Unsichere Software - Eine systemische Betrachtung

Vortrag[1] auf der Herbsttagung der Europäischen Akademie Bad Neuenahr−Ahrweiler

22.-23. November 2001

Dipl.-Inform. Robert A. Gehring

rag@cs.tu-berlin.de

Einleitung

Wir können seit Jahren beobachten, daß die rechtlichen Regularien im Bereich des ‘geistigen Eigentums’ -des ‘intellectual property’- zu Gunsten der Rechteinhaber und Rechteverwerter verschärft werden. Das gilt weltweit und betrifft Urheberrechts- und Patentschutz, sowie auch Markenrechte. Die zunehmende Digitalisierung von *Content* und das Vordringen des Internet bilden den Hintergrund dieser Entwicklung.

Betrachtet man die politischen und rechtlichen Verfahren, in denen die Schutzrechtsregelungen ausgehandelt werden, kann man eines feststellen: Die Frage, inwieweit die Sicherheit der Informationstechnologie davon betroffen ist, findet in der Regel keine Berücksichtigung

Der Druck durch Lobbyisten der Rechteverwerter auf die Politiker ist enorm. Sie dringen auf einen möglichst umfassenden Schutz ihrer kommerziellen Interessen, ohne Ausnahmen. Wenn bei ihnen von *Sicherheit* die Rede ist, dann geht es mitnichten um die Sicherheit der zugrundeliegenden IT-Infrastruktur. Vielmehr meinen sie damit die Sicherheit ihrer Vermögenswerte.

Eine solche Attitüde ist alles andere als unproblematisch, denn: Schutzrechte für Software werden selbst zum Sicherheitsrisiko, wenn sie die Verbreitung unsicherer Software befördern.

Und das heißt in der Konsequenz: Je mehr die gesellschaftlichen Institutionen sich elektronisch vernetzen, desto größer wird ihr Sicherheitsrisiko - durch unangemessene rechtliche Regularien.

Rechtliche Regularien existieren nämlich nicht in einer idealen Welt, obwohl man den Eindruck gewinnen kann, daß bei ihrer Schaffung regelmäßig von einem solchen Standpunkt argumentiert wird. Vielmehr entfalten sie ihre Wirkung unter ökonomischen und -im IT-Bereich- auch unter technischen Randbedingungen.

---

[1] Der Vortrag basiert auf der Präsentation **«"Software Patents" − IT-Security at Stake?»**, vgl. Gehring 2001.

Ursachen für die Unsicherheit von Software

Will man das Problem der Unsicherheit von Software verstehen, muß man eine systemische Betrachtung anstellen. Erst in der Analyse der Interaktion von technologischen, rechtlichen und ökonomischen Wechselwirkungen wird klar, warum es immer wieder zu Sicherheitsproblemen bei Software kommt - kommen muß.

Vor der Ursachenforschung möchte ich einen Blick auf die Realität werden.

Beispiele aus der Wirklichkeit

Im Jahr 1999 verursachte das so genannte «I love you»-Virus weltweit Schäden von ca. 12 Milliarden US$ (McAfee 2001).

Nimmt man zum Vergleich den Betrag, der 1998 für alle Virenschäden zusammenkam, so hat sich der Schaden mehr als verdoppelt - für ein einzelnes Virus!

Im Jahr 2000 war das «I love you»-Virus immer noch aktiv. Innerhalb von zwei Wochen verursachte es mehr als 6.7 Milliarden $ Schaden.

Aber Viren sind nur ein Teil des Problems.

Jüngsten Schätzungen zufolge zieht der Einsatz unzuverlässiger und unsicherer Software allein in den USA jährliche Kosten von etwa 78 Milliarden US$ nach sich (Levinson 2001).

Zum Vergleich: Der Jahresumsatz des weltgrößten Software-Herstellers, Microsoft, beträgt ca. 25 Milliarden US$, d.h. ein Drittel des genannten Betrages.

Wie die folgende Analyse zeigt, ist an all dem nicht das Schicksal schuld. Vielmehr muß man schlicht und einfach ein Marktversagen konstatieren. Der Reihe nach möchte ich einige der Ursachen dafür herausarbeiten.

Ich beginne mit den technischen Ursachen im Bereich der Software-Entwicklung.

Technische Ursachen (I)

Der klassische Software-Entwicklungsprozeß weist inhärente Schwächen auf.

Der größte Schwachpunkt besteht in der praktisch unvermeidlichen Unvollständigkeit der funktionalen Spezifikation.

Eine funktionale Spezifikation in je unterschiedlicher Ausprägung steht am Beginn der Software-Entwicklung. Sie repräsentiert eine Art Bauplan für die zu schreibende Software und liefert den Programmierern die Anhaltspunkte, was für Code sie zu schreiben haben.

Die funktionale Spezifikation ist außerdem ein Ausgangspunkt für die Festlegung von Testbedingungen.

Funktionale Spezifikationen zeichnen sich durch ihre Unvollständigkeit aus:

- Gemessen an den funktionalen Anforderungen an das Programm sind sie nicht immer, aber in der Regel unvollständig.

- Gemessen an Sicherheitsanforderungen sind funktionale Spezifikationen praktisch *immer* unvollständig.

Man kann es so zusammenfassen:

Je komplexer das zu schreibende Programm ist, und je komplexer die Einsatzumgebung des Programms ausfällt, desto unvollständiger wird seine Spezifikation sein.

> *«[M]odern systems have so many components and connections—some of them not even known by the systems' designers, implementers, or users—that insecurities always remain.»* (Schneier 2000: xii)

Technische Ursachen (II)

Wenn ein Stück Software in vollständiger Übereinstimmung mit der vorgegebenen Spezifikation geschrieben worden ist und anschließend alle Tests bestanden hat, spricht man davon, daß es *korrekt* sei.

Damit kommen wir zum nächsten Problem: Testspezifikationen sind ihrerseits in der Regel unvollständig. Aus diesem Grunde würde kein erfahrener Programmierer je behaupten, daß eine Software sicher sei, weil sie ja als korrekt getestet worden wäre.

> *«The developers are so in tune with what it should do, they cannot see what it might be able to do.»* (Pipkin 2000: 75)

Technische Ursachen (III)

Aber nicht nur die Spezifikationen sind unvollständig, auch die Testverfahren selbst sind in ihrer Aussagekraft inhärenten Beschränkungen unterworfen.

Es ist richtig, daß durch Tests jede Menge Fehler gefunden werden können. Tests sind deshalb ein notwendiger, unverzichtbarer Bestandteil der Software-Entwicklung.

Aber -und hier liegt das Problem- Tests bringen nicht alle Fehler zum Vorschein. Fehler zeigen sich im Test entweder als systematische Abweichung von den Testbedingungen oder als Zufallsfunde. Wenn sich kein Fehler zeigt, bedeutet das lediglich, daß unter den getesteten Bedingungen kein Fehler auftritt. Eine verläßliche Aussage für das Verhalten unter anderen Umständen läßt sich daraus nicht ableiten.

Der Schluß muß also lauten: Im Testverfahren kann man das Vorhandensein von Fehlern im Programm nachweisen. Das Nicht-Vorhandensein von Fehlern läßt sich durch Testen nicht nachweisen.

Soviel zu den technischen Randbedingungen. Ich komme zu den ökonomischen Ursachen für die Verbreitung unsicherer Software. Dazu untersuche ich die ökonomische Rationalität der Software-Hersteller.

Ökonomische Ursachen (I)

(1) Vor allem anderen ist proprietäre Software[2] ein Produkt, durch dessen Verkauf auf dem Markt der Hersteller Profit generieren muß.

Gelingt ihm das nicht, findet er keine Käufer für seine Ware, wird er aus dem Markt verdrängt. Der Preis für das Produkt kann daher nicht beliebig festgelegt werden, sondern ist durch die Marktnachfrage begrenzt.

Nun sind Testen und Fehlerbeseitigung die maßgeblichen Kostenfaktoren bei der Herstellung von Software. Schätzungen gehen von 40-60% Kostenanteil aus. Je umfangreicher die Tests ausfallen, desto höher müssen die Preise für das Endprodukt ausfallen. Dadurch lassen sich weniger Abnehmer finden, was direkt zu Einschnitten beim Profit führt.

Als rational entscheidender Marktteilnehmer wird ein Software-Hersteller seinen Profit maximieren wollen. Er wird sich überlegen, wie hoch seine Kosten für den Fall sein werden, daß er für Fehler seiner Ware haften muß. Dem wird er mögliche Kosteneinsparungen beim Testen und Beseitigen von Fehlern gegenüberstellen, die einerseits zu mehr Fehlern im Produkt führen, andererseits zu höheren Profiten. Fallen die möglichen Kosteneinsparungen größer aus als die Haftungssummen, so wird er sich dafür entscheiden, ein unsicheres Produkt auszuliefern.

Das ist -ökonomisch gesehen- völlig rationales Verhalten.

(2) Zum Zweiten ist der Software-Markt durch starke Informationsasymmetrien gekennzeichnet:

- Der Software-Anbieter weiß die Qualität seines Produkts vor dem Verkauf einzuschätzen.

  Ein Software-Käufer kann sich ein Urteil darüber erst erlauben, nachdem er das Produkt bezahlt, installiert und getestet hat.

  Zu diesem Zeitpunkt hat der Anbieter seinen Profit bereits gemacht.

[2] Das Attribut "proprietär" kommt −im Sinne der vorliegenden Betrachtung− einer Software dann zu, wenn daran exklusive Eigentumsrechte (property rights) geltend gemacht werden. In der Praxis sind weitere Unterscheidungen gebräuchlich, z.B. die Konformität zu Standards, die hier jedoch nicht betrachtet werden sollen.

- Da der Käufer um diese Asymmetrie bezüglich der Produktqualität weiß, kann er seine Kaufentscheidungen nicht an qualitativen Kriterien ausrichten. Ihm fehlt die Urteilsgrundlage dazu.

  Stattdessen wird er sich an anderen Kriterien, wie etwa Preis und Verbreitung der Software orientieren.

Und damit komme ich zum nächsten Punkt, zu den Netzwerk-Externalitäten.[3]

(3) Software ist ein Erfahrungsgut in einem Markt, der durch starke Netzwerk-Externalitäten gekennzeichnet ist.

Positiver Feedback, durch Netzwerk-Externalitäten verursacht, wirkt sich in solchen Märkten in der Regel zugunsten des stärksten Anbieters aus.

In der Folge sind die Anreize, zuerst mit seinem Produkt auf dem Markt zu sein, um es so als de facto-Standard zu etablieren, für den Anbieter sehr groß.

Einschränkungen bei den zeitaufwendigen Tests führen dazu, daß man mit seinem Produkt schneller auf dem Markt sein kann, um so den Wettbewerbern zuvorzukommen.

Mögen doch die Anwender sich mit den unfertigen und unsicheren Produkten herumschlagen ... nachdem sie bezahlt haben. Das Stichwort dazu: ‘Bananaware’ - Software, die beim Kunden *reift* .

Ökonomische Ursachen (II)

Nun könnte man auf die Idee kommen, daß dem mit einem angemessenen Service von Seiten des Software-Herstellers begegnen zu sei. Leider ist das aus wirtschaftlichen Gründen zur Erfolglosigkeit verdammt.

Die Service-Kapazitäten eines Herstellers von Software für einen Massenmarkt ist begrenzt, gemessen an den Millionen von Kunden. Service selbst wird dann zu einem raren Gut - mit einem entsprechenden Preis. Es wird nur einer verschwindend kleinen Minderheit der Software-Benutzer möglich sein, die geforderten Kosten für permanente Updates, Telefonsupport usw. aufzubringen.

Wendet man sich dem anderen Ende des Spektrums zu, den Anbietern von kundenspezifischer Software, so zeigt sich, daß unsichere und unzuverlässige Software ein praktisch unverzichtbarer Bestandteil des Geschäftsmodells ist:

Womit verdient ein Anbieter von kundenspezifischer Software sein Geld?

Mit Wartung, Updates, dem Stopfen von Sicherheitslöchern und neuen Versionen. Perfekt funktionierende, sichere Software würden dem Geschäft schnell ein Ende bereiten.

---

[3] Vgl. Shapiro/Varian 1999.

In vielen Technologiebereichen gibt es rechtliche Regularien, um dergleichen unerwünschte Auswirkungen geschäftlicher Tätigkeit im Zaum zu halten.

Nicht jedoch im Bereich der Software.

Rechtliche Ursachen

Wirksame Haftungsregelungen etwa, mit deren Hilfe die Hersteller zu mehr Sorgfalt gezwungen werden könnten, existieren nicht. Weder Produzentenhaftung noch Produkthaftung haben sich in der Vergangenheit als Mittel bewährt.

Als wäre das nicht genug, finden Software-Hersteller durch den besonderen Schutz im Urheberrecht und im Patentrecht weitere Unterstützung für ihr Vorgehen.

Was heißt das konkret?

Schwachstellen des Urheberrechts

(1) Urheberrechts- und Copyrightschutz für Software beinhalten einen sehr weitgehenden Schutz vor *reverse engineering*.

Es gibt darin keine Ausnahme für die Sicherheitsevaluierung oder notwendige Sicherheitserweiterungen.

Es gibt keine Ausnahme für die eigenhändige Beseitigung kleinerer Fehler, die sehr wohl ein Sicherheitsrisiko darstellen können.

Software wird im Urheberrecht und im Copyright als Sprachwerk, quasi als Literatur behandelt. Dabei handelt es sich aber um Sprachwerke, bei denen entscheidend nicht ihr Inhalt, sondern ihr Verhalten -im Zusammenspiel mit einem Computer- ist. Das wird nicht berücksichtigt.

Bei Sprachwerken ist es nun einmal so, daß man für die Verbreitung fehlerhafter Werke nicht haftet. (Es gibt Ausnahmen, wie etwa im Falle von Vertragsverletzungen.) Diese Regel gilt nicht nur für Bücher, sondern auch für Software.

Die Wahrheit sieht schlicht und einfach so aus, daß es keine effektiven Haftungsregelungen für Standard-Software gibt. Die Anwender haben die Folgen zu ertragen.

Schwachstellen des Patentrechts

Verschärft werden die Probleme durch die Besonderheit des doppelten Rechtsschutzes für Software. Zusätzlich zum Urheberrechtsschutz ist in den letzten Jahren der Patenschutz für Software ausgeweitet worden.

Patentschutz für Software gibt dem Patentinhaber die exklusiven Nutzungsrechte an der patentierten Technologie - unabhängig von der konkreten Implementierung.

Damit sind in meinen Augen insbesondere drei Probleme verbunden:

(1) Um das Risiko der Entdeckung von Patentverletzungen zu minimieren, bevorzugen die Software-Hersteller den Vertrieb ihrer Produkte im Binärcode. Dadurch wird die Transparenz der Code-Qualität verhindert.

(2) Sichere Technologie kann Gegenstand von Patenten sein, wodurch ihre schnelle Verbreitung -wegen der Lizenzkosten- behindert wird. Zudem können Konkurrenzprodukte mit ggf. höherer Qualität verhindert werden.

(3) Die Absicherung von IT-Systemen selbst kann als Geschäftsmodell patentiert werden, was in den USA bereits geschehen ist.[4]

In diesem Falle hätte man sich −um legal zu handeln− zuerst eine Lizenz zu besorgen, um sein System in bestimmter Art und Weise absichern zu dürfen.

Nimmt man die gezeigten Problembereiche aus technologischer, ökonomischer und rechtlicher Perspektive zusammen, so sieht es für die Zukunft sicherer Informationstechnologie alles andere als rosig aus.

Wie könnte man der prekären Situation entkommen?

Wie läßt sich die IT-Sicherheit verbessern?

Anders gefragt: Was kann man tun, um die IT-Sicherheit zu verbessern?

Jede Antwort hat die herausgearbeiteten Randbedingungen ins Kalkül zu ziehen. Ad hoc-Maßnahmen werden in diesem Sinne keine Lösung bringen. Vielmehr kann eine vernünftige Antwort nur lauten:

Wir brauchen eine ***adäquate Risikomanagement-Strategie***.

Eine solche Strategie muß so konzipiert werden, daß die mit dem Einsatz von Software verbundenen Risiken auf lange Sicht signifikant reduziert werden.

Aus Perspektive der Wissenschaft bedeutet das, den Einsatz von gut geprüften Standard-Komponenten zu fördern. Das beste Instrument zur gründlichen Prüfung ist das Verfahren des *peer review* durch *unabhängige* Fachleute.

---

[4] Vgl. z.B. Finjan Software, Inc. (San Jose, CA), U.S. Pat. 6,167,520 (26 Dec 2000): System and method for protecting a client during runtime from hostile downloadables; McAfee.com Corporation (Santa Clara, CA), U.S. Pat. 6,266,774 (24 Jul 2001): Method and system for securing, managing or optimizing a personal computer.

Das *peer review*-Verfahren muß dabei geeignet sein, Software-Einsatz in der Größenordnung des Internet zu begleiten. Hausinternes *peer review* eines Software-Herstellers genügt dem nicht.

Auch Zertifizierungsfirmen stellen keine Lösung dar, da sie auf den Verkauf von Zertifikaten angewiesen sind und sich dementsprechend nach den Kundenbedürfnissen, d.h. den Bedürfnissen der Software-Hersteller orientieren müssen (Anderson 2001).

An dieser Stelle kommt das Open Source-Modell in's Spiel.

Die Stärken des OSS-Ansatzes

Welche Stärken hat der Open Source-Ansatz Aufzuweisen?

Nach unserem heutigen Wissensstand ist der Open Source-Ansatz das bestgeeignete *peer review*-Verfahren, um die Sicherheit von Software zu evaluieren und zu verbessern. Die Grundlage dafür bildet die unbeschränkte[5] Verfügbarkeit des Quelltextes von Software.

Software-Entwickler in der ganzen Welt arbeiten mit dieser Codebasis, testen sie, erweitern sie und sichern sie ab.

Da der Quellcode öffentlich verfügbar ist, besteht gar kein Bedürfnis nach *reverse engineering*, das −wir erinnern uns− weitestgehend illegal war. Aus dem Quelltext kann herausgelesen werden, wie das Programm funktioniert.

Das allein löst natürlich noch keine Sicherheitsprobleme. Ohne öffentlich verfügbaren Quellcode aber, darin stimmen fast alle Experten überein, lassen sich Sicherheitsprobleme auf keinen Fall effizient bekämpfen.

Vor allen Dingen ermöglicht es die Verfügbarkeit des Quellcodes den Anwendern, Sicherheitslöcher vor Ort, d.h. schnellstmöglich zu beseitigen.

Es gibt noch eine ganze Reihe weiterer Vorteile technischer Natur, auf die ich an dieser Stelle jedoch nicht eingehen kann.

Stattdessen möchte ich noch kurz auf die ökonomischen Aspekte eingehen und fragen, ob die für proprietäre Software abgeführten ökonomischen Zwänge durch Open Source-Software überwunden werden können.

Meine Antwort ist "ja", wenigstens zum Teil.

Zum einen beseitigt die Verwendung offener Standards den Konkurrenzvorteil, den dominante Marktteilnehmer durch Eigenentwicklungen erzielen können. Kleine Anbieter bekommen ebenfalls eine Chance und ein Wettbewerb um Qualität kann in Gang kommen.

[5] d.h. die nicht durch die Ausübung exklusiver 'property rights' beschränkte Verfügbarkeit

Zum zweiten wächst die Markttransparenz, da die Qualität der angebotenen Produkte auf Code-Basis vergleichbar wird.

Und Drittens könnten sich die Hersteller durch Konkurrenten mit prüfbaren besseren Produkten veranlaßt sehen, größere Anstrengungen im Hinblick auf die Produktqualität zu unternehmen. Andernfalls würden sie −auf lange Sicht− riskieren, aus dem Markt verdrängt zu werden.

Allerdings ist das Open Source-Modell in Gefahr.

Der in den letzten Jahren zu beobachtende Goldrausch im Bereich der Software-Patente könnte zum Erliegen bringen, was -noch- so vielversprechend aussieht.

Die Gefährdung durch Patente

Open Source Software-Entwickler sind in besonderem Maße durch Patentverletzungsklagen bedroht, ist doch das Produkt ihrer Arbeit für jedermann einsehbar.

Im Gegensatz dazu schützen sich Anbieter von proprietärer Software durch den Binärvertrieb und kommen somit in den Genuß des Banns von *reverse engineering*, wie er im Urheberrecht ausgesprochen wird.

Durch die ungehemmte Patentierung selbst trivialster "Entwicklungen" bei denen die maßgebliche erfinderische Leistung im Bereich der Formulierung der passenden Patenschrift liegt, läßt sich komplexe Software heutzutage eigentlich nicht mehr risikolos ohne umfangreiche Patentrecherche durchführen.

Aber das ist die Arbeit von Patentanwälten, nicht von durchschnittlichen Programmierern.

Große Software-Hersteller bedienen sich deshalb der Hilfe eigener Patentabteilungen. Über solche Unterstützung verfügen jedoch die meisten Open Source-Programmierer nicht.

Unabhängige Entwickler, kleine und mittlere Unternehmen können leicht durch Patentverletzungsklagen aus dem Markt gedrängt werden. Die Anwaltskosten, um sich dagegen zu verteidigen, sind für sie untragbar.

Empfehlung

Aus diesem Grunde möchte ich eine Empfehlung aussprechen. Diese Empfehlung wurde erstmals in einem Gutachten unterbreitet, das im Auftrag des Wirtschaftsministeriums die Auswirkungen des Patentschutzes für Software auf die IT-Sicherheit untersuchte. (Lutterbeck et al. 2000)

Die Kernempfehlung des Gutachtens lautet: (Empfehlung PP-1):

«PP 1: Der Umgang mit dem Quelltext von Computerprogrammen muss patentrechtlich privilegiert werden. Das Herstellen, Anbieten, in Verkehr bringen, Besitzen oder Einführen des Quelltextes eines Computerprogrammes in seiner jeweiligen Ausdrucksform muss vom Patentschutz ausgenommen werden. (Quelltextprivileg)»

Damit komme ich zum Ende meiner Ausführungen und möchte noch einen Wunsch an all jene richten, die an Fragen der IT-Sicherheit interessiert sind.

Wir sollten es nicht zulassen, daß die Open Source-Entwicklung durch den wirtschaftlichen Egoismus von Patentinhabern zerstört wird. Im Interesse der Sicherheit unserer Informationstechnologie sollten wir uns den Weg nicht verbauen lassen.

Referenzen

Akerlof, George A.: **The Markets for "Lemons" – Quality Uncertainty and the Market Mechanism**, in: *Quarterly Journal of Economics*, 1970, Vol. 84, No. 3, pp 488-500.

Anderson, Ross: **Why Information Security is Hard – An Economic Perspective**, 2001, on the internet: http://www.cl.cam.ac.uk/ftp/users/rja14/econ.pdf [28 Aug 2001].

Dibona, Chris; Ockman, Sam; Stone, Mark: **Open Sources. Voices from the Open Source Revolution**, O'Reilly, Sebastopol, CA, 1999.

Floyd, Christiane: **Softwaretechnik** [Software engineering], **14.2.1 Eigenschaften von Software** [Properties of software], in: P. Rechenberg, G. Pomberger (eds.): Informatik–Handbuch, Carl Hanser Verlag, München, Wien, 1997.

Forno, Richard: **Microsoft: A Proven Danger to National Security: Conclusions of Reality**, Essay #2000-4, 15 May 2000, on the internet via: http://www.info-sec.com/internet/00/MSFOR_natsec.pdf [17 Oct 2001].

Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John: **Design Patterns. Elements of Reusable Object-Oriented Software**, Addison-Wesley, Reading, MA, 1995.

Gehring, Robert A.: **"Software Patents" – IT-Security at Stake?** Paper and presentation prepared for the international congress "Innovations for an e-Society. Challenges for Technology Assessment", October 17-19, 2001, Berlin, Germany, on the internet: http://ig.cs.tu-berlin.de/ap/rg/2001-10/index.html [18 Jan 2002].

IEEE U.S.A.: **Opposing Adoption of the Uniform Computer Information Transactions Act (UCITA) By the States**, Approved By the IEEE-USA Board of Directors, Feb. 2000(a), on the internet: http://www.ieeeusa.org/forum/positions/ucita.html [28 Aug 2001].

IEEE U.S.A.: **Recommended Amendments to Virginia Uniform Computer Information Transactions Act**, (Title 59.1, Chap. 43, § 59.1-501.1 et. seq.), 17 Oct 2000(b), on the internet: http://www.ieeeusa.org/forum/POLICY/2000/00oct17.html [28 Aug 2001].

Kaner, Cem: **Liability for Defective Content**, in: *Software QA*, 1996, Vol. 3, No. 3, p. 56, on the internet: http://www.badsoftware.com/badcont.htm [28 Aug 2001].

Kaner, Cem: **The Impossibility of Complete Testing**, in SOFTWARE QA, Volume 4, #4, p. 28, 1997(a), on the internet: http://www.kaner.com/articles.html [28 Aug 2001].

Kaner, Cem: **Software Liability**, 1997(b), on the internet: http://www.kaner.com/articles.html [28 Aug 2001].

Kaner, Cem: **The Problem of Reverse Engineering**, in *SOFTWARE QA*, Vol. 5, #5, 1998, on the internet: http://www.kaner.com/articles.html [28 Aug 2001].

Levinson, Meredith: **Let's Stop Wasting $78 Billion a Year**, in: *CIO Magazine,* October 15, 2001, on the internet: http://www.cio.com/archive/101501/wasting_content.html?printversion=yes [16 Oct 2001].

Litman, Jessica: **Digital Copyright**, Prometheus Books, Amherst, NY, 2001.

Lutterbeck, Bernd; Horns, Axel H.; Gehring, Robert A.: **Sicherheit in der Informationstechnologie und Patentschutz fuer Softwareprodukte - ein Widerspruch ?** [Security in Information

Technology and Patent Protection for Software Products: A Contradiction?, Short Expertise Commissioned by the Federal Ministry of Economics and Technology], Berlin, December 2000, on the internet: http://www.sicherheit-im-internet.de/download/Kurzgutachten-Software-patente.pdf [28 Aug 2001].

McAfee: **McAfee Lösungen**, 2001, on the internet: http://www.mcafeeb2b.com/international/germany/products/mcafee-solutions.asp [23.08.2001].

Myers, Glenford J.: **Software Reliabilty. Principles and Practices**, John Wiley & Sons, New York, NY, 1976.

National Security Agency (NSA): **Security-Enhanced Linux**, on the internet: http://www.nsa.gov/selinux/ [28 Aug 2001].

Pfitzmann, Andreas; Köhntopp, Kristian; Köhntopp, Marit: **Sicherheit durch Open Source? Chancen und Grenzen** [Security by Open Source? Opportunities and Limitations], in: *Datenschutz und Datensicherheit*, 9/2000, pp 508-513.

Pipkin, Donald L.: **Information Security**, Prentice Hall PTR, Upper Saddle River, NJ, 2000.

Raskind, Leo J.: **Copyright**, in: The New Palgrave Dictionary of Economics and The Law, Vol. 3, p. 478ff, Macmillan Reference Ltd., London, 1998.

Raymond, Eric S.: **The Cathedral & The Bazaar. Musings on Linux and Open Source by an Accidental Revolutionary**, O'Reilly, Sebastopol, CA, 1999.

Reed Elsevier, Inc.: **Response to 65 FR 35673**, on the internet: http://www.loc.gov/copyright/reports/studies/dmca/reply/Reply005.pdf [29 Aug 2001].

Samuelson, Pamela; Davis, Randall; Kapor, Mitchell D.; Reichman, J.H.: **A Manifesto Concerning the Legal Protection of Computer Programs.** Symposium: Toward a third intellectual property paradigm, in: *Columbia Law Review* 94 (1994) no 8, p. 2308ff.

Schmidt, Jürgen: **Sicherheitsrisiko Microsoft. Die Kehrseite des Windows-Komforts** [Security Risk Microsoft. The other side of Windows's comfort], pp 140-142, in: *c't* Magazin 21/2001.

Schneier, Bruce: **Secrets and Lies. Digital security in a networked world**, John Wiley & Sons, Inc., New York, 2000.

Shapiro, Carl; Varian, Hal R.: **Information rules. A strategic guide to the network economy**, Harvard Business School Press, Boston, MA, 1999.

Sommerville, Ian: **Software Engineering**, 6th edition (german translation), Pearson Studium, 2001.

Yoshida, Junko; Leopold, George: **Copy protection bill divides industry**, Hollywood, *EETimes*, October 1, 2001, on the internet: http://www.eetimes.com/story/OEG20010928S0110