Americas Conference on Information Systems (AMCIS)

AMCIS 2009 Proceedings

Association for Information Systems *Year* 2009

Security of Open Source and Closed Source Software: An Empirical Comparison of Published Vulnerabilities

Guido Schryen
International Computer Science Institute, schryen@gmx.net

Security of open source and closed source software: An empirical comparison of published vulnerabilities

Guido Schryen
International Computer Science Institute, Berkeley
schryen@gmx.net

ABSTRACT

Reviewing literature on open source and closed source security reveals that the discussion is often determined by biased attitudes toward one of these development styles. The discussion specifically lacks appropriate metrics, methodology and hard data. This paper contributes to solving this problem by analyzing and comparing published vulnerabilities of eight open source software and nine closed source software packages, all of which are widely deployed. It provides an extensive empirical analysis of vulnerabilities in terms of the mean time between vulnerability disclosures, the development of disclosure over time, and the severity of vulnerabilities, and allows for validating models provided in the literature. The investigation reveals that (a) the mean time between vulnerability disclosures was lower for open source software in half of the cases, while the other cases showed no differences, (b) 14 out of 17 software packages showed a significant linear or piecewise linear correlation between the time and the number of published vulnerabilities, and (c) no significant differences in the severity of vulnerabilities were found between open source and closed source software.

Keywords

Vulnerabilities, security, open source software, closed source software, empirical comparison

INTRODUCTION

Over the last few decades we have got used to acquiring software by procuring licenses for a proprietary, or binary-only, immaterial "object". We have come to regard software as a good we have to pay for just as we would pay for material objects, such as electronic devices, or food. However, in more recent years, this widely cultivated habit has begun to be accompanied by a new model, which is characterized by software that comes with a compilable source code (open source code). Often, such a source code is free of charge and may be modified and/or redistributed. The software type is referred to by the umbrella term "open source software". When discussing this alleged innovation in software distribution, we are reminded by (Glass, 2004) that, essentially, free and open source software dates right back to the origins of computing, as far back in fact as the 1950s, when all software was free, and most of it open. (Schwarz and Takhteyev, 2008) provide detailed insights into the history and the diffusion of open source software.

The application fields of open source software are manifold. Internet programs, such as the mail transfer agent *Sendmail* and the operating system *Linux* are some of the most popular examples. In the business sector, open source software is nowadays part of the core infrastructure of sophisticated technology companies, such as Amazon, Google, and Yahoo (Schwarz and Takhteyev, 2008). Obviously, open source software has arrived in the world of important and critical software environments that need security protection against attacks. Its increasing availability and deployment makes it appealing for hackers and others who are interested in exploiting software vulnerabilities, which become even more dangerous when software is not applied in a closed context, but interconnected with other systems and the Internet (this argument is valid for closed source software as well).

While there is consensus about the fact that opening source code to the public increases the potential number of reviewers, its impact on finding security flaws is controversially debated. Proponents of open source software stress the strength of the resulting review process (Payne, 2002) and argue in the sense of (Raymond, 2001) that, *"Given enough eyeballs, bugs are shallow."* (p. 19), while some opponents follow the argument of (Levy, 2000), who remarks *"Sure, the source code is available. But is anyone reading it?"* Interestingly, both parties essentially agree that open source basically makes it easy to find vulnerabilities; they only differ in their conclusions with regard to the resulting impact on security. For a detailed discussion of the arguments, see (Schryen and Kadura, 2009).

In order to have an unbiased discussion on open source and closed source security, it is helpful, if not necessary, to transparently measure the empirical security of software – be it open source or closed source software (Wolfe, 2007). However, measuring security is a challenging task, because security is somehow invisible. Despite the publication of an increasing number of quantitative research papers on measuring software security in the past years, what (Witten, Landwehr and Caloyannidis, 2001) observed is still true: what the discussion on software security specifically lacks is appropriate metrics, methodology and hard data.

Addressing this research gap, this paper analyzes and compares published vulnerabilities of eight open source software and nine closed source software packages, all of which are widely deployed. More specifically, this empirical study statistically analyses vulnerabilities in terms of the mean time between vulnerability disclosures, the development of disclosure over time, and the severity of vulnerabilities.

The rest of this paper is structured as follows: The next section presents the basic background on open and closed source software and work related to software vulnerabilities. Section 3 provides the methodology of this empirical study. The data used are described in Section 4. Section 5 presents the empirical findings, before Section 6 provides conclusions.

BACKGROUND AND RELATED WORK

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 criteria that software has to comply with (OSI, 2006). The definition includes the 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 modify the original version. When what is called "bazaar style" by (Raymond, 2001) is in place, any volunteer can provide source code submissions. Software development is then often based on informal communication between the coders (Gonzalez-Barahona, 2000). 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" (Raymond, 2001). The implementation of this modification procedure might have an impact on the security of software, so that a detailed discussion of open source security would need to take this into account.

A plethora of OSD-compliant licenses have come into operation, such as the *Apache License*, *BSD license*, and *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.*" (FSF, 2007). In contrast to the OSD definition, the FSF definition explicitly focuses on the option of releasing improved versions to the public (freedom 3), thereby rejecting the strong supervision of the modification process. Software is usually regarded as being "closed", if the source code is not available to the public.

Vulnerabilities

When software is executed in a way different from what the original software designers intended, this misbehaviour is rooted in software bugs. (Anderson, 2001) assumes the ratio between bugs and software lines of code (SLOC) to be about 1:35, i.e. Windows 2000 with its 35 Mio. SLOC would then have included one million bugs. The portion of bugs that are security-critical ("vulnerabilities") is assumed to be 1% (Anderson, 2001), resulting in an amazingly high figure of 350,000 vulnerabilities in Windows 2000. Detected vulnerabilities can further be divided into those being published and those that remain unpublished. An overview of the classification of bugs is provided in Figure 1, which also shows that in this paper only published vulnerabilities are considered.

Vulnerabilities are (software) product-related weaknesses, for which publicly accessible databases are available. They are the root for concrete security incidents (breaches), which are system-related and cause the actual harm. Breaches are much more difficult to investigate, because data is scarcer. For a detailed discussion of breaches, see (Jonsson, Strömberg and Lindskog, 2000; Kimura, 2006).

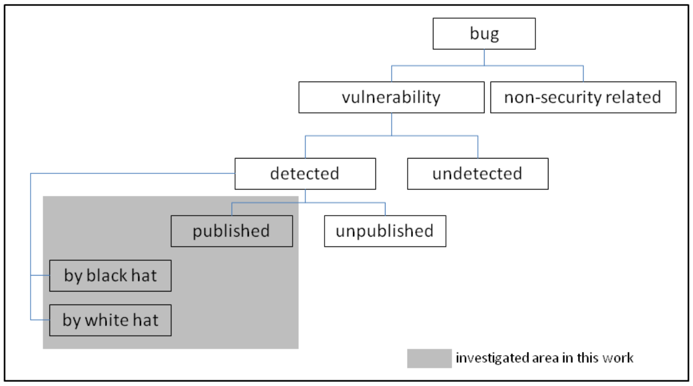


Figure 1. Classification of software bugs and vulnerabilities

(Alhazmi, Malaiya and Ray, 2005; Alhazmi, Malaiya and Ray, 2007) assume that the development of vulnerability discovery can be split up into three different phases. In phase 1, software testers gather sufficient knowledge of the system, which enables them to compromise the system. In phase 2, discovering vulnerabilities will be most rewarding for both white hat and black hat finders. Finally, in phase 3, vulnerability detection effort will then start shifting to the succeeding version of the software. These phases form an "S" shape that is assumed to follow the principle that the vulnerability discovery rate is linear in both the momentum gained by the market acceptance of the product and in the saturation of vulnerablity discovery due to a finite number of vulnerabilities. The model also implies that the total number of vulnerabilities that could eventually be found is limited. (Rescorla, 2004) adopts the probabilistic G-O model (Goel and Okumoto, 1979), but finds no significant empirical evidence of its appropriateness. A model that relates the number of vulnerabilities to the total effort spent on detecting vulnerabilities is proposed by (Alhazmi and Malaiya, 1995).

Once a vulnerability is detected, the question arises whether to disclose it or not. (Rescorla, 2004) argues against disclosure unless vulnerabilities are correlated. However, investigating the operating system *OpenBSD* (Ozment, 2005) finds vulnerabilities being correlated regarding their rediscovery and argues in favour of disclosure. Using game-theoretic models, (Nizovtsev and Thursby, 2007; Arora, Krishnan, Nandkumar, Telang and Yang, 2004; Arora, Telang and Xu, 2004) address the question of when software vulnerabilities should be disclosed and conclude that neither instant disclosure nor non-disclosure are optimal.

In a theoretical paper, (Anderson, 2005) draws on software reliability models and statistical thermodynamics and concludes that, under ideal conditions, open and closed systems are equally secure.

METHODOLOGY

Software Packages and Data Sources

The selection of software packages investigated is driven by the goals to

- have open and closed source software systems that serve the same purpose (for the sake of comparability),
- include both open source software developed in cathedral style and in bazaar style
- have a sufficiently large set of vulnerability data available,
- consider software that is known and relevant to the community, and
- cover a broad range of services provided by the overall set of software packages.

Following these guidelines, I chose to include the software listed in Table 3 (see Annex) and described in the data section. Overall, the software sample contains nine closed source software bundles and eight open source software bundles.

Each of the selected software bundles is analyzed regarding its vulnerabilities, as published in the National Vulnerability Database (NVD) of the National Institute of Standards and Technology (NIST). This database is one of the most comprehensive vulnerability databases. I analyze each software product in terms of the number of vulnerabilities, the disclosure rate, the development of disclosure over time, and the severity of vulnerabilities. The statistical analysis focuses on the detection of differences between open source and closed source software.

Vulnerability Measurement

I define the "mean time between vulnerability disclosures" (MTBVD) as the *number of days since the 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.[1]

A simple comparison of the MTBVD of software packages is not assumed to provide reliable results regarding the level of security, because vulnerability detection and publication are probably correlated with market and with 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). While SLOC values can be used at cardinal level, market share values are regarded at ordinal level (low, medium, high) in this paper for two reasons: (1) in some cases no precise figures are available, (2) market share values change over time so that data on the continuous development of market shares would be needed for a reasonable consideration at cardinal level. Each of the application types is discussed separately with regard to its MTBVD, SLOC and market share.

[1] 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.

DATA

Investigated software

In the few empirical studies on software security (for example, see (Rescorla, 2004; Alhazmi et al. 2007)), the application types mainly considered are operating systems, web browsers, web servers, email clients, and database management systems. Adopting this focus, this study considers five operating systems (Windows 2000, Windows XP, MAC OSX, Red Hat Enterprise Linux 4, Debian 3.1), two web browsers (Internet Explorer 7, Firefox 2), two web servers (IIS 5, Apache 2), two email clients (MS Outlook Express 6, Thunderbird 2), four database management systems (mySQL 5, PostgreSQL 8, Oracle 10g, DB2 v3), and, in addition, two office products (MS Office 2003, OpenOffice 2). Details on these packages are provided in Table **3** (see Annex).

Vulnerability sources

I consider those vulnerabilities that have been accepted as Common Vulnerabilities and Exposures (CVE) by MITRE (http://cve.mitre.org)[2]. Each of these vulnerabilities has a unique identifier, e.g. CVE-1999-0067. CVE identifiers are also used as references in many other vulnerability databases; for a list of such databases see (MITRE, 2009). Among these databases, the NIST NVD (http://nvd.nist.gov/) is one of the most comprehensive ones, which provides (xml) data feeds for each year; vulnerabilities prior to and including 2002 are stored in a single xml file. In contrast to the data feeds provided by MITRE (http://cve.mitre.org/cve/cve.html), the NVD feeds contain data on the severity and type of vulnerabilities. I do not consider any misconfigurations (CCE = Common Configuration Enumeration), because the NVD database is still being set up in this regard.

Overall, I consider two types of vulnerabilities: those that are explicitly applicable to the software version under consideration, and those that affect all versions of the particular software and that have been published after the release date of the considered version. The data used in this paper refers to vulnerabilities that were published prior to 01 February 2009.

Content of the NIST national vulnerability database (NVD)

Each vulnerability entry listed in the NIST xml files includes the following data (and even more, which are not used here):

- **CVE identifier**, e.g. CVE-1999-0067
- **Affected software and versions**: The NVD applies the structured naming scheme CPE (Common Platform Enumeration) provided by MITRE (http://cpe.mitre.org/index.html). An example is “cpe:/o:redhat:enterprise_linux:3”.
- **(Base) Score**: The NVD provides vulnerability scores for almost all published vulnerabilities using the “Common Vulnerability Scoring System” (CVSS) 2.0 (FIRST, 2007; http://nvd.nist.gov/cvsseq2.htm.). The scores are between 0 and 10 (highest severity) and the particular value depends on several characteristics of the vulnerability, such as the level of authentication needed to exploit the vulnerability and the impact of a security breach on confidentiality and integrity.
- **Vulnerability references**: strings that provide references to sources with additional information on the vulnerability, such as links to available patches
- **Original release date**: This date refers to the particular NVD release day. In some cases, the corresponding CVE entry in the MITRE database contains another date, labeled as the “assigned date”. I could not find any specific explanation of this date, nor for the differences between corresponding dates. Neither of these dates necessarily mirrors the point of time when the vulnerability was detected. However, as this paper aims at comparing data on open source and closed source software and I assume that no relevant statistical difference between the (detection, publication) time gaps of open source and closed source software vulnerabilities exist, I use the publication date as included in the comprehensive NVD data feeds.

[2] A good overview of enumerations, standards, and languages for software security provides the MITRE site (http://makingsecuritymeasurable.mitre.org/).

EMPIRICAL RESULTS

Development of vulnerabilities over time

As Table 3 shows, for some of the closed source software packages I could not get reliable SLOC data. As data on market share are at ordinal level (see section on methodology), it is not possible to compute and (statistically) compare weighted MTBVD values. I therefore discuss each of the application types separately (see Table 1):

- **Browser**: Although *Internet Explorer 7* (IE 7) has had a much higher market share and its SLOC is presumably not lower than that of *Firefox 2*, the MTBVD of *IE7* is more than two times higher that of *Firefox 2*.
- **Email client**: Although I could not find any reliable data on the market shares of email clients, MS Outlook Express 6 has probably been much more widely deployed than *Thunderbird 1*. As in the case of browsers, no data on the SLOC of *MS Outlook Express* is available, but if we assume that *MS Outlook Express 6* does not have considerably fewer SLOC, then the MTBVD of the closed source software is about eight times higher than that of the open source software. As this result seems surprising, I doublechecked the analyzed data.
- **Web server**: The market shares of the considered web servers are in the medium range, with *Apache 2* having been more widely deployed than *IIS 5*. Again, I have no information on the SLOC of *IIS 5*. The MTBVD values of both software bundles are quite similar to each other.
- **Office**: In the case of office software, the open source software shows a MTBVD that is about three times higher than that of the closed source software. However, the market share of *MS Office 2003* is medium or even high, in contrast to that of *OpenOffice 2*. Overall, this result is not surprising.
- **Operating system**: The analysis of five operating systems reveals that the widely deployed *Windows* operating systems have shown a MTBVD that is about two times higher than those of the open source operating systems and *MAC OSX*. On the other hand, the SLOC of *MAC OSX* and *Debian 3.1* are higher than those of the Windows operating systems.
- **DBMS**: Regarding database management systems, none of the systems dominates the market. Overall, the results show a mixed picture.

To sum up the MTBVD results, in three of six application types, closed source software shows higher mean times, while in three cases no significant differences exist (if we also consider market shares and SLOC). However, this result might be biased and not representative, as in all but one case (databases) software of Microsoft is involved so that a company bias might be included. On the other hand, the software packages under consideration belong to the most deployed ones and cover a large part of worldwide installed software systems. The result does not mean that closed source software features less vulnerabilities or that less vulnerabilities have been detected, it only refers to vulnerabilities that have been published (see Figure 1).

While the discussion above provides a static picture of the history of vulnerabilities, I now address the development of vulnerabilities over time (see Figure 2-Figure 7 in the Annex for a graphical representation). For ten out of 17 considered software packages, a significant linear correlation between time and the number of vulnerabilities is found. For each package, the shape of its curve is given in Table 1, with R^2 (adj.) denoting adjusted R^2 when applying ordinary least squares (OLS). Four other packages either show a piecewise linear correlation – which, presumably, indicates the occurrence of specific events – or a linear correlation, for which, however, statistical evidence is weak due to the small number of data points. Three packages show a development that follows an S-shape in the beginning, as suggested by (Alhazmi, Malaiya and Ray, 2005), but finally changes its characteristics with the second derivation becoming positive again. Therefore, the results do not support their model regarding the qualitative development of vulnerability detection.[3] The results also show that (Alhazmi, Malaiya and Ray, 2005) underestimate the number of vulnerabilities that will eventually be found in Windows XP (88) and Windows 2000 (163), because the NVD lists 297 and 385 published ones, respectively (date: 31 January 2009).

Overall, there is no observable difference between open source and closed source software with regard to the (qualitative) development of vulnerabilities over time, and there is also no observable difference between open source software developed in bazaar and in cathedral style. The reason why three out of 17 packages show a different behaviour is not clear at this level of aggregation. An analysis of the particular types of vulnerabilities might reveal more facts.

[3] To be more precisely, (Alhazmi, Malaiya and Ray, 2005; Alhazmi, Malaiya and Ray, 2007) model the development of the number of detected vulnerabilities, while in this paper the number of published vulnerabilities is analyzed. On the other hand, (Alhazmi, Malaiya and Ray, 2007) use data on published vulnerabilities to show that their model fits.

Application type	Product	#vuln	MTBVD [days]	Development of vulnerability disclosure over time		
				Curve shape	R^2 (adj.)	Remark
Browser	Internet Explorer 7	74	13.29	Linear	0.99	
	Firefox 2	167	5.16	Linear	0.99	
Email client	MS Outlook Express 6	23	120.73	Linear	0.97	
	Thunderbird 1	110	13.79	S-shape, then strong increase		
Web server	IIS 5	83	40.90	Piecewise linear		
	Apache2	80	40.63	Linear	0.99	
Office	MS Office 2003	99	19.22	S-shape, then strong increase		
	OpenOffice 2	19	63.16	Linear	0.95	
Operating system	Windows 2000	385	9.35	Linear	0.99	
	Windows XP	297	8.97	Linear	0.98	
	MAC OSX	300	4.64	Linear	0.96	
	Red Hat Enterprise Linux 4[1)]	54 +284[2)] =338	4.32	Linear	0.95	
	Debian 3.1[1)]	22 +244[2)] =266	5.02	linear	0.96	
Database Management System	mySQL 5	33	46.00	linear		Too few data points available for any reliable statistic conclusion
	PostgreSQL 8	25	58.96	linear		
	Oracle 10g	63	29.72	S-shape, then strong increase		
	DB2 v8	13	136.38	linear		

[1)] The NVD lists linux kernel vulnerabilities separately from vulnerabilities of specific Linux distributions. Both *Red Hat Enterprise Linux 4* and *Debian 3.1* contain *Linux kernel 2.6*. As many consecutive versions of *Linux kernel 2.6* have been released, in each case I consider only those kernel 2.6 vulnerabilities that were published after the release date of *Red Hat Enterprise Linux 4* and *Debian 3.1*, respectively.

[2)] Linux kernel

Table 1. Published vulnerabilities in terms of MTBVD and development over time

Severity of vulnerabilities

I analyzed the severity of vulnerabilities for each software package in terms of mean, median, standard deviation, and proportion of highly severe vulnerabilities. For each application type, the median of medians is also given (see Table 2). 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, the number of investigated software bundles is still too low to deduce general hypotheses. An investigation of the type of vulnerabilities might reveal the reasons for the observed differences.
- 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 level of severity. However, applying statistical analysis (Mann-Whitney U-test), no statistically significant differences can be found: the two-tailed test provides a high number for P (P=0.1139). Applying the same test to the proportion figures, the test, again, does not indicate that the samples are significantly different (P=0.06). Summing up, I find no significant difference between the severity of vulnerabilities in open source and closed source software.
- Comparing open source software developed in bazaar style with that developed in cathedral style, no significant difference in terms of median (P=0.25) and also no significant difference in terms of the proportion of highly severe vulnerabilities occur (P=0.39).

Application type	**Product**	**Severity (range=[0;10])**				
		mean	**median**	**std. dev.**	**Proportion of highly severe vuln. ([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
	OpenOffice 2	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 OSX	6.18	6.80	2.13	41.33%	
	Red Hat Enterprise Linux 4[2)]	4.81	4.90	2.20	24.56%	
	Debian 3.1[2)]	4.79	4.90	2.15	22.93%	
Database Management Systems	mySQL 5	5.05	4.90	2.02	12.12%	5.7
	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%	

[1)] compliant with CVSS severity ratings

Table 2. Severity of published vulnerabilities

CONCLUSIONS

Reviewing literature on open source and closed source security reveals a lack of research in applying appropriate metrics, methodology and hard data. This paper contributes to solving this problem by analyzing and comparing published vulnerabilities of widely deployed open source software and closed source software packages.

The empirical investigation shows that the mean time between vulnerability disclosures was lower for open source software in three out of six cases, while the other cases show no differences. This means that only if vulnerability disclosure supports software security, open source software would (tend to) be more secure. It should also be noted that the present analysis does not cover detected, but unpublished vulnerabilities. This gap leads to the interesting research question of the relevance of this gap.

A surprising result of the empirical analysis is that for 14 out of 17 considered software packages, an (in most cases) significant linear or piecewise linear correlation between the number of published vulnerabilities and time occurs, while in only three cases the development follows an S-shape (at least at the beginning), as assumed in the literature. This does not only mean that the detection of vulnerabilities in the beginning of a software lifecycle is underestimated, it also shows that the detection of vulnerabilities does not level off over time. Consequently, addressing vulnerabilities must not be neglected in any phase of the software lifecycle. However, it is still an open question why some software packages show an S-shape. An analysis of particular types of vulnerabilities might reveal more facts.

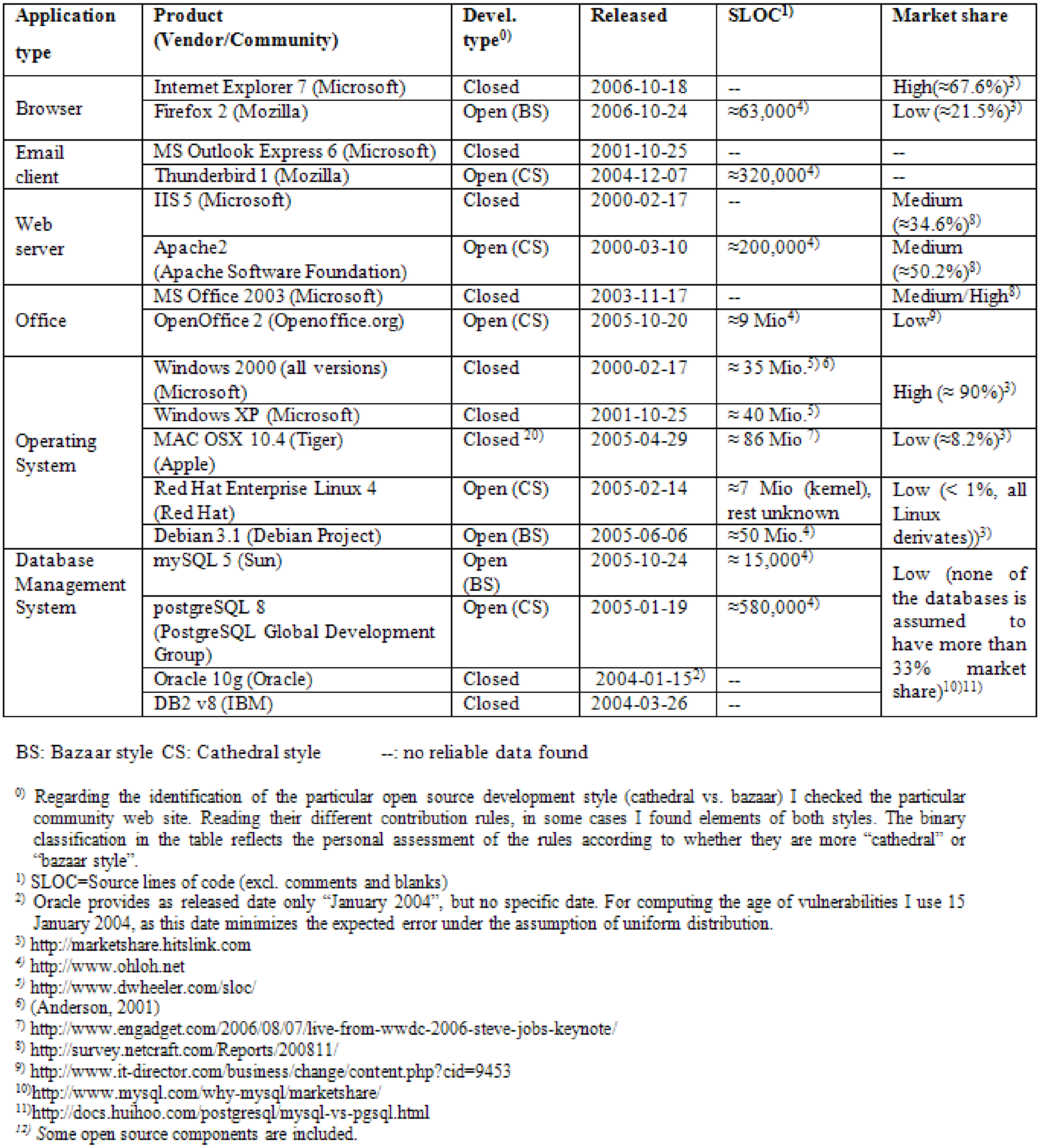
The empirical analysis shows differences in terms of vulnerability severity for different application types. Again, an investigation of the vulnerability type might reveal the reasons. However, no significant differences in terms of vulnerability severity were found between open source and closed source.

ACKNOWLEDGMENT

This work was made possible in part by the National Science Foundation under grant NSF-0433702. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors or originators and do not necessarily reflect the views of the National Science Foundation.

ANNEX

Software



Application type	Product (Vendor/Community)	Devel. type[0]	Released	SLOC[1]	Market share
Browser	Internet Explorer 7 (Microsoft)	Closed	2006-10-18	--	High(≈67.6%)[3]
	Firefox 2 (Mozilla)	Open (BS)	2006-10-24	≈63,000[4]	Low (≈21.5%)[3]
Email client	MS Outlook Express 6 (Microsoft)	Closed	2001-10-25	--	--
	Thunderbird 1 (Mozilla)	Open (CS)	2004-12-07	≈320,000[4]	--
Web server	IIS 5 (Microsoft)	Closed	2000-02-17	--	Medium (≈34.6%)[8]
	Apache2 (Apache Software Foundation)	Open (CS)	2000-03-10	≈200,000[4]	Medium (≈50.2%)[8]
Office	MS Office 2003 (Microsoft)	Closed	2003-11-17	--	Medium/High[8]
	OpenOffice 2 (Openoffice.org)	Open (CS)	2005-10-20	≈9 Mio[4]	Low[9]
Operating System	Windows 2000 (all versions) (Microsoft)	Closed	2000-02-17	≈ 35 Mio.[5] [6]	High (≈ 90%)[3]
	Windows XP (Microsoft)	Closed	2001-10-25	≈ 40 Mio.[5]	
	MAC OSX 10.4 (Tiger) (Apple)	Closed [20]	2005-04-29	≈ 86 Mio [7]	Low (≈8.2%)[3]
	Red Hat Enterprise Linux 4 (Red Hat)	Open (CS)	2005-02-14	≈7 Mio (kernel), rest unknown	Low (< 1%, all Linux derivates))[3]
	Debian 3.1 (Debian Project)	Open (BS)	2005-06-06	≈50 Mio.[4]	
Database Management System	mySQL 5 (Sun)	Open (BS)	2005-10-24	≈ 15,000[4]	Low (none of the databases is assumed to have more than 33% market share)[10][11]
	postgreSQL 8 (PostgreSQL Global Development Group)	Open (CS)	2005-01-19	≈580,000[4]	
	Oracle 10g (Oracle)	Closed	2004-01-15[2]	--	
	DB2 v8 (IBM)	Closed	2004-03-26	--	

BS: Bazaar style CS: Cathedral style --: no reliable data found

[0] Regarding the identification of the particular open source development style (cathedral vs. bazaar) I checked the particular community web site. Reading their different contribution rules, in some cases I found elements of both styles. The binary classification in the table reflects the personal assessment of the rules according to whether they are more "cathedral" or "bazaar style".

[1] SLOC=Source lines of code (excl. comments and blanks)

[2] Oracle provides as released date only "January 2004", but no specific date. For computing the age of vulnerabilities I use 15 January 2004, as this date minimizes the expected error under the assumption of uniform distribution.

[3] http://marketshare.hitslink.com

[4] http://www.ohloh.net

[5] http://www.dwheeler.com/sloc/

[6] (Anderson, 2001)

[7] http://www.engadget.com/2006/08/07/live-from-wwdc-2006-steve-jobs-keynote/

[8] http://survey.netcraft.com/Reports/200811/

[9] http://www.it-director.com/business/change/content.php?cid=9453

[10]http://www.mysql.com/why-mysql/marketshare/

[11]http://docs.huihoo.com/postgresql/mysql-vs-pgsql.html

[12] Some open source components are included.

Table 3. Investigated open and closed source software

Vulnerability disclosure

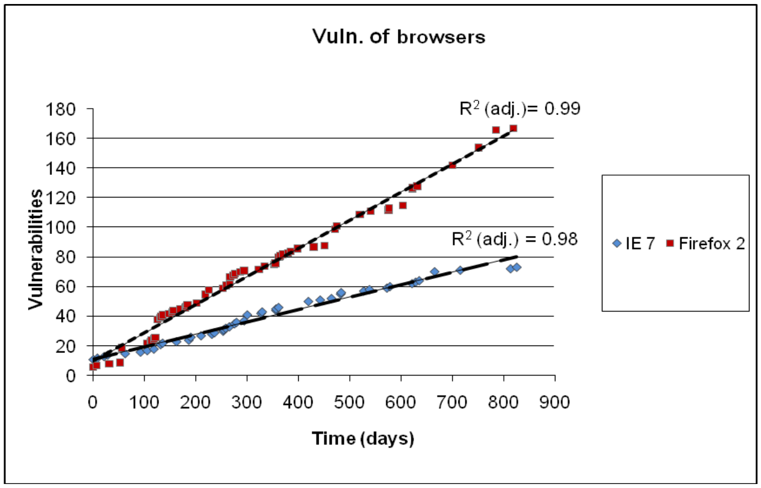


Figure 2. Vulnerability disclosure of browsers over time

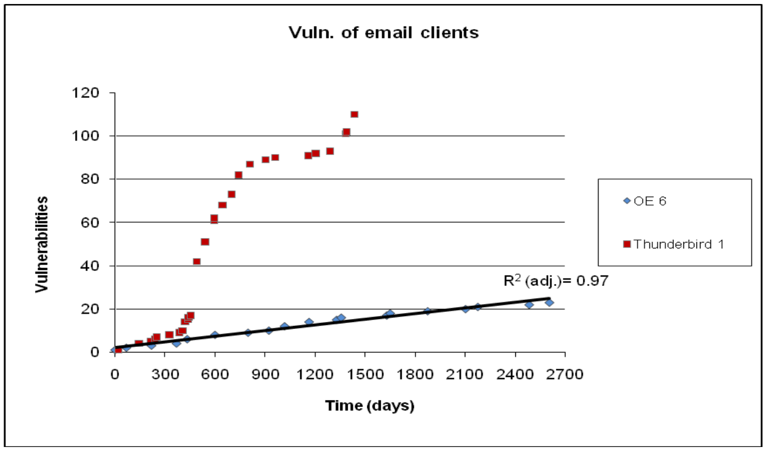


Figure 3. Vulnerability disclosure of email clients over time

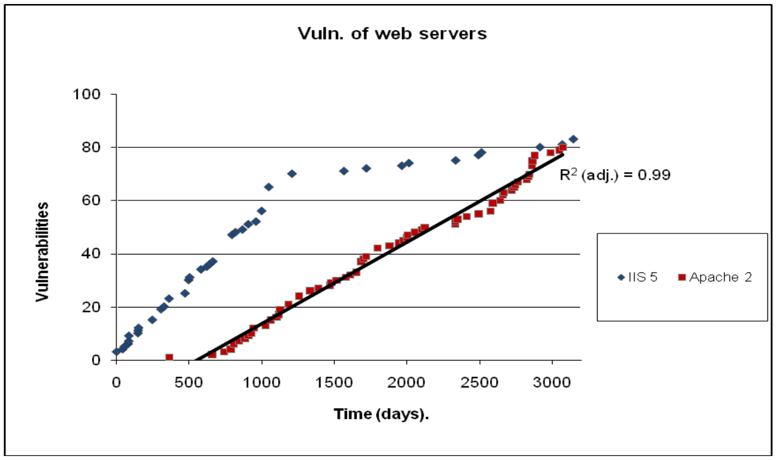


Figure 4. Vulnerability disclosure of web servers over time

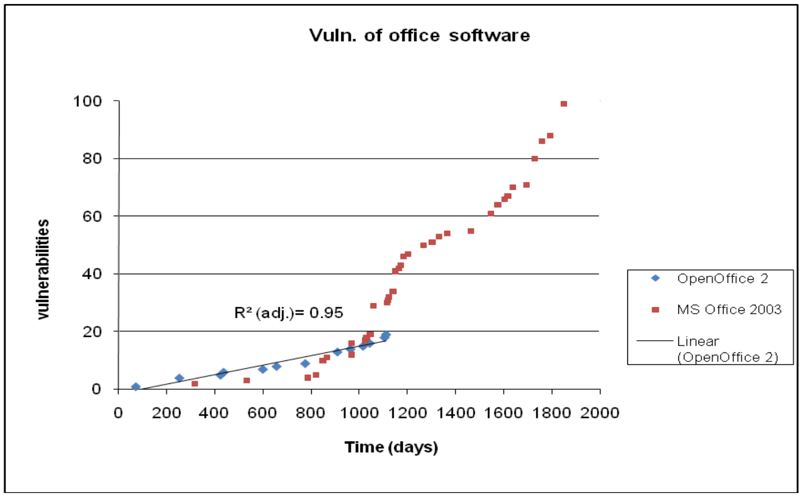


Figure 5. Vulnerability disclosure of office software over time

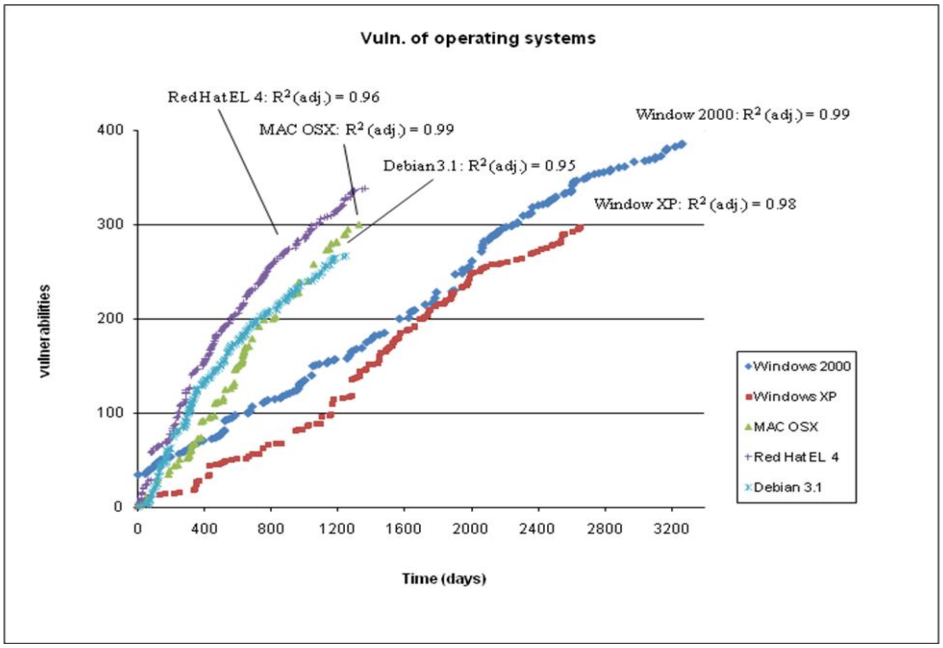


Figure 6. Vulnerability disclosure of operating systems over time

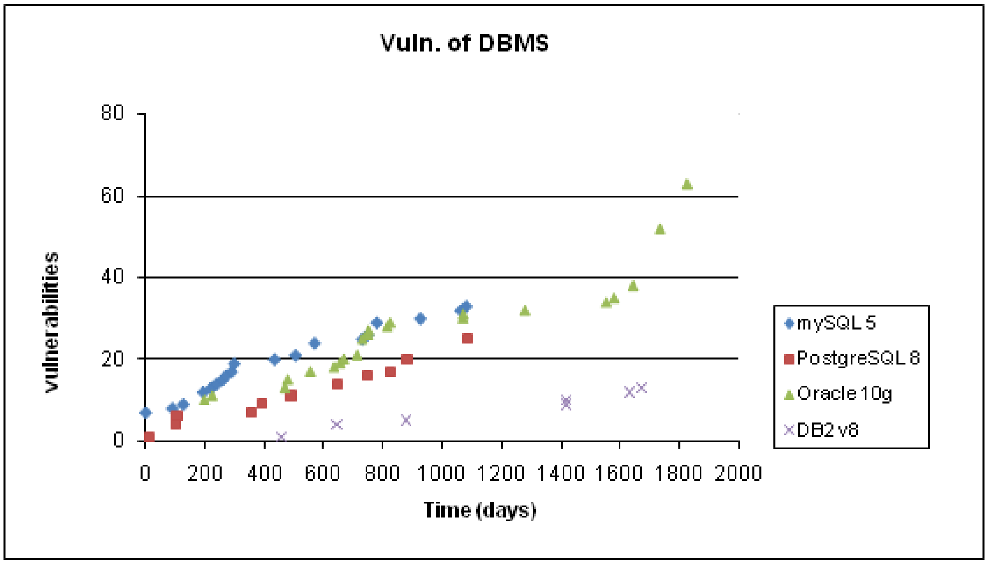


Figure 7. Vulnerability disclosure of DBMS over time

REFERENCES

1. Alhazmi, O., Malaiya, Y. and Ray, I. (2005) Security Vulnerabilities, in *Software Systems: A Quantitative Perspective in Data and Applications Security 2005*, LNCS 3654, 281-294.
2. Alhazmi, O., Malaiya, Y., Ray, I. (2007) Measuring, analyzing and predicting security vulnerabilities in software systems, in *Computers & Security*, 26, 3, 219-228.
3. Anderson, R. (2005) Open and Closed Systems are Equivalent (that is, in an ideal world), in Feller, J., Fitzgerald, B., Hissam, S. A. and Lakhani, K.R. (Eds.) *Perspectives on Free and Open Source Software*, MIT Press, Cambridge, 127–142.
4. Anderson, R. (2002) Security in Open versus Closed Systems – The Dance of Boltzmann, Coase and Moore, in *Proceedings of the Conference on Open Source Software Economics*, Toulouse, France, June 20-21, 1-13.
5. Anderson, R. (2001) Why Information Security is Hard – An Economic Perspective, in *Proceedings of the Seventeenth Computer Security Applications Conference*, New Orleans, December 10-14, 358-365.
6. Arora, A., Krishnan, R., Nandkumar, A., Telang, R. and Yang, Y. (2004) Impact of Vulnerability Disclosure and Patch Availability – An Empirical Analysis, in *Proceedings of the Third Workshop on the Economics of Information Security*, University of Minnesota, May 13-14, 1-20.
7. Arora, A., Telang, A. and Xu, H. (2004), "Optimal Policy for Software Vulnerability Disclosure", in *Proceedings of the Third Annual Workshop on Economics and Information Security*, University of Minnesota, May 13-14, 52-59.
8. FIRST (2007) A Complete Guide to the Common Vulnerability Scoring SystemVersion 2.0, http://www.first.org/cvss/cvss-guide.html.
9. Free Software Foundation (FSF) (2007) The Free Software Definition, http://www.fsf.org/licensing/essays/free-sw.html.
10. Glass, R.L. (2004) A look at the economics of open source, in *Comm. of the ACM*, 47,2, 25-27.
11. Goel, A.L. and Okumoto, K. (1979) Time-Dependent Error-Detection Rate Model for Software and Other Performance Measures, in *IEEE Transactions on Reliability*, 28, 3, 206-211.
12. Gonzalez-Barahona, J. M. (2000) Free Software/Open Source: Information Society Opportunities for Europe?, Working group on Libre Software, http://eu.conecta.it/paper/cathedral_bazaar.html.
13. Jonsson, E., Strömberg, L. and Lindskog, S. (2000) On the functional relation between security and dependability impairments, in *Proceedings of the 1999 Workshop on New Security Paradigms*, Caledon Hills, Ontario, Canada, September 22 – 24, 104-111.
14. Kimura, M. (2006) Software vulnerability: definition, modelling, and practical evaluation for e-mail transfer software, in *International Journal of Pressure Vessels and Piping*, 83, 4, 256-261.

15. Levy, E. (2000) Wide open source, http://www.securityfocus.com/news/19.
16. Messmer, E. (2005) Open source vs. Windows: security debate rages, in *Network World*, 22, 26, 26-27.
17. MITRE (2009) Vulnerability Management Products & Services by Product Type, http://cve.mitre.org/compatible/vulnerability_management.html.
18. Naraine, R. (2006) DHS backs open-source security, in *eWeek*, 23, 3, 20.
19. Nizovtsev, D. and Thursby, M. (2007) To disclose or not? An analysis of software user behavior, in *Information Economics and Policy*, 19, 1, 43-64.
20. Open Source Initiative (OSI) (2006) The Open Source Definition, http://www.opensource.org/docs/osd.
21. Ozment, A. (2005) The Likelihood of Vulnerability Rediscovery and the Social Utility of Vulnerability Hunting, in *Proceedings of the Fourth Workshop on the Economics of Information Security*, Harvard University, June 2-3, Cambridge, Massachusetts, 1-21.
22. Payne, C. (2002) On the security of open source software, in *Information Systems Journal*, 12, 1, 61-78.
23. Raymond, E.S. (2001) The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary, O'Reilly, Beijing, China.
24. Rescorla, E. (2004) Is finding security holes a good idea?, in *Proceedings of the Third Annual Workshop on Economics and Information Security*, University of Minnesota, May 13-14.
25. Schryen, G. and Kadura, R. (2009) Open Source vs. Closed Source Software: Towards Measuring Security, in *Proceedings of the 2009 ACM Symposium on Applied Computing*, Honolulu, Hawaii, USA, March 8-12, 2016-2023.
26. Schwarz, M. and Takhteyev, Y. (2008) Half a Century of Public Software Institutions: Open Source as a Solution to Hold Up Problem, http://www.takhteyev.org/papers/Schwarz-Takhteyev-2008.pdf.
27. Witten, B., Landwehr, C. and Caloyannidis, M. (2001) Does open source improve system security?, in *IEEE Software*, 18,5, 57-61.