zuungunsten des proprietären Microsoft Internet Information Servers aus. Anscheinend werden die weitverbreiteten Auffassungen, daß 'Open Source Software'

1. weniger sicherheitskritische Fehler aufweist und
2. entdeckte Fehler schneller beseitigt werden

durch das Apache–Beispiel bestätigt.

2.2.3 Diskussion

Das Beispiel des Apache belegt, daß Open Source Software sicherer als oder mindestens genauso sicher wie proprietäre Closed Source Software sein *kann.* Eine Verallgemeinerung kann man daraus pauschal sicher nicht ableiten.

Apache könnte aufgrund seiner Verbreitung als Beleg für Raymonds (1998) These ("Given enough eyeballs, all bugs are shallow.") herhalten. Bei den vielen Apache–Anwendern könnten tatsächlich "enough eyeballs" zusammenkommen. Im Falle weniger verbreiteter Open Source–Applikationen muß dasselbe nicht gelten.

Zu den quantitativen und qualitativen Feststellungen Ritcheys kommt hinzu, daß der Apache Webserver deutlich weiter verbreitet ist als der Internet Information Server und insofern ein 'größeres' Ziel für Angreifer bietet, die Schwachstellen ausmachen und –nutzen wollen.[14]

Damit, daß der Apache selten in der Standardkonfiguration des Apache–Konsortiums eingesetzt wird und insofern die Hürde für potentielle Angreifer höher liegt als bei einem uniform verbreiteten IIS, kann die geringere Fehler–/Verletzungsanfälligkeit nicht wirklich begründet werden, denn die Existenz des Fehlers in einer der vielen verbreiteten Versionen genügt, um in die SecurityFocus–Statistik einzugehen. Im Aggregat würde nicht mehr erkennbar sein, daß es sich um unterschiedliche Versionen handelt. Die weite Verbreitung von Derivaten verringert insofern nicht automatisch die Verletzlichkeit.

Es scheint vielmehr so zu sein. daß im Falle eines erfolgreichen Angriffs der Schaden eingedämmt wird. Ein erfolgreicher ‘exploit’ für die von RedHat gelieferten Apache–Version wird nicht ohne weiteres auf die Apache–Version von SuSE oder eine selbst kompilierte Version anwendbar sein, da ‘exploits’ in der Regel darauf angewiesen sind, die auszunutzende Schwachstelle an einer bestimmten Adresse im Binärcode vorzufinden. Jedwede Modifikation des Quellcodes mit anschließender Kompilation, sei es durch den Distributor oder sei es durch den Endanwender, führt zu einer gewissen Varianz im Hinblick auf den Binärcode, die Angriffe signifikant erschwert.

Im Unterschied dazu wird der IIS in der Regel in einem Format mit identischem Binärcode auf Millionen von Computern in identischer Konfiguration ausgeliefert, was die Lokalisierung etwaiger Schwachstellen erleichtert. Damit ließe sich dessen höhere Anfälligkeit erklären.

Hinweis: Mittlerweile hat das Bundesamt für Sicherheit in der Informationstechnik zwei Berichte zur Sicherheit von Apache und IIS herausgegeben, deren Ergebnisse hier noch nicht berücksichtigt wurden.[15]

2.3 Open Source Code–Qualität am Beispiel TCP/IP

Die Sicherheit von Softwarepaket hängt u.a. auch von der Qualität des zugrundeliegenden Quellcodes ab. Typischerweise gefährden sogenannte ‘buffer–overflows’ und ‘memory leaks’ die Stabilität eines Systems und können darüberhinaus ein Einfallstor für bösartigen Code bilden. Die absolute Menge derartiger Schwachstellen im Code hat entscheidenden Einfluß auf die Qualität und Sicherheit des daraus erzeugten Binärprogrammes. Die Frage nach der Qualität (im Hinblick auf Sicherheit) von Open Source *Code* im Vergleich zu Closed Source Code ist daher von Interesse.

2.3.1 TCP/IP–Qualität(en)

Reasoning, eine Unternehmensberatung mit Schwerpunkt Software–Inspektion, ist der Frage nachgegangen, wie um die Qualität von Open Source Software im Vergleich zu proprietärem Code steht (Shankland 2003). Um die Vergleichbarkeit zu gewährleisten wurden unterschiedliche Implementationen des TCP/IP–Stacks, d.h. derjenigen Softwarebestandteile die in den Betriebssystemen für die Internetkommunikation zuständig sind, untersucht:

> «Five implementations of TCP/IP from a variety of commercial systems, both general purpose operating systems and embedded applications, were inspected. Reasoning also inspected the TCP/IP code in version 2.4.19 of the Linux kernel.» Reasoning (2003, S. 2)

Die Befunde waren durchaus überraschend.

Ausweislich des Untersuchungsberichts[16] wurden 817 Quelltextdateien aus der Linux 2.4.19–Implementierung mit insgesamt 125.502 Zeilen Code (LOC) untersucht. Dabei wurden insgesamt 8 Defekte folgender Zusammensetzung ermittelt:

- Memory Leak: 1[17]
- NULL Pointer Dereference: 3
- Bad Deallocation: 0
- Out of Bounds Array Access: 3
- Uninitialized Variable: 1

Eine qualitative Bewertung ergibt, daß nur die Hälfte der ermittelten Fehler Auswirkungen auf die Systemstabilität und ggf. Sicherheit haben.

> «In summary: one defect is real, 4 defects are not real, and 3 are undecided.» Reasoning (2003, S.10)

Vergleicht man diese Werte mit anderen TCP/IP–Implementationen, so ergibt sich das Bild, daß die Open Source Code–Qualität sehr gut abschneidet.

> «Based solely on quantitative criteria used in all of Reasoning's inspection projects, the open source implementation of TCP/IP in the Linux operating system exhibited a defect density of 0.10 defects per thousand lines of source code (Defects/KLSC), while the composite defect density of the five commercial projects was 0.55. In its most recent analysis of 200 commercial projects totaling 35 million lines of source code, Reasoning determined that 33% had defect densities below 0.36; 33% had defect densities between 0.36 and 0.71, and the remaining 33% had defect densities above 0.71. Thus, the TCP/IP implementation in the Linux operating system ranks in the upper third, while the composite code quality of the commercial implementations ranks in the middle third.» Reasoning (2003, S. 3).

Untersucht man die Ergebnisse genauer, so fällt die Einschätzung für den Linux–Code im Grunde sogar noch besser aus, als es die Zuordnung zum 'besten Drittel' erkennen läßt.

> «[T]he most reliable metric is which defects need to be fixed according to the developers or maintainers of the code. . . . On average, both the reported and the repaired defect densities are higher for the commercial implementations compared to the open source implementation.» Reasoning (2003, S. 12 f.)

2.3.2 Diskussion

Aus der Reasoning–Studie geht hervor, daß mit der Open Source–Entwicklungsmethode eine Code–Qualität erreichbar ist, die proprietär entwickelte Produkte hinter sich zu lassen geeignet ist.

Auch hier ist jedoch zu differenzieren. Einerseits ist der TCP/IP–Code schon verhältnismäßig lange ‘im Umlauf’ und in der Entwicklung, so daß er als ausgereifter gelten darf als ‘frischer Code’ oder solcher mit geringer Verbreitung. In dieser Hinsicht, d.h. was das Reifen von Code angeht, unterscheidet sich Open Source–Code wohl nicht grundsätzlich von proprietärem Code.

Interessanter ist dagegen, daß die Open Source Code–Qualität im selben Bereich liegt wie die Qualität des proprietären Codes für ‘Embedded Systems’. Dort wird diese Qualität durch wesentlich rigorosere Entwicklungsverfahren und Testverfahren (oft in Form formaler Verfahren) erreicht als beispielsweise bei der Entwicklung von handelsüblicher PC–Standardsoftware. Wenn Open Source Code damit qualitativ gleichziehen kann, wirft das m.E. Fragen hinsichtlich der oft betonten Überlegenheit der formalen Entwicklungsmethoden auf, jedenfalls in der Praxis. Eine Antwort habe ich an dieser Stelle jedoch nicht zu bieten, ich sehe vielmehr erheblichen Forschungsbedarf.

2.4 Zusammenfassung der Beispiele

Aus den drei hier vorgestellten Beispielen geht hervor, daß man Sicherheit und Qualität von Software nicht als eine Glaubenssache behandeln muß. Es gibt —jenseits theoretischer Diskussion— Möglichkeiten, objektive Kriterien aufzustellen und *empirisch* zu prüfen. Geht man dabei methodisch vor, erhält man bemerkenswerte Resultate, die jedenfalls prima facie für das Argument zu sprechen scheinen, daß Software die mit der Open Source–Methode entwickelt wird, ‘besser’ ist (im Hinblick auf Qualität und Sicherheit) als proprietär entwickelte Software. Eine solche Aussage ist immer cum grano salis zu bewerten: Die hier vorgestellten Beispiele zeichnen sich alle durch ihren hohen Verbreitungsgrad aus. Zu Software mit geringerer Verbreitung liegen vergleichbar belastbare Erkenntnisse derzeit nicht vor — weder für Open Source Software noch für prorietäre, Closed Source Software.

Das letzte Wort soll an dieser Stelle dem CEO von Reasoning, Scott Trappe, gehören:

> «One bright spot in the industry, said Trappe, is open–source development, with its emphasis on people reading each other’s code. The potential embarrassment of public ridicule is probably a powerful motivation to get things right–or even better, to write code that’s impressively craftsmanlike, rather than merely good enough. "There aren’t enough data points yet to be statistically significant, but what we’ve seen so far suggests that open source code may have much lower defect density than commercial software–about 1 defect per 10,000 lines, about three times better than we typically see," Trappe estimated.» Coffee (2002)

2.5 Ausblick

Es gibt etliche weitere Forschungsberichte im Bereich der Open Source Sicherheit die Aufmerksamkeit verdienen (Chou u. a. 2001; Wright u. a. 2002; Pourzandi u. a. 2002). Nur durch gründliche Untersuchung und Analyse wird man letztlich die Stufe des “Glaubenskrieges” verlassen und zu einer angemessenen, wissenschaftlichen Beurteilung kommen.

3 Ökonomie der Software–Entwicklung: Was spricht für ‘Open Source’?

Im zweiten Teil meiner Betrachtungen möchte ich wieder theoretisch werden und einige Überlegungen zur Ökonomie der Software anstellen. In diesen Bereich fallen sowohl Fragen der Ökonomie der Softwareentwicklng, der –anwendung als auch der –wartung.

3.1 Ein profitables Geschäft

Jeder Anbieter von Produkten für den Massenmarkt möchte Profit machen, Software bildet da keine Ausnahme. Absatz finden Produkte nur, wenn sich Käufer finden. Die Kaufnachfrage beschränkt die mögliche Höhe des Preises und damit auch der Profite.

Nun sind Testen und Fehlerbeseitigung zeit– und personal–, d.h. kostenintensiv. Je umfangreicher die Testprozeduren ausfallen, desto höher werden die Fixkosten für das Produkt ausfallen müssen. Nimmt man an, daß der Marktpreis durch die Nachfrage begrenzt ist, bedeutet das einen Profitverlust für den Anbieter. Im Umkehrschluß ergibt sich ein höherer Profit, wenn bei Testen und Debugging Kosten eingespart werden.

Unterstellt man dem Anbieter ökonomische Rationalität, so wird dieser bestrebt sein, die Kosten einzusparen und mittelbar auf die Kunden abzuwälzen. Solange dem nicht andere, komplementäre Kostenfaktoren etwa aus Haftungsansprüchen entgegenstehen, wird diese Strategie erfolgreich sein. Untersucht man, wie es in der Praxis um die Haftung bei Softwarefehlern bestellt ist, fällt das Ergebnis ernüchternd aus. Haftung spielt im Softwaremarkt für Endverbraucher keine nennenswerte Rolle, darauf können sich die Hersteller sowohl in den USA als auch in Europa verlassen.

> «The reality is that the amount of software quality offered is market driven. The top suppliers of reusable commercial software have determind how to maximize profit with minimal acceptable quality.» Kotyk Vossler und Voas (2000, S. 483)

In Zukunft, daran arbeiten die Lobbyisten intensiv, werden die Haftungslücken noch größer sein als heute schon (Vossler und Voas 2000).

Wer nicht haften muß spart bares Geld, wenn er unreife Software auf den Markt bringt. Und in der Tat können wir das entsprechende Verhalten in der Praxis eher häufiger als seltener beobachten.

> «[T]he huge push to deploy software at breakneck speeds in Internet time leads to software systems being released and implemented with inadequate testing. Some software companies view problem reporting and bug fixing as software testing.» Pipkin (2000, S. 75)

3.1.1 Service als knappes Gut

Service ist ein weiterer Problemfaktor. Setzt man die Millionen Kunden zur Mitarbeiterzahl der Softwarehersteller ins Verhältnis, wird klar, daß Service ein knappes Gut ist. Den Marktgesetzen folgend wird dieses Gut zu einem hohen Preis gehandelt werden, den nur eine Minderheit der Kunden wird bezahlen könnnen. Die Majorität wird mit einem sehr begrenzten Service auskommen und ihre Qualitätsansprüche zurückschrauben müssen. Insofern Sicherheit als Qualitätsmerkmal wahrgenommen wird, kann nur ein begrenztes Maß an Sicherheitsdienstleistungen eingekauft werden.[18]

3.1.2 Service als Geschäftsmodell

Nun könnte man einwenden, der Mangel an Service sei ein besonderes Problem des Massenmarktes. Das täsucht allerdings. Wirft man einen Blick auf das zweite wichtige Geschäftsmodell im Softwarebereich, auf die Anbieter kundenspezifischer Software, fallen die Befunde anders, aber nicht besser aus. Das Geschäftsmodell der Anbieter in diesem Bereich basiert zu wesentlichen Teilen auf dem Verkauf von teuren Serviceleistungen. Fehlerhafte, unsichere Software macht Serviceleistungen notwendig. Je mehr Service verkauft wird, desto höher die fallen die Profite aus . . .

Ohne (falsche) Larmoyanz kann man schlicht feststellen, daß sich die Anbieter eben ökonomisch verhalten. Das ist es auch, was in der Marktwirtschaft von ihnen verlangt wird und wofür sie belohnt werden.

3.2 Software–Käufer

Damit ist die Frage zu stellen, warum denn die Kunden sich nicht dagegen wehren, warum sie dieses ‘Spiel mitspielen’ und sich als als kostenlose Tester zur Verfügung stellen. Warum sagen sie nicht einfach ‘nein’ zu unausgereifter, fehlerhafter und unsicherer Software?

Ich lasse an dieser Stelle diejenigen, die sich mittlerweile genau dazu entschieden haben und etwa auf Open Source Software ‘umgestiegen’ sind der Einfachheit einmal außer Acht. Als Rechtfertigung mag dienen, daß Microsoft–Betriebssysteme auf Arbeitsplatzrechnern immer noch um die 90% Marktanteil aufzuweisen haben.

Warum also bleibt die große Abkehr der Kunden aus? Das erklärt sich an erster Stelle mit den Informationsasymmetrien im Markt für Softwareprodukte. Software ist ein Erfahrungsgut und das hat Konsequenzen für die Informationsverteilung zwischen den Marktteilnehmern.

- Die Softwareanbieter wissen um die Qualität und Sicherheit ihrer Produkte, bevor sie dieselben verkaufen.

- Softwareanwender können die Software erst beurteilen, nachdem sie sie installiert und damit gearbeitet haben.

> «Even if consumers are willing to pay for more secure systems, choosing a system based on its security properties is difficult. This is not a failing of the consumer, as even industry experts rarely have little more than crude heuristics available to them to compare the security of competing products.» Schechter (2002, S. 1)

Kaufentscheidungen können daher *in der Regel* nicht auf objektiven Qualitätserwägungen basiert werden.[19]

Im Ergebnis werden Kunden sich eher vom Preis eines Produkt und seiner Funktionsvielfalt als von seiner Sicherheit zum Erwerb bewegen lassen;[20] Sicherheit und Qualität werden keine entscheidenden Kriterien im Markt für Softwareprodukte sein (können). Auf diese Weise entsteht das, was man einen 'Markt für Zitronen' (Akerlof 1970) — charakterisiert durch 'adverse selection' — nennt. Anbieter besserer Produkte haben in einem solchen Markt kaum Chancen, Fuß zu fassen, da sie die höheren Preise für die Entwicklung von Qualitätsprodukten nicht an die Endkunden weitergeben können, solange es keine Möglichkeit gibt, die verborgenen Qualitäten sichtbar, oder besser noch: prüfbar, zu machen.

3.3 Network Externalities

Im Unterschied zu den Märkten für Autos, Einbauküchen und exotischen Früchten ist der Softwaremarkt durch starke Netzwerkeffekte (Katzenberg und Shapiro) gekennzeichnet. In Märkten mit Netzwerkeffekten werden Produkte gehandelt die dadurch charakterisiert sind, daß die Bedeutung eines Gutes x für Kunde A wächst, wenn Kunde B ebenfalls ein Exemplar des Gutes x erwirbt und das Gut x sich durch Interoperabiltät (bzw. Komplementarität) auszeichnet.

Das klassische Beispiel sind Telefonanschlüsse: Je mehr Leute einen Telefonanschluß haben, desto mehr Leute kann ich anrufen, wenn ich mir ein Telefon zulege. Der Nutzwert meines Telefons wächst mit jedem neuen Telefonkunden mit dem ich in Verbindung treten kann.

Im Bereich der Massenmarktsoftware kann man sich den Einfluß von Netzwerkeffekten am besten dadurch veranschaulichen, daß man sich Microsofts Monopolposition und ihr Zustandekommen u.a. durch den 'faktischen Zwang zur Kompatibilität' vor Augen führt, bei dem man Dokumente zu verarbeiten ist, die nur in proprietären Formaten zur Verfügung stehen (z.B. Worddokumente mit Macros).

Dieser 'Zwang zur Kompatibilität' wird auch als 'positive feedback effect' bezeichnet und er wirkt — siehe Microsoft — in der Regel zugunsten des größten Anbieters im Markt, auf Kosten kleinerer Konkurrenten. Ist ein Markt noch nicht aufgeteilt, gilt es für alle Wettbewerber, schnell große Marktanteile zu gewinnen und sich so die Vorteile des 'positive feedback' zu sichern. Daraus folgt, daß die 'time to market' ein kritischer Erfolgsfaktor im Geschäft mit der Software ist.

> «[T]ime to market is a critical factor in remaining in the software business.» Hamlet (1995, S. 194)

Faßt man das eben gesagte zusammen, so lautet die Erkenntnis, daß die ökonomischen Anreize, früh mit seinem Produkt auf dem Markt zu sein, um dieses als de facto–Standard zu etablieren, für Softwarehersteller gewöhnlich hoch sind.

Das hat Konsequenzen für die Qualität und Sicherheit der angebotenen Produkte. Senkt ein Hersteller die Qualitätsanforderungen bis zur Auslieferung seines Produkts, so kann er wertvolle ‘time to market’ gewinnnen. Hat sich das Produkt einmal etabliert, ist es für Kunden praktisch unmöglich, zu einem — i.d.R. inkompatiblen — Konkurrenzprodukt zu wechseln.[21] Eine dominante Marktposition läßt sich solange halten, wie die Schwelle zum Wechseln hinreichend hoch ist.

In der Konsequenz werden Betrachtungen von Qualität und Sicherheit hinter solchen der Nutzbarkeit, d.h. Kompatibiltät, zurückstehen (müssen) (Gehring 2003).

> «By keeping its interface proprietary and by providing an exclusive set of applications, a platform owner has some hope of exploiting “network effects” to become a de facto standard in the market.» Samuelson und Scotchmer (2002, S. 1617)

Die Schwelle zum Wechseln kann noch künstlich erhöht werden, indem proprietäre Schnittstellen und Dateiformate zum Einsatz kommen und bei Bedarf ‘aktualisiert’ werden: Damit ist man den Konkurrenten, die sich in einem Iminationswettbewerb befinden, immer ein Schritt voraus.

Wasserdicht wird die Marktdominanz jedoch erst in dem Moment, indem man selbst den Imitationswettberb ausschalten kann. Wie das geht? Nun, durch Unterstützung von Seiten des Staates in Form von Monopolrechten. Im Softwarebereich heißt daß: Patente.

3.4 Und ‘Open Source’?

Läßt man die oben angeführten empirischen Erkenntnisse zu Open Source Software einmal außer acht, finden sich doch noch immer genügend theoretiche Argumente, die zugunsten von Open Source sprechen. An dieser Stelle meine ich dabei nicht die bekannten informatischen Argumente, sondern die weniger bekannten ökonomischen.

Open Source Software

- macht die Code–Qualität transparent und somit objektiv meßbar;
- ist lizenzkostenfrei und auch sonst kostengünstig, was die Verbreitung sicherer Software fördert (siehe Apache);
- verwendet offene Standards ohne Monopolwirkung, d.h.
- ermöglicht und fördert den Wettbewerb zwischen Anbietern kompatibler Produkte.

Wie die Entwicklungen der jüngsten Zeit gezeigt haben, gerät der dominierende Softwareanbieter dadurch nennenswert unter Druck:

- “Sicherheit” wurde zum obersten Ziel erklärt und eine ‘trustworthy computing’–Initiative ins Leben gerufen;[22]
- dem Bedürfnis zur Code–Inspektion wird durch die Ausweitung des ‘shared code’–Programes Rechnung getragen;[23]
- Preise werden gesenkt, um mit freier/Open Source Software konkurrenzfähig zu bleiben;[24]
- Schnittstellen, bisher als Kronjuwelen gehütet, sollen –teilweise jedenfalls– offengelegt werden;[25]
- Kerntechnologie im sicherheitsrelevanten Bereichen der Software wird nicht mehr in der Art der ‘security through obscurity’ behandelt.[26]

Das sind deutliche Fortschritte ggü. der Situation vor ein oder zwei Jahren und zeigt, daß Sicherheit und Qualität beleibe kein Randthema bleiben müssen. Ohne den Marktdruck im Bereich Open Source der insbesondere von Linux ausgeht, wäre eine solche kulturelle Entwicklung bei Microsoft wohl undenkbar.

4 TCPA und ‘Open Source’ —Eine Spekulation

Im letzten Teil möchte ich mich einer quasi brandaktuellen Entwicklung kritisch zuwenden. Es soll um die Frage gehen, inwieweit das TCPA–Konzept, das von seinen Befürwortern als eine Art ultimative Lösung aller Sicherheitsprobleme gehandelt wird, mit Open Source vereinbar sein kann.

4.1 Das ‘trusted system’–Konzept

Das ‘trusted computing’–Konzept der Trusted Computing Platform Alliance (TCPA) hat bereits im Vorfeld seiner Umsetzung massive Kritik erfahren (Anderson 2002a). Ursächlich dafür verantwortlich dürfte die Vermischung zweier unterschiedlicher Vorstellungen davon sein, wie und wozu ‘trusted systems’ zum Einsatz kommen sollen. An dieser Stelle ist daher ein wenig Aufklärungsarbeit gefordert.

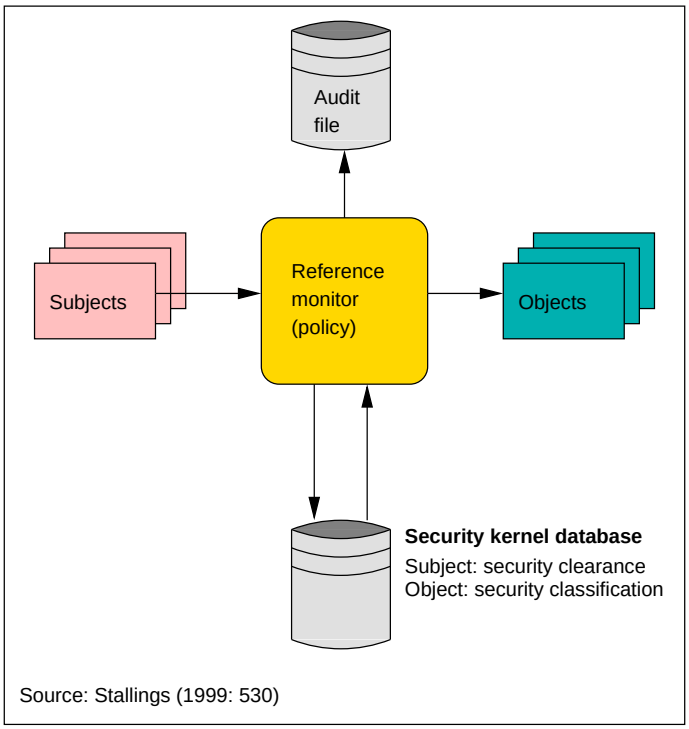
Für diejenigen die mit dem TCPA–Konzept nicht vertraut sind, folgt zuerst ein kurze Einführung. Anschließend werden einige Aspekte diskutiert die m.E. problematisch für den Open Source Prozeß sind bzw. werden können.[27]

4.1.1 Der klassische Ansatz

Eine Vorstellung, die ursprünglichere von beiden, hat ihre Wurzeln in den alten Bestrebungen, IT–Systeme sicher zu machen. An die Stelle der bloßen Mechanismen zur Durchsetzung einer ‘security policy’ tritt ein Mechanismus zur prüfbaren Durchsetzung einer ‘security policy’:

> «**Trusted System** A computer and operating system that can be verified to implement a given security policy.» (Stallings 1999, 543)

Das klassische Konzept der Zugriffsrechteverwaltung wird bei ‘trusted systems’ durch einen sogenannten ‘reference monitor’ ersetzt, der für die Durchsetzung der einsatzspezifischen ‘security policy’[28] sorgt. Gebildet wird dieser ‘reference monitor’ i.d.R. aus einer Kombination von Hard– und Software (Stallings 1999, 529–531). Die folgende Grafik zeigt das Schema des ‘reference monitor’–Konzepts.



Source: Stallings (1999: 530)

Die Implementation des Softwareanteils des ‘reference monitor’, von der TCPA auch ‘trusted computing base’ genannt (Pearson u. a. 2003, 8), wird in der Regel in geeigneter Form im Betriebssystem vorgenommen, wobei die Hardwareunterstützung eine Umgehung der ‘security policy’ durch Software verhindern soll. Dadurch soll die Sicherheit der im System gespeicherten Daten gegenüber unauthorisiertem Zugriff garantiert werden.

Ein solcher Ansatz ist generisch und unterscheidet Daten nicht nach ihrem Inhalt, sondern nach ihrer Klassifikation im System.

4.1.2 Der TCPA–Ansatz

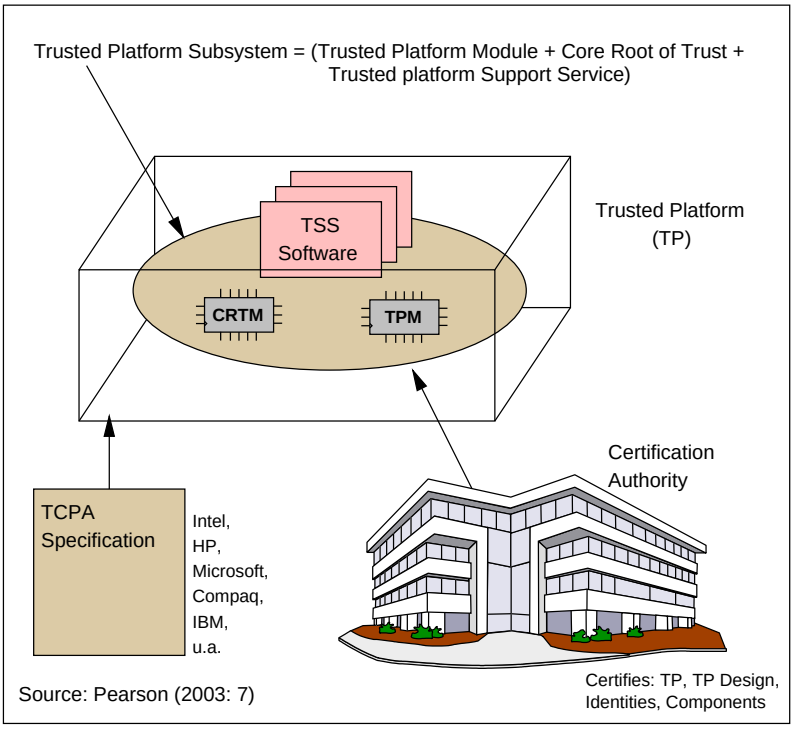
Die von den im TCPA–Konsortium zusammengeschlossene Unternehmen vorgelegte Spezifikation für eine ‘trusted computing platform architecture’ versucht, das Problem inkompatibler Systeme und hoher Kosten zu überwinden, das durch proprietäre Ansätze immer wieder erzeugt wird.

In diesem Sinne wurde eine Reihe von gemeinsamen Merkmalen festgelegt, die ‘trusted systems’ aufweisen müssen, damit sie als TCPA–konform gelten können und sich ‘trusted platform’ (TP) nennen dürfen.

Eine ‘trusted platform’ besteht demnach (Pearson u. a. 2003, 5–8) aus:

- spezieller Hardware, d.h. eine ‘trusted component’ ist im System integriert;[29]
- TCPA–konformer Softwareunterstützung auf Betriebssystemebene;
- kryptographischen Schlüsseln;
- Zertifizierungsdienstleistungen für Hard– und Software.

Die folgende Abbildung zeigt schematisch das Zusammenspiel der einzelnen Bestandteile, die zum Trusted Platform Model gehören.



Source: Pearson (2003: 7)

Alle Bestandteile werden in einer sogenannten ‘chain of trust’ so miteinander verknüpft, daß —aus Anwenderperspektive— ein wohldefininierter, sicherer Systemzustand garantiert wird. Welcher Art dieser Zustand ist, wird gegenüber dem Anwender durch eine sogenannte ‘Trusted Third Party’ (TTP) erklärt.

> «It is this aspect that ultimately differentiates a *Trusted Platform* from a conventional *secure computer*.» (Pearson u. a. 2003, 7)

Im TCPA–Ansatz wird der vom ‘trusted systems’–Konzept bekannte Grundsatz der ‘policy’–Neutralität weitestgehend übernommen. Die Definition und Durchsetzung einer ‘security policie’ ist nicht *ax ante* mit einem bestimmten Einsatzzweck (‘copyright box’) verbunden.

Der Form nach wird die ‘security policy’ einer ‘trusted platform’ durch das Zusammenwirken der genannten Komponenten implizit beschrieben. Es handelt sich um klassische *Code–Regulierung* im Sinne von Lawrence Lessig (1999). Die Gestaltung dieser impliziten ‘security policy’ liegt mithin in den Händen jener, die über die Gestaltung der einzelnen Komponenten zu entscheiden haben. Dies ist der *entscheidende* Punkt, auf den ich später zurückkommen werde.

4.1.3 Der Ansatz von *Stefik*

In einem einflußreichen Aufsatz hat Mark Stefik (1997) dem alten Konzept des ‘trusted system’ einen neuen Anstrich gegeben und ihm eine neue Rolle zugedacht: Aus einem Konzept, das schlicht die Durchsetzung von im Prinzip beliebigen ‘security policies’ gewährleisten sollte, wurde ein Konzept, daß die Durchsetzung eines bestimmten Typs von ‘security policies’ zum Gegenstand hatte. Ziel seiner Bemühungen war es, aus normalen Anwendercomputern (PCs) ‘Copyright–Verwaltungsmaschinen’ zu machen:

> «A trusted system is a system that can be relied on to follow certain rules. In the context of digital works, a trusted system follows rules governing the terms, conditions and fees for using digital works.» (1997, Abs. II (A) Par. 1)

Stefik hat diesen Ansatz später weiter ausgebaut (1999) und diskutiert ‘trusted systems’ im Anwendungskontext des Internets als

> «systems that protect and govern the use of digital objects and information for commercial purposes. . . to ensure against unsanctioned access and copying and to produce accurate accounting and reporting data for billing.» (55).

Kurzum, es handelt sich bei ‘trusted systems’ nach Stefiks Auffassung um «Copyright Boxes» (55), d.h. die durchzusetzende ‘security policy’ dient dem Zweck, ein ‘Digital Rights Management (DRM)’–System zu implementieren. Ohne auf die Details von DRM–System genauer eingehen zu wollen, möchte ich drei der von Stefik genannten Aufgaben des Systems präzise herausarbeiten. Ein ‘trusted system’ im Sinne von Stefik wird:

1. den Zugang zu im System gespeicherten Daten kontrollieren (‘access’);
2. Zugriffsinformationen speichern (‘accounting’); und
3. Daten zur Weitergabe an eine ‘Zahlungsstelle’ erzeugen (‘reporting data for billing’).

An diesen Punkten entzündet sich die schon erwähnte Kritik an ‘trusted systems’. Nur zum Teil berechtigt ist ihre oft undifferenzierte Übertragung auf das TCPA–Konzept.

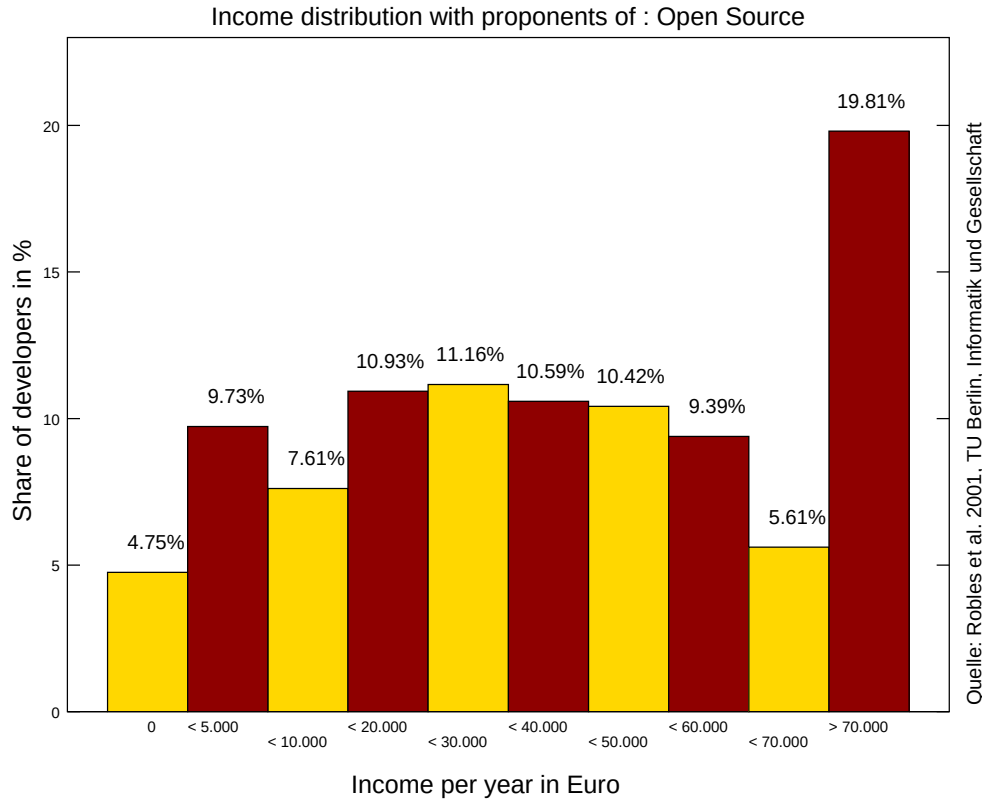
4.1.4 Die Kritik

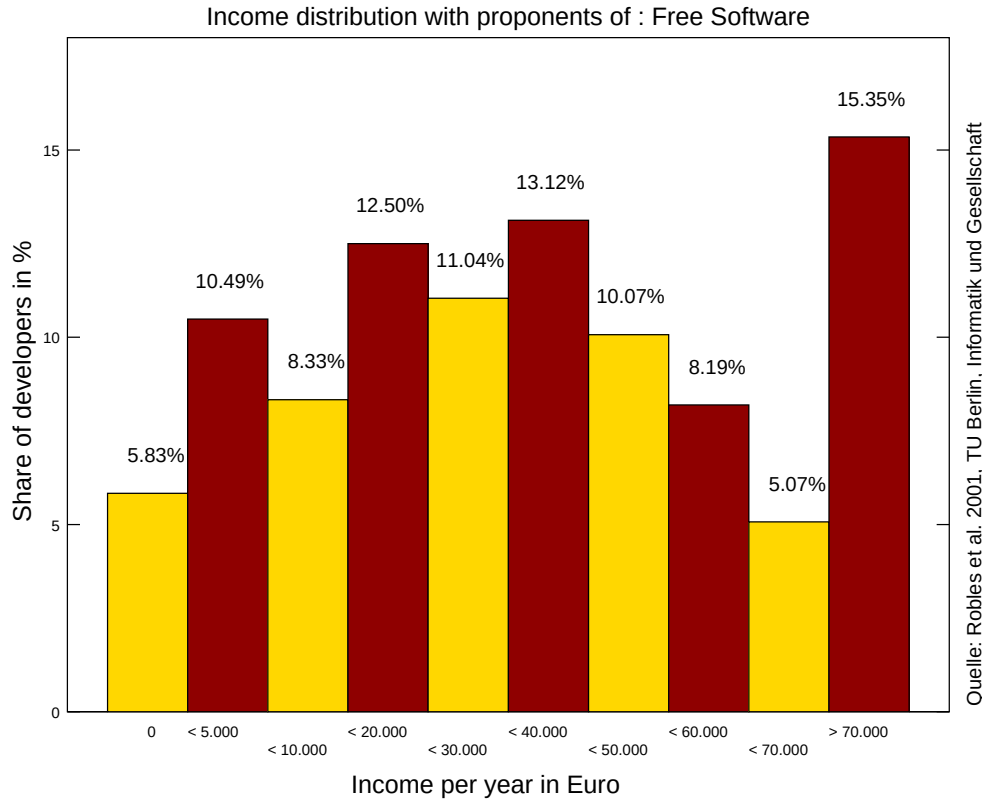
Einzelne Kritiker fordern bereits, die TCPA–Spezifikation nach ‘Open Source’–Prinzipien umzusetzen (Arbaugh 2002, 79). Vom HP–Forscher Dirk Kuhlmann wurde in diesem Sinne auf dem 2002’er Chaos Communication Congress in Berlin ein recht konkreter Vorschlag unterbreitet (Krempl 2002).

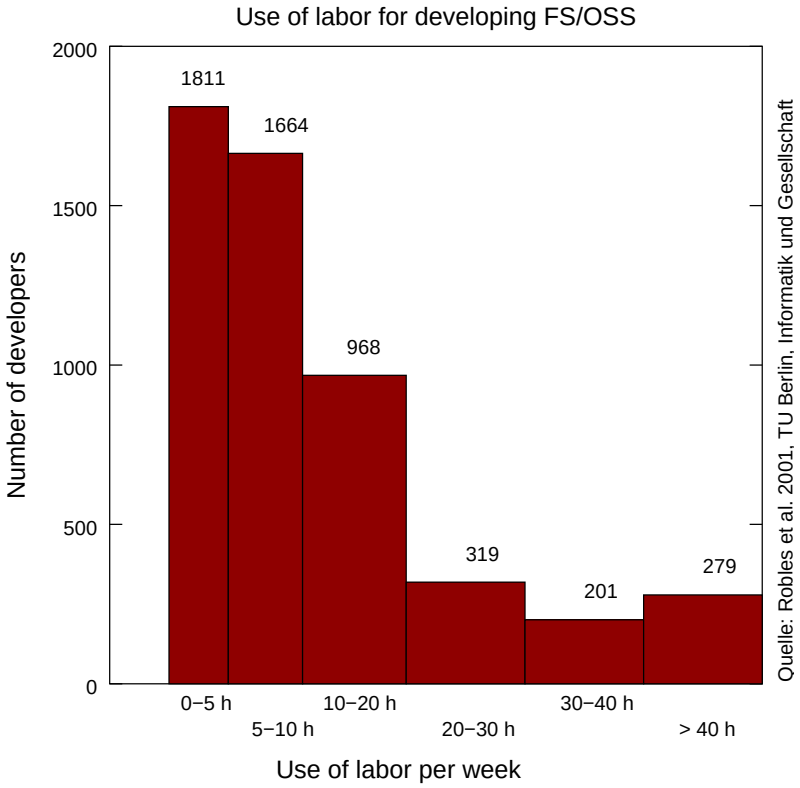
4.2 TCPA und Open Source

[TODO]

4.2.1 Soziologische Randbedingungen




Income distribution with proponents of : Free Software
Share of developers in %
0
5
10
15
5.83%
10.49%
8.33%
12.50%
11.04%
13.12%
10.07%
8.19%
5.07%
15.35%
0
< 5.000
< 10.000
< 20.000
< 30.000
< 40.000
< 50.000
< 60.000
< 70.000
> 70.000
Income per year in Euro
Quelle: Robles et al. 2001. TU Berlin. Informatik und Gesellschaft



(Robles u. a. 2001)(Ghosh u. a. 2002)

Notes

[1]Vortrag gehalten im Rahmen des CAST–Workshops “Sicherheit mit Open Source”, Darmstadt, 20. März 2003.

[2]Der Präzision wegen möchte ich festhalten, daß ich den Terminus ‘Open Source Software’, dem etablierten Sprachgebrauch folgend, durchweg generisch, d.h. sowohl für ‘Open Source Software’ im Sinne der ‘Open Source Definition’ (http://www.osi.org/osd.html) als auch im Sinne der ‘Free Software Definition’ (http://. . .) gebrauche (es sei denn, es ist anders ausgewiesen):

> «The source must be available for redistribution without restriction and without charge, and the license must permit the creation of modifications and derivative works, and must allow those derivatives to be redistributed under the same terms as the original work.» (O’Reilly 1999, 34)

[3]Der Aufsatz gibt den Inhalt des Vortrages (vgl. FN 1) nicht wörtlich, sondern in erweitertem Umfang wieder.

[4]Siehe (Gehring 2003; Gehring 2002).

[5]Eine Anzahl von Artikeln zum Thema finden sich z.B. auf dem Webserver des Heise–Verlages unter http://www.heise.de/ct/pgpCA/krypto.shtml [11 May 2003]. Informativ, wenn auch nicht mehr ganz aktuell, ist auch die Seite der Electronic Frontier Australia (EFA) zu Kryptofragen, erreichbar unter http://www.efa.org.au/Issues/Crypto/cryptfaq.html [11 May 2003]. Zeitnahe Betrachtungen sind auch zu entnehmen: Lance J. Hoffman, Building in Big Brother: The Cryptographic Policy Debate, Springer, 1995.

[6]Einen Überblick über verschiedene Prozesse und ihre Folgen gibt Radlo (1999). LEGAL ISSUES IN CRYPTOGRAPHY by Edward J. Radlo, Partner, Fenwick & West LLP, Palo Alto, California July 1998, http://www.fenwick.com/pub/ip_pubs/Cryptography/cryptography.pdf [11 May 2003].

[7]Für weitergehende Betrachtungen zu diesem Aspekt siehe u.a. die Aufsätze im Virginia Journal of Law and Technology, Vol. 2, online: http://www.vjolt.net/vol2/issue/index.html [19 May 2003].

[8]Zwar war ‘open source’ wohl der dominierende Faktor, nicht jedoch der einzige, wie das Beispiel ‘Fortify’ zeigt. Ein Resultat der Kryptodebatte war, daß Webbrowser die in den USA entwickelt und von dort exportiert wurden nur mit schwachen Verschlüsselungstechniken ausgestattet wurden. Das betraf sowohl den damals populären Netscape Navigator als auch seinen Konkurrenten, den Internet Explorer von Microsoft. Statt mit langen, sicheren Schlüsseln, konnten beide Browser nur mit kurzen Schlüsseln aufwarten. In der Folge sahen sich die Anwender mit einem Dilemma konfrontiert: Wie sicher war die verschlüsselte Kommunikation mit SSL, wie sie etwa beim Internet–Einkauf eingesetzt wurde? Konnte man seine Kreditkartennummer unbesorgt von seinem Webbrowser übertragen lassen? Wie groß war die Mißbrauchsgefahr? Solche Fragen waren nicht leicht zu beantworten. Hilfe kam dann aus dem Internet. In Australien wurde ein Program namens Fortify entwickelt, dessen Anwendung es ermöglichte, die verar-

beitete Schlüssellänge aufzurüsten. Das Programm wurde auf einem Webserver verfügbar gemacht und konnte so, da in Australien wesentlich liberalere Bestimmungen gelten, in alle Welt exportiert werden. Im Endeffekt war die Exportregulation zwar nicht direkt wirkungslos, aber kein wirkliches Hindernis.

[9]Siehe dazu Tim J. Hudson und Eric A. Young (1998): SSLeay FAQ, online: http://www2.psy.uq.edu.au/~ftp/Crypto/ [11 May 2003]. Informativ, allerdings eher für technisch Interessierte, ist auch der Artikel von Holger Reif (1996): Ready encrypted. Secure Socket Layer: Encode and certify with SSLeay, englische Übersetzung eines Artikels in iX 6/1996, Seite 128, online: http://www.heise.de/ix/artikel/E/1996/06/128/ [11 May 2003].

[10]Beide arbeiten mittlerweile für RSA in Australien. Vgl. http://www.cryptsoft.com/ [11 May 2003].

[11]Siehe http://www.openssl.org [11 May 2003].

[12]Vgl. Dietmar Mueller: Clinton lockert Krypto- Exportbeschränkungen, ZDNet, 20. September 1999, 00:01 Uhr, online: http://news.zdnet.de/story/0,,t101-s2048923,00.html [18 May 2003].

[13]Vgl. auch Diffie und Landau (2002).

[14]Gemäß der aktuellen Netcraft–Statistik vom Mai 2003 hält Apache einen Marktanteil von etwa 62%, wohingegen der IIS auf nur ca. 28% kommt. Vgl. http://news.netcraft.com/ [11 May 2003].

[15]Siehe dazu: Isabel Münch (Hg.): Apache Webserver Sicherheitsstudie, online: http://www.bsi.bund.de/literat/studien/sistudien/Apache_2003.pdf, und Isabel Münch (Hg.): Microsoft Internet Information Server Sicherheitsstudie, online: http://www.bsi.bund.de/literat/studien/sistudien/IIS_2003.pdf, beide Bundesamt für Informationssicherheit, 2003.

[16]Reasoning (2003): Illuma Defect Data Report for Linux 2.4.19 Networking Sample, zu beziehen über http://www.reasoning.com/news/pr/03_03_03.html [22 May 2003].

[17]Zur Zeit der Auswertung der Untersuchung war bereits eine Nachfolgeversion des Linuxkernels (2.4.20) verfügbar, die diesen Fehler nicht mehr enthielt. Die Fehlerbeseitigung erfolgte unabhängig von den Reasoning–Aktivitäten (Reasoning 2003, S. 10).

[18]Folgt man (Gordon und Loeb 2002), so kann dieses Maß u.U. ausreichend sein, sich gegen die ökonomischen Folgen eines Einbruchs in das eigene IT–System abzusichern. Allerdings untersuchen die Autoren in ihrem (theoretischen) Modell die Folgen von Externalitäten, wie sie in einer vernetzten Umgebung eher die Regel sein dürften — man denke an die vielen Viren und Würmer, die sich in den letzten Jahren regelmäßig ausgebreitet haben —, nicht. Die Übertragbarkeit auf vernetzte IT–Systeme ist insofern nicht unmittelbar aus dem Modell abzuleiten.

[19]Subjektive Qualitätserwartungen, Emotionen, kommen dagegen durchaus ins Spiel und werden maßgeblich durch ‘brand–building’ von seiten der Hersteller angesprochen: ‘Wo Microsoft drauf steht kann man Microsoft–Qualität erwarten.’

[20]Dieser Effekt ist auch in Laborstudien zu beobachten (Hong und Lerch 2002).

[21]Die Anreize zum Wechsel müßten schon außerordentlich hoch sein, würde der Kunde doch die bisher getätigten Investitionen im Falle eines inkompatiblen Produkts praktisch vollständig abschreiben müssen. Jeder ökonomisch handelnde Akteur scheut davor verständlicherweise zurück. Letztlich läuft es auf eine politische Entscheidung pro oder contra die Abhängigkeit von einem bestimmten Hersteller hinaus.

[22]Vgl. http://www.microsoft.com/billgates/speeches/2002/04-17glc.asp [17 Apr 2002].

[23]Vgl. http://www.idg.net/go.cgi?id=643493 [22 Feb 2002].

[24]Vgl. http://zdnet.com.com/2100-1104-990741.html [03 Mar 2003].

[25]Vgl. http://zdnet.com.com/2100-1104-955655.html [22 May 2003].

[26]Vgl. http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/news/NGSCB.asp [22 May 2003].

[27]Eine ausführlichere Darstellung erfolgt in Kuhlmann und Gehring (2003).

[28]Eine umfangreiche Darstellung von ‘security policies’ geben (Anderson u. a. 2001).

[29]Derzeit ist dafür ein sogenannter ‘TCPA–chip’ vorgesehen. Im Prinzip kommt aber die Integration der TPM–Funktionalität in die CPU in Frage und wurde auch bereits von verschiedenen Herstellern angekündigt.

Literatur

[Akerlof 1970] AKERLOF, George A.: The Markets for Lemons–Quality Uncertainty and the Market Mechanism. In: *Quarterly Journal of Economics* 84 (1970), Nr. 3, S. 488–500

[Anderson 2002a] ANDERSON, Ross: Free Speech Online and Offline. In: *IEEE Computer* 35 (2002a), Nr. 6, S. 28–30

[Anderson u. a. 2001] ANDERSON, Ross; STAJANO, Frank ; LEE, Jong-Hyeon: Security Policies. In: *Advances in Computers* 55 (2001), S. 185–235. – ISBN 0–12–012155–7

[Arbaugh 2002] ARBAUGH, Bill: Improving the TCPA Specification. In: *IEEE Computer* 35 (2002), Nr. 8, S. 77–79

[Chai und Gao 2003] CHAI, Winston; GAO, Ken: McNealy: It's mankind vs. Microsoft. In: *CNET News.com* (2003). – CNET News, 19 March 2003, online: http://news.com.com/2100-1104-993226.html [20 Mar 2003]

[Chou u. a. 2001] CHOU, Andy; YANG, Junfeng; CHELF, Benjamin; HALLEM, Seth ; ENGLER, Dawson: An Empirical Study of Operating Systems Errors, 2001. – Proceedings of the 18th ACM Symposium on Operating Systems Priciples, 2001, Banff, Alberta, Canada, online: http://portal.acm.org/citation.cfm?id=502042&coll=portal&dl=ACM&ret=1 [11 Mar 2003]

[Coffee 2002] COFFEE, Peter: Rotten Code Is a Weapon of Mass Destruction. In: *eWeek* (2002). – eWeek, 09 Dec 2002, online: http://www.eweek.com/2/0,3959,762844,00.asp [22 May 2003]

[Diffie und Landau 2001] DIFFIE, Whitfield; LANDAU, Susan: The Export of Cryptography in the 20th Century and the 21st / Sun Microsystems. 2001. – Forschungsbericht. online: http://http://research.sun.com/features/tenyears/volcd/TOC.htm [22 May 2003]

[Diffie und Landau 2002] DIFFIE, Whitfield; LANDAU, Susan: September 11th Did Not Change Cryptography Policy. In: *Notices of the AMS* 49 (2002), Nr. 4, S. 450–454. – online: http://www.ams.org/notices/200204/comm-diffie.pdf [22 May 2003]

[Gehring 2002] GEHRING, Robert A.: Software Development, Intellectual Property Rights, and IT Security. (2002). – 1st International Workshop on Economics and Information Security, May 16–17, 2002, University of California, Berkeley, online: http://www.sims.berkeley.edu/resources/affiliates/workshops/econsecurity/ [09 Mar 2003]

[Gehring 2003] GEHRING, Robert A.: Software Development, Intellectual Property Rights, and IT Security. In: *Journal of Information, Law & Technology* 8 (2003). – forthcoming 2003, online: http://elj.warwick.ac.uk/jilt/

[Ghosh u. a. 2002] GHOSH, Rishab A.; GLOTT, Rüdiger; KRIEGER, Bernhard ; ROBLES, Gregorio: Free/Libre and Open Source Software: Survey and Study / International Institute of Infonomics, Maastricht, The Netherlands and Berlecon Research GmbH, Berlin, Germany. 2002. – Forschungsbericht. online: http://www.infonomics.nl/FLOSS/report/ [12 Mar 2003]

[Gordon und Loeb 2002] GORDON, Lawrence A.; LOEB, Martin P.: The Economics of Information Security Investment. In: *ACM Transactions on Information and System Security* 5 (2002), Nr. 4, S. 438–457

[Hamlet 1995] HAMLET, Dick: Software Quality, Software Process, and Software Testing. In: *Advances in Computers* 41 (1995), S. 191–229. – ISBN 0–12–012141–7

[Hong und Lerch 2002] HONG, Se-Joon; LERCH, F. J.: A Laboratory Study of Consumers' Preferences and Purchasing Behavior with Regards to Software Components. In: *The DATABASE for Advances in Information Systems* 33 (2002), Nr. 3, S. 23–37

[Hudson und Young 1998] HUDSON, T. J.; YOUNG, E. A.: SSLeay and SSLapps FAQ. (1998). – online: `http://psych.psy.uq.oz.au/~ftp/Crypto/` [11 Mar 2003]

[Krempl 2002] KREMPL, Stefan: HP–Forscher wirbt für Allianz von Open–Source–Bewegung und TCPA. In: *heise newsticker* (2002). – heise newsticker, 28 Dec 2002, online: `http://www.heise.de/newsticker/data/se-28.12.02-003/` [28 Dec 2002]

[Kuhlmann und Gehring 2003] KUHLMANN, Dirk; GEHRING, Robert A.: TCPA, DRM, and Beyond. In: BECKER, Eberhard (Hrsg.); BUHSE, Willms (Hrsg.); GÜNNEWIG, Dirk (Hrsg.) ; RUMP, Niels (Hrsg.): *Digital Rights Management. Technological, Economic, Legal and Political Aspects* Bd. ???, Berlin, Heidelberg, New York : Springer, 2003. – erscheint 2003; ISBN 3–7880–???, S. ???

[Lessig 1999] LESSIG, Lawrence: *Code and Other Laws of Cyberspace*, New York, NY : Basic Books, 1999. – ISBN 0–465–03912–X

[O'Reilly 1999] O'REILLY, Tim: Lessons From Open Source Software Development. In: *Communications of the ACM* 42 (1999), Nr. 4, S. 33–37

[Pearson u. a. 2003] PEARSON, Siani; BALACHEFF, Boris; CHEN, Liqun; PLAQUIN, David ; PROUDLER, Graeme: *Trusted Computing Platforms. TCPA Technology in Context*, Upper Saddle River, NJ : Prentice Hall PTR, 2003. – ISBN 0–13–009220–7

[Pipkin 2000] PIPKIN, Donald L.: *Information Security. Protecting the Global Enterprise*, Upper Saddle River, NJ : Prentice Hall PTR, 2000. – ISBN 0–13–017323–1

[Pourzandi u. a. 2002] POURZANDI, Makan; HADDAD, Ibrahim; LEVERT, Charles; ZAKRZEWSKI, Miroslav ; DAGENAIS, Michel: A Distributed Security Infrastructure for Carrier Class Linux Clusters, 2002. – IEEE Cluster 2002, September 23-26, 2002, Chicago, Illinois online: `http://www.cs.concordia.ca/~grad/i_haddad/conf2002.html` [09 Mar 2003]

[Raymond 1998] RAYMOND, Eric S.: The Cathedral and the Bazaar. In: *First Monday* 3 (1998), Nr. 3. – online: `http://firstmonday.org/issues/issue3_3/raymond/index.html` [22 May 2003]

[Reasoning 2003] REASONING: *Open-source versus commercial software: a quantitative comparison*. Technical White Paper. February 2003. – online: `http://www.reasoning.com/downloads/Open_Source_White_Paper_v1.1.pdf` [22 May 2003]

[Ritchey 2001] RITCHEY, Ronald W.: Open Source Vs. Close Source Software. An Experiment To Determine Which is More Secure / Booz Allen Hamilton. 2001. – Forschungsbericht. online: `http://www.isse.gmu.edu/faculty/ofut/classes/763/studpapers/Ritchey.pdf` [22 Dec 2001]

[Robles u. a. 2001] ROBLES, Gregorio; SCHEIDER, Hendrik; TRETKOWSKI, Ingo ; WEBER, Niels: Who Is Doing It? A research on Libre Software developers. (2001). – online: `http://ig.cs.tu-berlin.de/s2001/ir2/ergebnisse/OSE-study.pdf` [24 Mar 2002]

[Samuelson und Scotchmer 2002] SAMUELSON, Pamela; SCOTCHMER, Suzanne: The Law and Economics of Reverse Engineering. In: *Yale Law Journal* 111 (2002), Nr. 7, S. 1575–1663

[Schechter 2002] SCHECHTER, Stuart: Quantitatively Differentiating System Security. (2002). – 1st International Workshop on Economics and Information Security, May 16–17, 2002, University of California, Berkeley, online: http://www.sims.berkeley.edu/resources/affiliates/workshops/econsecurity/ [09 Mar 2003]

[Shankland 2003] SHANKLAND, Stephen: Study lauds open-source code quality. In: *CNET News.com* (2003). – CNET News, 19 February 2003, online: http://news.com.com/2100-1001-985221.html [12 Mar 2003]

[Singleton 1998] SINGLETON, Solveig: Encryption Policy for the 21st Century. A Future without Government–Prescribed Key Recovery / Cato Institute. 1998. – Forschungsbericht. online: http://www.cei.org/pdf/2915.pdf [16 Dec 2002]

[Stallings 1999] STALLINGS, William: *Cryptography and Network Security. Principles and Practice*, 2nd ed., Upper Saddle River, NJ : Prentice Hall, 1999. – ISBN 0–13–869017–0

[Stefi k 1997] STEFIK, Mark: Shifting the Possible: How Trusted Systems and Digital Property Rights Challenge Us to Rethink Digital Publishing. In: *Berkeley Technology Law Journal* 12 (1997), Nr. 1, S. 137. – online: http://www.law.berkeley.edu/journals/btlj/articles/vol12/Stefik/html/reader.html [07 Mar 2003]

[Stefi k 1999] STEFIK, Mark: *The Internet Edge. Social, Legal, and Technological Challenges for a Networked World*, Cambridge, MA; London, UK : MIT Press, 1999. – ISBN 0–262–19418–X

[Vossler und Voas 2000] VOSSLER, Colleen K.; VOAS, Jeffrey: Defective Software: An Overview of Legal Remedies and Technical Measures Available to Consumers. In: *Advances in Computers* 53 (2000), S. 451–497. – ISBN 0–12–012153–0

[Wright u. a. 2002] WRIGHT, Chris; COWAN, Crispin; SMALLEY, Stephen; MORRIS, James ; KROAH-HARTMAN, Greg: Linux Security Modules: General Security Support for the Linux Kernel. In: *Ottawa Linux Symposium 2002. Proceedings*, 2002. – online: http://www.cse.ogi.edu/~crispin/ [09 Mar 2003]