

What does vulnerability and patch data say?

BY GUIDO SCHRYEN

Is Open Source Security a Myth?

DURING THE PAST few decades we became accustomed to acquiring software by procuring licenses for a proprietary, or binary-only, immaterial object. We regard software as a product we have to pay for, just as we would pay for material objects. However, in more recent years, this widely cultivated habit has begun to be accompanied by a software model characterized by software that comes with a compilable source code. This type of software is referred to by the term “open source software” (OSS).

While there is consensus that opening up source code to the public increases the number of reviewers,

the impact of open source on software security remains controversial. Proponents of OSS stress the strength of the resulting review process¹⁹ and argue in the sense of Raymond²⁰ that, “Given enough eyeballs, all bugs are shallow.” while some opponents follow the argument of Levy,¹¹ who remarks “Sure, the source code is available. But is anyone reading it?” Interestingly, Ozment and Schechter¹⁸ report that in the OpenBSD source, foundational vulnerabilities have a median lifetime of at least 2.6 years, which seems to refute Raymond’s argument.

While the security discussion is rife with beliefs and guesses, only a few quantitative models and some empirical studies appear in the literature.^{1,9,10,13,17,21,24,25} Many of these empirical studies investigate one package or a few software packages only, and to my best knowledge, no prior study has been conducted to comprehensively study differences between open source and closed source security. The reason why comprehensive empirical studies have been neglected is probably due to the need for laborious collection and analysis of reliable data and the associated manual work.

This article presents a comprehensive empirical investigation of published vulnerabilities and patches of 17 widely deployed open source and closed source software packages. The empirical analysis uses comprehensive vulnerability data contained in

» key insights

- **The security discussion of open source and closed source software is rife with beliefs and guesses. Data-driven insights based on an empirical analysis, as examined here, provide new insight into such security issues.**
- **The analysis illustrates there is no empirical evidence that the particular type of software development is the primary driver of security. Rather, the particular policies of vendors determine the patching behavior.**
- **It is most important to provide economic incentives for software producers to make software less vulnerable and to provide patches.**

the NIST National Vulnerability Database¹⁴ and a newly compiled data set of vulnerability patches. Based on these comprehensive data sets, this study is capable of providing empirical evidence that open source and closed source software development do not

significantly differ in terms of vulnerability disclosure and vendors' patching behavior, a phenomenon that has been widely assumed, but hardly investigated.

Open and Closed Source Software

Generally, the availability of source code to the public is a precondition for software being denoted as "open source software." Beyond this requirement, the Open Source Initiative (OSI) has defined a set of compliance criteria for open source software.¹⁵ The (open source) definition (OSD) includes permission to modify the code and to redistribute it. However, it does not govern the software development process in terms of who is eligible to generate and to modify software. In this regard, two options are distinguished by Raymond:²⁰ When what is called "bazaar style" is in place, any volunteer can provide source code submissions. In a more closed environment, software is crafted by individual wizards, and the development process is characterized by a relatively strong control of design and implementation. This style is referred to as "cathedral style."

The OSI approved several licenses, including the Apache License, the BSD license, and the GNU General Public License (GPL), which is maintained by the Free Software Foundation (FSF). The FSF provides a definition of "free software" [as] a matter of liberty, not price."⁸ In contrast to the OSD definition, the FSF definition explicitly focuses on the option of releasing the improvements to the public. Software is usually regarded as "closed" if the source code is not available to the public.

The categorization of software and its development process as "open source software (development)" or "free software (development)" in contrast to "closed source software (development)" mirrors the perspective of developers and specifies the type of development. Complementarily, one could also adopt the software users' point of view by distinguishing between software that is charged for and software that is free of charge. The resulting classification scheme is shown in Table 1.

The Life Cycle of Vulnerabilities and Patches

When software is executed in a way which is different from that which the original software designers intended, this misbehavior is rooted in software bugs. Anderson³ assumes the ratio between software bugs and software lines of code (SLOC) to be about 1:35. When bugs can be directly used by at-

Table 1. Classification of software.

Users' perspective	Developers' perspective	
	Open Source (license)	Closed Source
Free of charge	Linux, Apache web server	Adobe Acrobat Reader
Subject to charge	MySQL (dual licensing ¹)	Microsoft Windows operating systems

¹ "[D]istributors that...do not wish to distribute the source code for the commercially licensed software under version 2 of the GNU General Public License (the "GPL") must enter into a commercial license agreement with Sun." (<http://www.mysql.com/about/legal/licensing/oem/>)

Figure 1. Vulnerability life cycle.

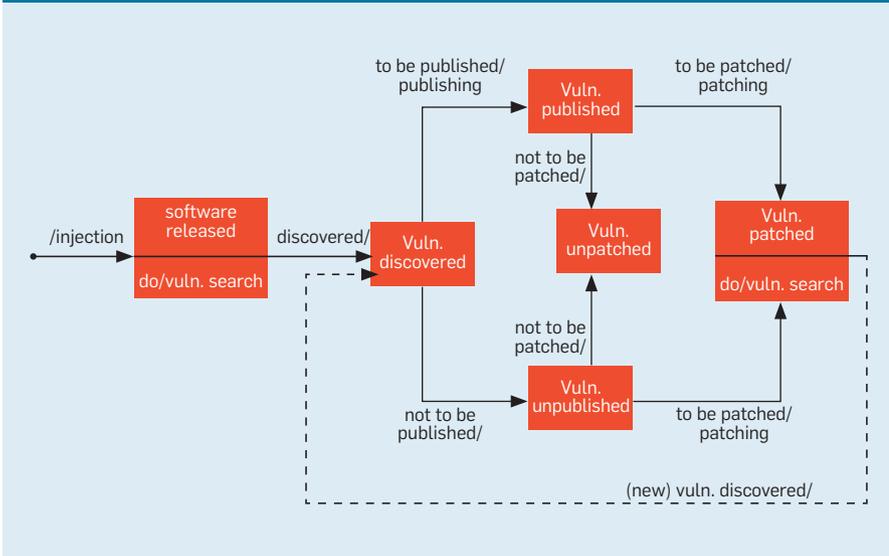
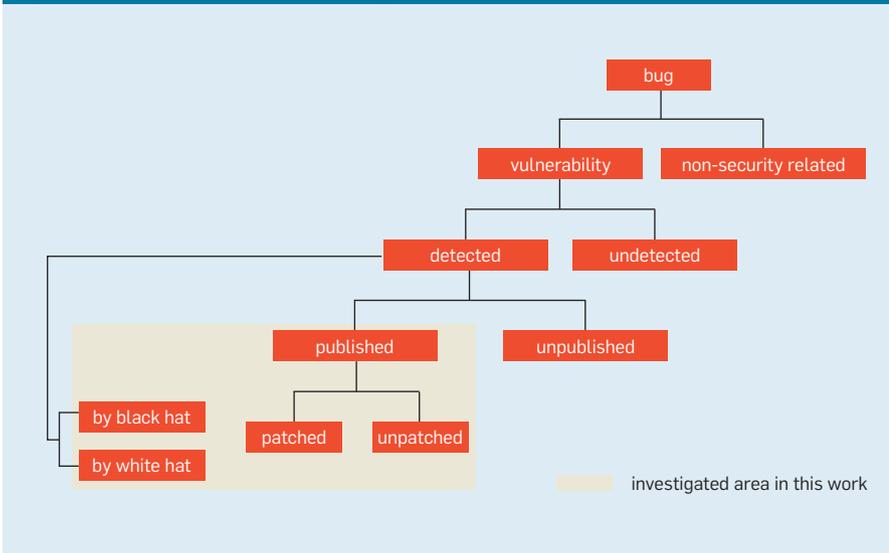


Figure 2. Classification of software bugs and vulnerabilities.



tackers to gain access to a system or network, they are termed “vulnerabilities” by the U.S. MITRE Corporation.¹² Although there are other definitions of “vulnerabilities,”^{16,23} the adoption of the MITRE definition is useful in a pragmatic sense for three reasons:

- ▶ Most empirical studies implicitly use this definition by analyzing “Common Vulnerability and Exposures (CVE)” entries, which are provided by MITRE. CVE identifiers are not only widely used by researchers, they are also used by information security product/service vendors. Thereby, the CVE definition has become a de facto standard.

- ▶ The process of accepting a potential software bug as a CVE vulnerability is well documented, and the assessment is conducted by security experts.¹²

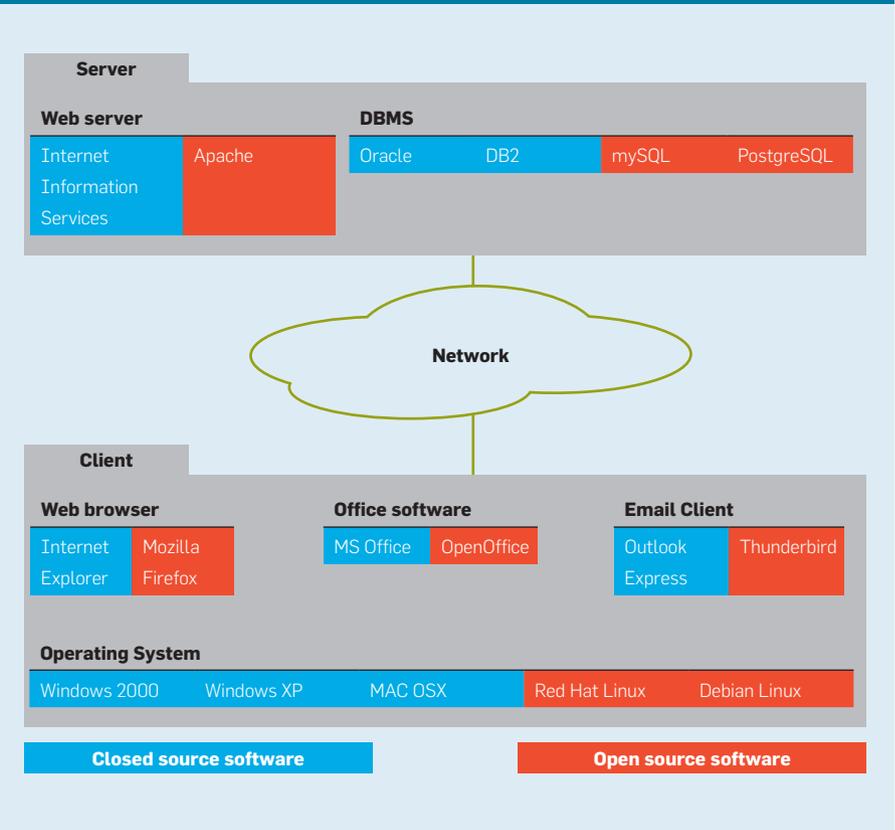
- ▶ The U.S. National Institute of Standards and Technology (NIST) adopts the MITRE understanding of vulnerabilities in its National Vulnerability Database (NVD), which is probably the largest database of security-critical software bugs and which provides comprehensive CVE vulnerability data feeds for automated processing.

Vulnerabilities and their dynamic behavior can be described through the “vulnerability life cycle,” which is shown in Figure 1 as a UML statechart diagram. The diagram provides a process-oriented perspective on a single vulnerability and its patch (for the consideration of exploits see the study of Frei⁹), integrates states that have been introduced by Arbaugh et al.,⁵ and uses a cycle to account for the fact that patching vulnerabilities can even create new vulnerabilities.⁵

Injection: The life cycle starts with the injection of a vulnerability into the software. In principle, a vulnerability can find its way into software through the intentional behavior of software developers, who strive to sell or exploit vulnerabilities, or to harm their employer, or unintentional behavior, which can be rooted in careless programming or in using “insecure” development tools. After testing, the software is finally released, the public starts searching for vulnerabilities, and the software vendor potentially continues searching.

Discovery and publication: The discovery of vulnerabilities can be based

Figure 3. Selected open source and closed source software packages.



on coincidental detection or on the active search of persons with intrinsic or with extrinsic motivation. The emergence of vulnerability markets,²² including bug auctions, bug challenges, and vulnerability brokers,⁷ provides economic incentives (at least for “white hats”) to search for and to disclose information on vulnerabilities.

When a vulnerability is discovered, the question of whether it should be published or not occurs. If a “black hat” detects the vulnerability, his or her decision depends on whether s/he aims at making the vulnerability available to as many other “black hats” as possible and to gain reputation, or to a closed group of potential attackers. If the vulnerability is detected by a “white hat,” including the software vendor, it is still not clear whether the vulnerability should be published or not, as vulnerability information is useful for both the “good guys,” who can provide patches, and the “bad guys,” who probably would not have gained knowledge of the vulnerability otherwise. Some researchers have addressed this question: Rescorla²¹ argues against disclosure, as he finds the probability of vulnerability rediscovery to be vanish-

ingly small. However, investigating the operating system OpenBSD, Ozment¹⁷ finds that vulnerabilities are correlated regarding their rediscovery, and argues in favor of disclosure.

Patching: Once a vulnerability is published, at first glance it seems obvious that the vendor should provide a patch as soon as possible. But it can be economically reasonable for the vendor not to provide a patch when it is the customers who suffer the most cost of failure and when competitors behave likewise. If the vulnerability is not published (and detected by “white hats” other than the vendor), again, the question arises of whether the vendor should provide a patch or not. While the aforementioned economic argument still holds, the decision not to provide a patch may be rooted in the assumptions that a nonpublished vulnerability is hardly exposed to attacks; any vulnerability disclosure reduces the vendor’s reputation; and the patch reveals the vulnerability to attackers, who then try to compile exploits and to use them to attack unpatched systems.

When a vulnerability patch is available, the search for newly injected vulnerabilities starts since it is known

that patches can contain new vulnerabilities.⁶ As the injection refers to a new vulnerability, Figure 1 shows a dashed line. An overview of the classification of vulnerabilities is provided in Figure 2.

The previous discussion of the life cycle stresses that the empirical security of software goes beyond technological phenomena and also depends on economic conditions. In the particular context of open source and closed source software, Anderson² shows that although, under ideal conditions, open and closed systems are equally secure, this symmetry can be broken owing to economic phenomena.

Investigated Software Packages

In order to draw a picture of empirical open source and closed source

software security, it seems alluring to consider as many software packages and vulnerability data as possible. But this (quantitative) approach suffers from at least two limitations. First, for many software packages only (too) few vulnerability data is available, as the packages are rarely deployed and probably hardly attractive for attackers. Second, a comparison of open source and closed source software remains strongly biased, unless the software packages under consideration are comparable in terms of functionality. However, for many open source and closed source software packages, no functional counterparts are available.

Due to these issues, I decided to follow a qualitative approach, and to manually select widely deployed soft-

ware packages for the empirical analysis. Assuming that most software is usually attacked through the Internet, I adopt the client-server perspective to frame the selection of software packages (see Figure 3). On the client side, the most widely deployed operating systems (OS) are Microsoft OS, *MAC OS X* and Linux derivatives (<http://marketshare.hitslink.com>). Among the Microsoft OS, Windows 2000, Windows XP, and Windows Vista are the leading ones in terms of market share, but I excluded the latter due to its short history (release date: January 30, 2007). Regarding Linux, I selected Red Hat Linux and Debian Linux, which are widely deployed Linux distributions. In addition to operating systems, I analyze web browsers, email clients, and office software. Regarding web browsers, Internet Explorer and Firefox are the most widely used programs (<http://marketshare.hitslink.com>), regarding email clients and office software, I found no reliable statistics. I selected Outlook Express and Thunderbird, which are comparable in terms of functionality in contrast to *Outlook*, which integrates much more functionality, and MS Office (Word, Excel, and Powerpoint) and OpenOffice.

On the server side, I analyze Web servers and (relational) database management systems (DBMS), which are widely used application types. Internet Information Services and Apache are the most frequently used Web servers (http://news.netcraft.com/archives/web_server_survey.html). Oracle and DB2 are two of the mostly used closed source DBMS (<http://www.gartner.com/it/page.jsp?id=507466>), while for open source DBMS no reliable data could be found. Having explored database-related Web sites, I decided to use MySQL and PostgreSQL, which are widely deployed. The specific versions of the software packages are given in Table 2.

Vulnerabilities

Having decided to adopt the vulnerability definition of the MITRE CVE group (discussion earlier), the question remains of how to gather CVE data as the CVE group “...only contains the standard identifier number with status indicator, a brief description, and references to related vulnerability reports and advisories” (<http://cve.mitre.org/>)

Table 2. Vulnerability data.

Application Type	Product	Devel. Type	Release date	#vuln	MTBVD [days]	Development of vulnerability disclosure over time
Browser	Internet Explorer 7	Closed	2006-10-18	74	13.29	Linear
	Firefox 2	Open (BS)	2006-10-24	167	5.16	Linear
Email client	MS Outlook Express 6	Closed	2001-10-25	23	120.73	Linear
	Thunderbird 1	Open (CS)	2004-12-07	110	13.79	Not linear
Web server	IIS 5	Closed	2000-02-17	83	40.90	Not linear
	Apache2	Open (CS)	2000-03-10	80	40.63	Linear
Office	MS Office 2003	Closed	2003-11-17	99	19.22	Not linear
	OpenOffice2	Open (CS)	2005-10-20	19	63.16	Linear
Operating system	Windows 2000	Closed	2000-02-17	385	9.35	Linear
	Windows XP	Closed	2001-10-25	297	8.97	Linear
	Mac OS X 10.x	Closed ²	2005-04-29	300	4.64	Linear
	Red Hat Enterprise Linux 4	Open (CS)	2005-02-14	264 ¹	5.48	Not linear
	Debian 3.1	Open (BS)	2005-06-06	207 ¹	6.45	Linear
Database Management System	MySQL 5	Open (BS)	2005-10-24	33	46.00	Linear
	PostgreSQL 8	Open (CS)	2005-01-19	25	58.96	Linear
	Oracle 10g	Closed	2004-01-15	63	29.72	Not linear
	DB2 v8	Closed	2004-03-26	13	136.38	Linear

BS: Bazaar style CS: Cathedral style

¹ The NVD lists linux kernel vulnerabilities separately from vulnerabilities of specific Linux distributions. Red Hat Enterprise Linux (RHEL) 4 uses kernel 2.6.9, (http://www.linuxcompatible.org/Red_Hat_Enterprise_Linux_4_Nahant_Beta_2_Public_Availability_s36797.html), Debian 3.1 uses kernels 2.4.27 or 2.6.8 (<http://www.debian.org/News/2005/20050606>). I consider only those kernel vulnerabilities that were published after the release date of RHEL 4 and Debian 3.1, respectively.

² Some open source components are included.

about/faqs.html). I decided to use the NIST National Vulnerability Database (NVD) (<http://nvd.nist.gov/>)¹⁴, which provides full database functionality for the complete MITRE CVE dictionary. Information on MITRE CVE vulnerabilities and the NIST NVD is provided on the Web sites of MITRE¹² and NIST,¹⁴ respectively.

Vulnerability data. I discuss the quality of used data and the implications of how the MITRE CVE dictionary and the NIST NVD are built for the analysis of vulnerabilities. As the data sources of the CVE group are manifold and include trustworthy organizations, such as US-CERT and SecurityFocus, the CVE input can be assumed to be comprehensive, although it cannot be guaranteed that all disclosed vulnerabilities are considered. The analysis of potential vulnerabilities by the MITRE CVE group assures that each CVE candidate has been inspected by security professionals. In cases where software vendors dispute vulnerabilities, I chose to use MITRE data in favor of an unbiased assessment. Overall, the CVE dictionary is a valuable resource for vulnerability analysis in terms of both quantity and quality. As the NVD acquisition procedure considers all CVE vulnerabilities in a timely manner and provides them in xml data feeds, the NVD is an appropriate database for the analysis of vulnerabilities in general. However, while the NVD provides a comprehensive database of (CVE) vulnerabilities, the properties of some vulnerability attributes added by NVD analysts need some more attention:

► **Original release date (ORD):** Owing to two potential time gaps, the ORD assigned to a CVE identifier by the NVD does not necessarily mirror the actual date of disclosure: Time between the actual disclosure of a vulnerability and its consideration in the “Assigned” phase of the MITRE CVE workflow. This gap is zero when the vulnerability has not been disclosed to the public. Moreover, time between the “Assigned” date and the NVD publication date. As no information on the overall time gaps is available, the computation of patch times and exploit times would contain errors of unknown size. However, I assume the effect of the errors on the particular types of development of vulnerability disclosure over time

(for example, linear or S-shape) to be less important, as this development is not affected by the full time gaps, but only by differences between the time gaps (standard deviation of time gaps). In addition, unusually large time gaps would be detectable in the graphics shown in Figure 4.

► **Common Weakness Enumeration (CWE):** The NVD analysis team assigns a type (for example, buffer overflow) to a vulnerability, based on a subset of the MITRE CWE structure. However, by Dec. 31, 2008, only about one fourth of all NVD CVE entries (9,748 out of 34,091) contained a CWE name so that an analysis of vulnerability types is not reasonable.

► **Common Platform Enumeration (CPE):** The NVD applies the structured naming scheme CPE, provided by MITRE, to assign names of vulnerable product versions to CVE identifiers.

► **Common Vulnerability Scoring System (CVSS):** Vulnerabilities are scored by the NVD analysis team regarding their severity. CVSS 2.0 provides for three scores, with each score value being between 0 and 10 (highest severity). The base score is an aggregation of six base score metrics. This score is mandatory, and specified by vulnerability bulletin analysts and software vendors. The NVD team works closely with the CVSS working group, MITRE, public vulnerability sites, vendors and security researchers to come to a consensus on scoring some of the more commonly occurring vulnerabilities. Other score types are the temporal score and the environmental score. For our analysis, only the base score is applicable. It should be noted that the NVD scoring system changed over time: CVSS 2.0 scores for the CVE vulnerabilities published prior to 9/11/2005 were converted by the NVD team from prior CVSS metric data. The investigation of the NVD conversion script provided by the NVD Program Manager (C. Johnson) reveals that for all CVSS 2 characteristics corresponding CVSS 1 characteristics are available, and a “natural” conversion was conducted. To sum up, older scores that have been converted into CVSS 2 are comparable with “new” CVSS 2 scores. Based on the aforementioned analysis, I regard CVSS 2.0 scores to be useful for further analysis.

The following analysis of NVD vul-

nerabilities is based on NVD xml data feeds as available on Jan. 31, 2009. All feeds were imported into MS Office Excel 2007 and processed using filters and MS Query. In order to assure that vulnerabilities listed in the NVD data feeds have not been accidentally misattributed regarding the affected software version, I double-checked the affected software versions of each vulnerability on the Web sites of vendors, MITRE, and SecurityFocus. In very few cases of inconsistencies, I excluded the particular vulnerability from any further analysis.

Mean time between vulnerability disclosures. Table 2 lists for each software package the number of published vulnerabilities and the “mean time between vulnerability disclosures” (MTBVD) defined—analogously to the software engineering term “mean time between failures”—as the average number of days since software release divided by the number of published vulnerabilities. With regard to determining the MTBVD, I consider only those vulnerabilities that have been published after the release date. Vulnerabilities that have been published earlier than the release date and that also affect the version under consideration are due to the development process of earlier versions. A simple comparison of the MTBVD of software packages does presumably not provide reliable results regarding the level of security because of two reasons: the severity of vulnerabilities should be considered, for example by using vulnerability weights, and the vulnerability detection and publication are probably correlated with market and software factors. For example, an important market factor is the attractiveness of the software for “vulnerability searchers,” an important software factor is software size, as given by “software lines of code” (SLOC). However, reliable and precise figures are available for few software packages only, so that figures must not be used for a fair comparison of security levels. On the other hand, MTBVD data is not useless as it provides an impression of how often vulnerabilities are published and how insecure especially operating systems are, regardless of the particular development style. Overall, data suggests the vulnerability publication rate is

probably determined by other factors than the software development style.

Development of vulnerability disclosure over time. While the previous discussion provides a static picture of the history of vulnerabilities, I now address the development of vulnerabilities over time. In an earlier empirical investiga-

tion, Alhazmi et al.¹ assume the development of vulnerability discovery can be described by an “S” shape. However, the analysis of vulnerability data shows that for 12 of 17 considered software packages a linear correlation between time and the number of published vulnerabilities is found (see Table 2 and

Figure 4). This supports Rescorla’s argument that the rate of vulnerability finding is constant over long periods of time. The reason why five packages show a different behavior is not clear. Interestingly, three closed source packages (MS Office, Oracle, and DB2) and one open source package (Apache) show no vulnerability disclosure during the first 316, 202, 460, and 367 days, respectively, while for all other packages vulnerabilities were published quickly after their release. The available vulnerability data explains these large time gaps only partially:

- ▶ In contrast to the other three packages, Apache 2 was a substantial rewrite of much of the Apache 1.x code. As knowledge of vulnerabilities of Apache 1.x was of very limited use for finding vulnerabilities in Apache 2, the discovery of the first vulnerabilities could be expected to take a while, but it seems questionable as to whether this explains a one-year gap. Data on published vulnerabilities does not tell us whether the quality of code prevented vulnerabilities from being detected quickly or whether discovered vulnerabilities were published later or are even still unpublished.

- ▶ In the case of DB2, the software showed only 13 vulnerabilities, so that a large time gap is less surprising.

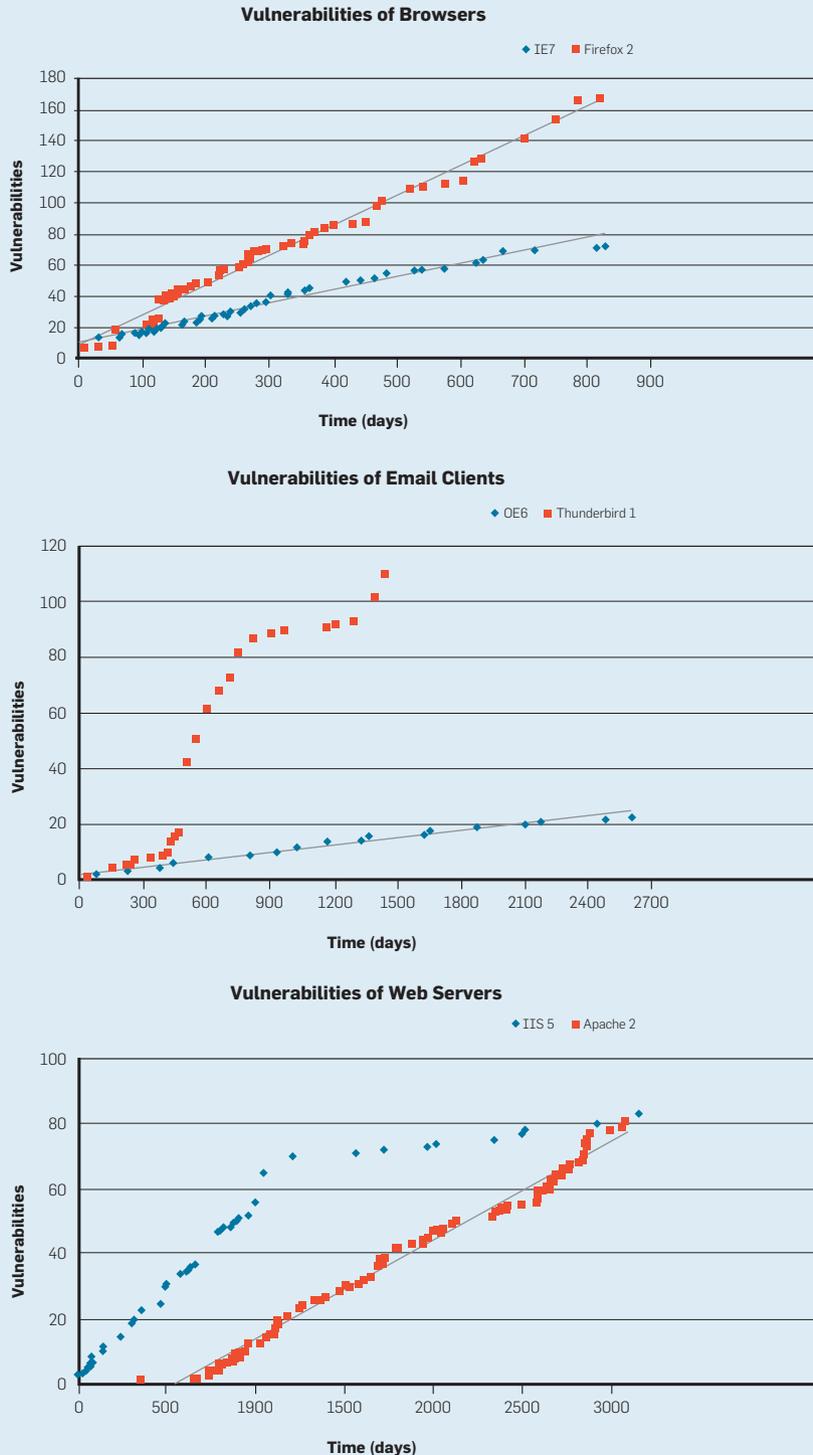
- ▶ Regarding Oracle, the NVD published the first (ten) vulnerabilities altogether on Aug. 4, 2004. It seems reasonable to assume that these published vulnerabilities had been detected much earlier and that their publication was delayed either by Oracle, MITRE, or NIST NVD.

- ▶ The case of Office 2003 showed none of the aforementioned reasons. Vulnerability data does not reveal why the first vulnerabilities were published so late.

Despite the aforementioned minor issues in analyzing vulnerability data, overall there is no observable difference between open source and closed source software with regard to the (type of) development of vulnerabilities over time, and there is also no observable difference between open source software developed in bazaar style and open source software developed in cathedral style.

Severity of published vulnerabilities. Having considered the number of

Figure 4. Development of vulnerability disclosure over time.



(continued on next page)

vulnerabilities, I now analyze whether open source and closed source software differ in the severity of published vulnerabilities. This perspective is important as well, because a single highly severe vulnerability that enables attackers to get root access to a system is probably more crucial than 10 low severe vulnerabilities that grant only reading access to unauthorized users. I analyze the severity of vulnerabilities for each software package in terms of the median and the proportion of highly severe vulnerabilities. Mean, standard deviation and, for each application type, the median of medians is also given (see Table 3). The analysis provides the following results:

► The medians of medians reveal that the vulnerabilities of office products are much more severe (8.45) than those of Web servers (5.0), while the values of the other application types are close to each other. However, a statistical analysis of the medians is not reasonable here due to the low number of values.”

► When we determine the medians of medians of open source software (5.7) and closed source software (6.8) and also the corresponding medians of the proportions of highly severe vulnerabilities (30.28% and 45.95%, respectively), the first impression is that open source software is more secure in terms of the severity level. However, applying statistical analysis (Mann-Whitney U-test) on the medians,^a no statistically significant differences can be found: the two-tailed test provides a high number for p (p=0.11). Applying the same test to the proportion figures, the test, again, does not indicate that the samples are significantly different at the 0.05 level (p=0.06).^b

a Although the severity of a vulnerability is given by a number, this number is at ordinal scale level only (due to the characteristics of the CVSS evaluation process). Consequently, variances of the samples cannot be determined.

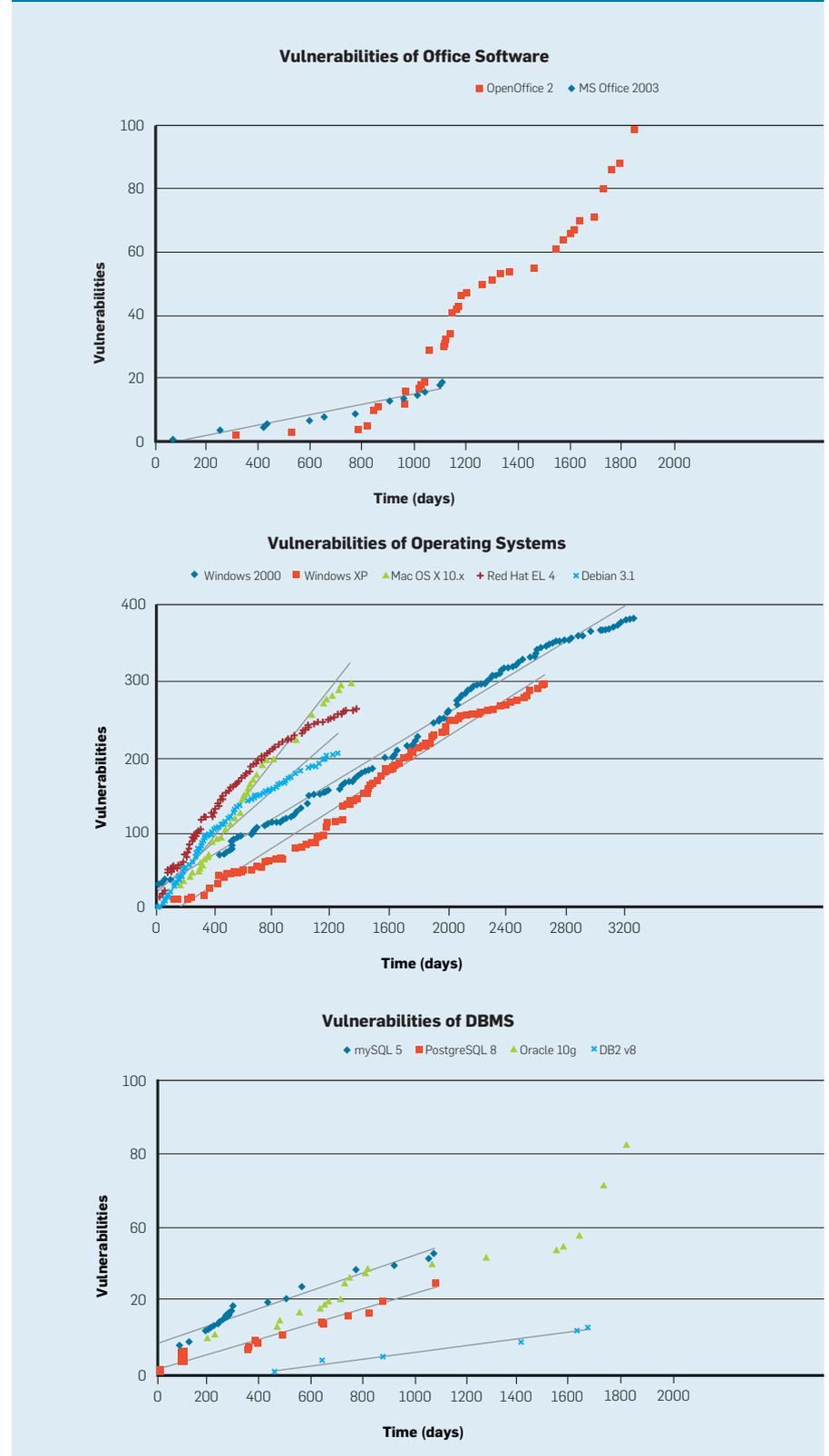
b In contrast to severities of vulnerabilities, proportions of highly severe vulnerabilities are at cardinal level. As the equality of the sample variances cannot be rejected at the 0.05 level ($F_{\text{sample}} = 1.62 < 3.5 = F_{(7,8)}$), the assumption that samples are normally distributed cannot be rejected at the 0.05 level (Kolmogorov-Smirnow test), we assume that variances are approximately equal so that the Mann-Whitney U-test can be applied.

Vendors' Patching Behavior

Patch data. While the analysis of vulnerabilities and their publication refers to the first three phases of the software vulnerability life cycle, and thereby mirrors the behavior of soft-

ware communities in terms of creating, detecting, and publishing vulnerabilities, the investigation of the provision of patches aims at identifying how vendors behave in actively addressing and finally removing vulnerability is-

Figure 4 (continued from previous page). Development of vulnerability disclosure over time.



sues. In order to detect differences in the patching behavior of open source and closed source vendors, I analyze how many of the vulnerabilities remained unpatched and whether any correlation between the patch status and the severity of vulnerabilities exists. Although vendor sites provide patch dates, I do not analyze the time gap between vulnerability disclosure and vendors' provision of patches, as the vulnerability publication dates contained in the NVD do not necessarily give the actual publication date (discussion earlier). In contrast to vulnerability publication data, reliable data on patches can be (manually) collected by directly looking up vendors' sites and vendor-neutral Web sites. More specifically, I used the following data sources to obtain reliable patch data: NVD, MITRE site, US-CERT Vulnerability Notes Database, SecurityFocus, Microsoft Security Bulletins, OpenOffice.org, The Open Source Vulnerability Database, The X-Force database (IBM), Mozilla Foundation Security Advisories, Red Hat Network, Apache Security Reports, Apple Mailing Lists, IBM FixPaks, VU-PEN Security, MySQL Forge, and Oracle Security Alerts and Patch Updates.

The newly compiled data pool contains patch data on the aforementioned browsers, email clients, Web servers, office products, operating systems, and database management systems. In cases where vulnerabilities have not been patched by those vendors regarded as "responsible" by NIST (NVD), I manually checked the MITRE description of these vulnerabilities. On the basis of this inspection I decided whether it's the responsibility of a particular vendor to provide a patch or not. In the few cases where I regarded the assigned vendor as not being responsible, I did not count the vulnerability as unpatched (by that particular vendor).

(Un)patched Vulnerabilities

Table 4 shows aggregated patch data for each software package. Vulnerabilities for which I could not find any patch information by Feb. 28, 2009 are classified as unpatched. As I used NVD xml data feeds as available on Jan. 31, 2009, the analysis considers a time gap of four weeks in order to account for delays in vendors' patching behavior, such as those due to "patch release days."

It is remarkable to see that 17.6% (30.4%) of the published open (closed)

source software vulnerabilities (in terms of the median) are still unpatched. However, standard deviations differ enormously (6.9% and 24.7%, respectively). Apparently, the proportion of still unpatched vulnerabilities largely depends on the specific vendor. I discuss this behavior in detail below.

Interestingly, the case of Microsoft also shows that even the same vendor can display different patching behavior, depending on the particular application type: while only 4% of MS Office 2003 vulnerabilities remain unpatched, one out of three vulnerabilities of both operating systems remain unpatched, half of the vulnerabilities of IIS are still open, and even two out of three vulnerabilities of the Internet clients remain unpatched. The case of operating systems shows that the proportion of unpatched vulnerabilities of software cannot be explained by simply considering the number of vulnerabilities, it rather depends on the vendors' patching priorities.

Severity of (un)Patched Vulnerabilities

It is interesting to compare the severity median of unpatched vulnerabilities with the median of patched vulnerabilities, in order to detect vendors' patching priorities, and to detect differences between open source and closed source software. The data in Table 4 reveals that, for all six Microsoft products, there is a strong bias toward patching severe vulnerabilities. This result indicates that Microsoft decides to leave less severe vulnerabilities unpatched, probably because the economic efforts would not be compensated by the (minor) gain in software security. However, on the other hand, the result also shows that Microsoft is interested in patching the most severe vulnerabilities, which reveals that software security is regarded to be a serious market issue. Apple (MAC OS X) shows a similar behavior in their operating system in terms of the severities of patched and unpatched vulnerabilities, but, in contrast to Microsoft, Apple seems to be interested in patching most of the vulnerabilities. We find this strong interest in patching vulnerabilities also in the cases of Oracle and IBM (DB2), but the severity medians of

Table 3. Severity of published vulnerabilities.

Application Type	Product	Severity (range=[0;10])				
		mean	median	std. dev.	Proportion of highly severe vulnerabilities ([7;10])	Median of medians
Browser	Internet Explorer 7*	6.65	6.80	2.07	45.95%	6.6
	Firefox 2**	6.38	6.40	2.11	36.53%	
Email client	MS Outlook Express 6*	6.18	5.10	1.76	39.13%	5.95
	Thunderbird 1**	6.53	6.80	2.23	47.27%	
Web server	IIS 5*	6.00	5.00	1.55	36.14%	5.00
	Apache2**	5.36	5.00	1.50	18.75%	
Office	MS Office 2003*	8.11	9.30	1.91	67.72%	8.45
	OpenOffice2**	7.61	7.60	1.79	63.16%	
Operating system	Windows 2000*	6.58	7.20	2.10	57.92%	6.8
	Windows XP*	6.67	7.20	2.16	58.92%	
	Mac OS X 10.x*	6.18	6.80	2.13	41.33%	
	Red Hat Enterprise Linux 4**	4.72	4.90	2.20	23.11%	
	Debian 3.1**	4.75	4.90	2.21	23.19%	
Database Management System	mySQL 5**	5.05	4.90	2.02	12.12%	6.15
	PostgreSQL 8**	6.17	6.80	1.89	36.00%	
	Oracle 10g*	5.96	5.50	2.05	33.33%	
	DB2 v8*	6.22	7.2	2.75	53.85%	

* closed source software median of medians = 6.8
 ** open source software median of medians = 5.7

unpatched vulnerabilities are higher than those of the patched ones. To sum up, three out of four closed source software vendors leave only a few vulnerabilities unpatched, while the other vendor focuses on patching more severe vulnerabilities.

Regarding the medians of patched and unpatched vulnerabilities of open source vendors, I do not find any pattern. In addition, the patching behavior of open source vendors shows that the proportion of unpatched vulnerabilities varies between 12% and 26.25%, and can differ considerably. On the other hand, none of the eight open source software packages shows an outlier, in contrast to closed source software.

As a result of the analysis of the patching behavior of software vendors, it turns out the behavior is not mainly determined by the particular software development style, but by the policy of the particular vendor.

Threats to Validity

Although the presented empirical study uses comprehensive data on vulnerabilities and patches of widely deployed software packages, and manual work was carried out in order to check and to improve data quality, some threats to the validity of results remain. First, the analysis investigates only those vulnerabilities that have been published as CVE vulnerabilities, that is, it excludes

vulnerabilities that have not been disclosed at all or have not been included in the MITRE CVE dictionary. Second, neither the “Assigned date” provided by MITRE nor the “Original release date” included in the NVD necessarily mirror the actual date of vulnerability disclosure to the public. I am not aware of any data sources that provide reliable and complete information on actual disclosure dates. Third, the CVE-to-vendor/product mapping (CPE) in the NVD is incomplete and not always clear with regard to which vendor is responsible for releasing a patch. The mapping was manually inspected by myself. In some cases, conflicts were resolved by adopting my point of view. Fourth, MTBVD values are limited in their validity, as missing market and software factors of packages (for example, market share, SLOC) would need to be considered.

Discussion

The analysis and comparison of open source and closed source software packages reveals that the type of software development is not the primary driver of software security in terms of the development of vulnerability disclosure over time, the severity of published vulnerabilities, and unpatched vulnerabilities and their severity. Consequently, we should explore other factors rather than asking whether open source or closed source software

leads to higher levels of security. One approach would be to investigate the technical roots of vulnerabilities, for example by adopting the “Common Weakness Enumeration” scheme provided by MITRE. This approach might help to explain differences between various software types with regard to the severity of vulnerabilities. Unfortunately, the NVD provides CWE values for only a few vulnerabilities, so that we need a more comprehensive data set. A second approach would be the comparison of MTBVD values by identifying and considering software and market factors. We could then apply regression models, and control for these factors. As the sound application of such models also requires the availability of the actual dates of vulnerability disclosures, we would need more precise data than is currently available. To sum up, we still face several data problems that impede the identification and explanation of differences in software security.

In contrast to vulnerability data, patch data is even more difficult to obtain. In order to facilitate further analysis and to avoid the tedious collection of patch information, comprehensive data pools would be useful. The analysis of vendors’ patching behavior shows a diffuse picture that misses clear patterns. We find patching behaviors that do not only differ between vendors but also between different products of the

Table 4. Patched and unpatched vulnerabilities.

Application Type	Product	Vulnerabilities (un)patched			Median of severities		
		#vuln.	#vuln. unpatched	Prop. of unpatched vuln.	unpatched	patched	overall
Browser	Internet Explorer 7	74	49	66.22%	5.0	9.3	6.8
	Firefox 2	167	34	20.36%	5.0	6.8	6.4
Email client	MS Outlook Express 6	23	15	65.22%	5.0	7.3	5.1
	Thunderbird 1	110	6	5.45%	3.45	6.95	6.8
Web server	IIS 5	83	40	48.19%	5.0	7.2	5.0
	Apache2	80	21	26.25%	4.7	5.0	5.0
Office	MS Office 2003	99	4	4.04%	5.05	9.3	9.3
	OpenOffice2	19	4	21.05%	5.25	9.3	7.6
Operating system	Windows 2000	385	117	30.39%	5.1	7.2	7.2
	Windows XP	297	91	30.64%	5.0	7.5	7.2
	Mac OS X 10.x	300	20	6.67%	5.0	6.8	6.8
	Red Hat Enterprise Linux 4	264	39	14.77%	4.9	4.9	4.9
	Debian 3.1	207	30	14.49%	4.9	4.9	4.9
Database Management System	mySQL 5	33	8	24.24%	4.6	4.9	4.9
	PostgreSQL 8	25	3	12.00%	9.0	6.3	6.8
	Oracle 10g	63	8	12.70%	7.35	5.5	5.5

same vendor (Microsoft). However, high figures of unpatched vulnerabilities show that exogenous incentives for software vendors to avoid vulnerabilities and/or to provide patches still need to be amplified, although a bias toward patching most severe vulnerabilities occurs. Economic countermeasures may provide such incentives.^{3,4}

Conclusion

This work has presented the first comprehensive empirical study on the security of open source and closed source software. It compared 17 well known and widely deployed software packages regarding published vulnerabilities and software vendors' patching behavior. The empirical results have shown that open source and closed source software do not significantly differ in terms of the severity of vulnerabilities, the type of development of vulnerability disclosure over time, and vendors' patching behavior. Although open source software development seems to prevent "extremely bad" patching behavior, overall there is no empirical evidence that the particular type of software development is the primary driver of security. Rather, the policy of the particular vendor determines the patching behavior. Consequently, in order to make software less vulnerable, it is most important to provide strong economic incentives for software producers to provide patches (at least for published vulnerabilities) or, even better, to avoid vulnerabilities from the outset.

Acknowledgement

The author would like to thank Christopher Johnson, Program Manager of the NIST National Vulnerability Database, for valuable discussions and the provision of background information, and Eliot Rich for his valuable support in writing this article. I appreciate the critical remarks of three anonymous reviewers. All errors are the sole responsibility of the author. □

References

1. Alhazmi, O., Malaiya, Y., Ray, I. Measuring, analyzing and predicting security vulnerabilities in software systems. *Computers & Security* 26, 3 (2007) 219-228.
2. Anderson, R. Open and closed systems are equivalent (that is, in an ideal world). *Perspectives on Free and Open Source Software*. J. Feller, B. Fitzgerald, S.A. Hissam, and K.R. Lakhani (eds). MIT Press, Cambridge, MA, 2005, 127-142.



It is still challenging to retrieve consistent and comprehensive vulnerability data and patch data. In order to facilitate further analysis and to avoid tedious data collection, comprehensive data pools would be useful.



3. Anderson, R. Why information security is hard—an economic perspective. In *Proceedings of the 17th Computer Security Applications Conference*, (New Orleans, LA, Dec. 10-14, 2001), 358-365.
4. Anderson, R. and Moore, T. Information security economics—and beyond. Information Security Summit 2008; http://www.cl.cam.ac.uk/~rja14/Papers/econ_czech.pdf.
5. Arbaugh, W.A., Fithen, W.L. and McHugh, J. Windows of vulnerability: A case study analysis. *IEEE Computer* 33, 12 (2000), 52-59.
6. Beattie, S., Arnold, S., Cowan, C., Wagle, P., Wright, C. and Shostack, A. Timing the application of security patches for optimal uptime. In *Proceedings of 16th Systems Administration Conference*, (Berkeley, CA, 2002), USENIX Association, 233-242.
7. Böhme, R. Vulnerability markets. What is the economic value of a zero-day exploit? In *Proceedings of 22nd Chaos Communication Congress*, (Berlin, Germany, Dec. 27-30, 2005).
8. Free Software Foundation (FSF). The free software definition; <http://www.fsf.org/licensing/essays/free-sw.html>, 2007.
9. Frei, S., May, M., Fiedler, U. and Plattner, B. Large-scale vulnerability analysis. In *Proceedings of the ACM SIGCOMM 2006 Workshop*, (Nov. 11, 2006, Pisa, Italy).
10. Gopalakrishna, R. and Spafford, E. H. A trend analysis of vulnerabilities. Technical Report 2005-05, CERIAS, Purdue University, May 2005.
11. Levy, E. Wide open source; <http://www.securityfocus.com/news/19>, 2000.
12. MITRE. Common vulnerabilities and exposures; <http://cve.mitre.org>, 2009.
13. Neuhaus, S., Zimmermann, T., Holler, C., Zeller, A. Predicting vulnerable software components. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS 2007)*, (Alexandria, VA, Oct. 2007), 529-540.
14. NIST. National Vulnerability Database; <http://nvd.nist.gov>, 2009.
15. Open Source Initiative (OSI). The Open Source Definition; <http://www.opensource.org/docs/osd>, 2006.
16. Ozment, A. Improving vulnerability discovery models: Problems with definitions and assumptions. In *Proceedings of the 3rd Workshop on Quality of Protection*, (Alexandria, VA, Oc. 29, 2007).
17. Ozment, A. The likelihood of vulnerability rediscovery and the social utility of vulnerability hunting. In *Proceedings of the 4th Workshop on the Economics of Information Security*, (Harvard University, June 2-3, 2005, Cambridge, MA), 1-21.
18. Ozment, A. and Schechter, S.E. Milk or wine: Does software security improve with age? In *Proceedings of the 15th Conference on USENIX Security Symposium*, (Vancouver, B.C., July 31-Aug. 4, 2006).
19. Payne, C. On the security of open source software. *Information Systems Journal* 12, 1 (2002), 61-78.
20. Raymond, E.S. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly, Beijing, China, 2001.
21. Rescorla, E. Is finding security holes a good idea? In *Proceedings of the 3rd Annual Workshop on Economics and Information Security*, (University of Minnesota, May 13-14, 2004).
22. Radianti, J., Rich, E. and Gonzalez, J.J. Vulnerability black markets: Empirical evidence and scenario simulation. In *Proceedings of the 42nd Hawaii International Conference on System Sciences*, (Big Island, Hawaii, Jan. 5-8, 2009).
23. US-CERT. Vulnerability notes database field descriptions (2009); <http://www.kb.cert.org/vuls/html/fieldhelp/>
24. Woo, S.-W., Alhazmi, O.H. and Malaiya, Y. K. An analysis of the vulnerability discovery process in Web browsers. In *Proceedings of the 10th International Conference on Software Engineering and Applications*, (Dallas, TX, Nov. 13-15, 2006).
25. Woo, S.-W., Alhazmi, O. H. and Malaiya, Y. K. Assessing vulnerabilities in Apache and IIS HTTP servers. In *Proceedings of the 2nd International Symposium on Dependable, Autonomic and Secure Computing*, (Indianapolis, IN, Sept. 29-Oct. 1, 2006), 103-110.

Guido Schryen (guido.schryen@wiwi.uni-regensburg.de) is a professor of information systems at the University of Regensburg, Germany.