A comprehensive and comparative analysis of the patching behavior of open source and closed source software vendors

Guido Schryen
RWTH Aachen University, Institute of Business Information Systems, Germany
schryen@gmx.net

Abstract

While many theoretical arguments against or in favor of open source and closed source software development have been presented, the empirical basis for the assessment of arguments is still weak. Addressing this research gap, this paper presents a comprehensive empirical investigation of the patching behavior of software vendors/communities of widely deployed open source and closed source software packages, including operating systems, database systems, web browsers, email clients, and office systems. As the value of any empirical study relies on the quality of data available, this paper also discusses in detail data issues, explains to what extent the empirical analysis can be based on vulnerability data contained in the NIST National Vulnerability Database, and shows how data on vulnerability patches was collected by the author to support this study. The results of the analysis suggest that it is not the particular software development style that determines patching behavior, but rather the policy of the particular software vendor.

1. Introduction

During the past 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. However, in more recent years, this widely cultivated habit has begun to be accompanied by a software model, which is characterized by software that comes with a compilable source code. This source code is often free of charge and may be even modified or redistributed. The software type is referred to by the term "open source software" (OSS).

The application fields of OSS are manifold. Internet programs, such as the mail transfer agent Sendmail, the Web server Apache, the operating system Linux, the database system MySQL, and the office package OpenOffice are some of the most popular examples. Beyond these application types, we also find computer games (http://osswin.sourceforge.net/games.html) and even business applications, such as AvErp, which is a German stock inventory system for small- and medium-sized businesses (http://www.synerpy.de/), or an Enterprise Resource Planning (ERP) system that is being built by a group of U.S. universities and that is being overseen by the Kuali Foundation (http://kuali.org/). OSS has even become part of the core infrastructure of sophisticated technology companies, such as Amazon, Google, and Yahoo [1]. Obviously, OSS has arrived in the world of important and critical information systems 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.

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 OSS stress the strength of the resulting review process [2] and argue in the sense of Raymond [3] that, "Given enough eyeballs, all bugs are shallow." (p. 19), while

some opponents follow the argument of Levy [4], who remarks "Sure, the source code is available. But is anyone reading it?" Viega [5] further doubts the superior effectiveness of the open source community and argues that (1) most code reviewers do not explicitly look for vulnerabilities and (2) those who do, are mostly interested in finding those vulnerabilities that are easy to detect and that bring them high reputation.

While the security discussion is pervaded with "beliefs and guesses", only few quantitative models and some empirical studies [6-15] appear in the literature. Most of these empirical studies investigate one package or few software packages only, and to the best knowledge of the author no prior study has been conducted to comprehensively study differences between the patching behavior of open source and closed source software vendors. The reason why comprehensive empirical studies have been neglected in general is probably much due to the fact that the collection and analysis of reliable data is still laborious and much manual work is required. However, empirical research is necessary, as it has the potential to provide insights in the security of widely deployed information systems, to support researchers in developing models for security measurement, and to enrich the security discussion with the provision of facts.

Interestingly, past empirical studies focus on the number of vulnerabilities and neglect to consider their severities and its' impact on vendors' patching behavior. However, this perspective is important as a single highly severe vulnerability that enables attackers to get root access to a system is usually more crucial than 10 low severe vulnerabilities that only grant reading access to unauthorized users. Addressing this lack in research, this study collects comprehensive empirical data and analyzes open and closed source software with regard to vendors' behavior in patching vulnerabilities. Thereby, it extends an earlier study [14] in two ways: it builds up a new data pool of patching data, which is not available in publicly accessible databases, and it uses these data to investigate vendors' behavior in terms of which vulnerabilities have been patched.

The remainder of this paper is organized as follows. The following section presents the background of open source and closed source software. Section 3 proposes a vulnerability lifecycle model and applies this model to (a) synthesize literature findings on software vulnerabilities and patches, and (b) to discuss patching activities through a theoretical lens. Section 4 explains the research methodology of this study, including the selection of investigated software packages and used data, before Section 5 presents the findings of this empirical study. Finally, the results are summarized and conclusions are presented.

2. 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 [16]. The (open source) definition (OSD) 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 [3] is in place, any volunteer can provide source code submissions. Software development is then often based on informal communication between the coders [17]. 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" [3]. As the particular development style might have an impact on the security of software, a detailed discussion of open source security should take this into account.

Several 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." [18] In contrast to the OSD definition, the FSF definition explicitly focuses on the option of releasing the improvements to the public, thereby rejecting a strong supervision of the modification process. More specifically, the definition says: "If you do publish your changes, you should not be required to notify anyone in particular, or in any particular way." Similar to the discussion of what open or free software is, we need to define what "closed software" is: Software is usually regarded as being "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)" reflects the perspective of developers and specifies the type of development. Complementarily, one could also adopt the software user's point of view by distinguishing between software that needs to be paid for and software for which no fee applies. The resulting (two-dimensional) classification scheme is shown in Table 1.

Table 1. Classification of software [19; p. 2018]

	Open Source (license)	**Closed Source (source code not available)**
Free of charge	Linux, Apache web server	Adobe Acrobat Reader
Subject to charge	MySQL (dual licensing: GPL/proprietary license for Enterprise Edition)	Microsoft Windows operating systems

3. The vulnerability lifecycle

When software is executed in a way different from what the original software designers intended, this misbehavior is rooted in software bugs. Anderson [20] assumes the ratio between software bugs and software lines of code (SLOC) to be about 1:35, i.e. Windows 2000 with its 35 Mio. SLOC would have one million bugs included. When bugs can be directly used by attackers to gain access to a system or network, they are termed (information security) "vulnerabilities" by MITRE [21]. Although there are other definitions of "vulnerabilities" [22;23], the adoption of the MITRE definition is useful (in a pragmatic, but not necessarily normative sense) for four reasons:

(1) Most empirical studies implicitly use this definition by analyzing "Common Vulnerability and Exposures (CVE)" entries, which are based on the understanding of MITRE. CVE names 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".

(2) The process of accepting a potential software bug as CVE vulnerability is well documented and the assessment is conducted by security experts [21].

(3) The U.S. National Institute of Standards and Technology (NIST) adopts the MITRE understanding of vulnerabilities in their 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.

(4) The definition is precise (http://cve.mitre.org/about/terminology.html):

A vulnerability is a state in a computing system (or set of systems) that either:
- *allows an attacker to execute commands as another user*
- *allows an attacker to access data that is contrary to the specified access restrictions for that data*
- *allows an attacker to pose as another entity*
- *allows an attacker to conduct a denial of service*

It should be noticed that this definition does not exactly match the US-CERT vulnerability definition, but is closely related: "While the mapping between CVE names and US-CERT vulnerability IDs are usually pretty close, in some cases multiple vulnerabilities may map to one CVE name, or vice versa. The CVE group tracks a large number of security problems, not all of which meet our criteria for being considered a vulnerability."[23]

Vulnerabilities and their dynamic behavior can be described with 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 Frei [8]), integrates states that have been introduced by Arbaugh et al. [24], and uses a cycle to account for the fact that (the patching of) vulnerabilities can create new vulnerabilities [24]. The lifecycle starts with the injection of a vulnerability into software. In principle, a vulnerability can find its way into software through (a) the intentional behavior of software developers, who strive for selling or exploiting vulnerabilities, or for harming the employer, or (b) unintentional behavior, which can be rooted in careless programming or in using "insecure" development tools. This behavior can be economically rational as companies often do not have sufficient incentives to avoid vulnerabilities [25]. After some testing, the software is finally released and the search for vulnerabilities begins for the public (and potentially continues for the software vendor).

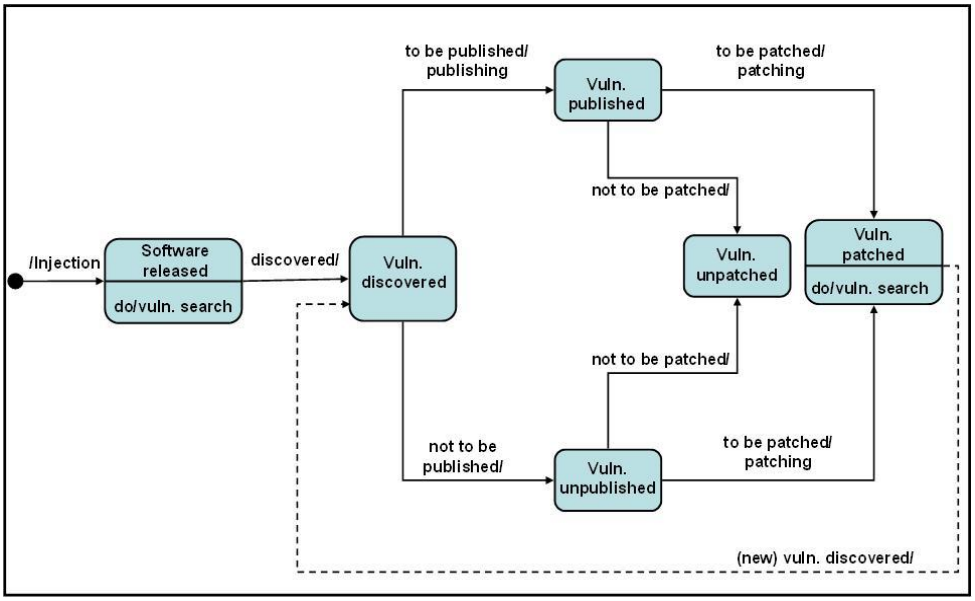

Figure 1. Vulnerability life cycle

The discovery of a vulnerability can be based on coincidental detection or on the active search of persons with intrinsic motivation (to make software more secure) or with extrinsic motivation (to get reputation, to gain financial advantage, or "to do their job"). When a vulnerability is discovered, the question occurs whether it should be published or not. If the vulnerability is detected by a "black hat", 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, who can exploit the vulnerability exclusively. 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: Rescorla [13] argues against disclosure as he finds the probability of vulnerability rediscovery being vanishingly small. However, investigating the operating system OpenBSD, Ozment [15] finds vulnerabilities being correlated regarding their rediscovery and argues in favour of disclosure. Using game-theoretic models, Arora et al. [26] and Nizovtsev and Thursby [27] address the question of when software vulnerabilities should be disclosed and conclude that neither instant disclosure nor non-disclosure is optimal. Arora et al. [28] use empirical analysis to support their hypothesis, pointing out that the optimal policy depends upon how quickly vendors provide patches and upon how likely attackers are to find and exploit vulnerabilities. Choi et al. [29] discuss different disclosure regimes and conclude that mandatory disclosure improves welfare only when the probability of attack is high and the expected damage is small. An overview of the classification of vulnerabilities is provided in Figure 2, which also shows that in this paper only published vulnerabilities are considered, as no reliable data is available for unpublished vulnerabilities.

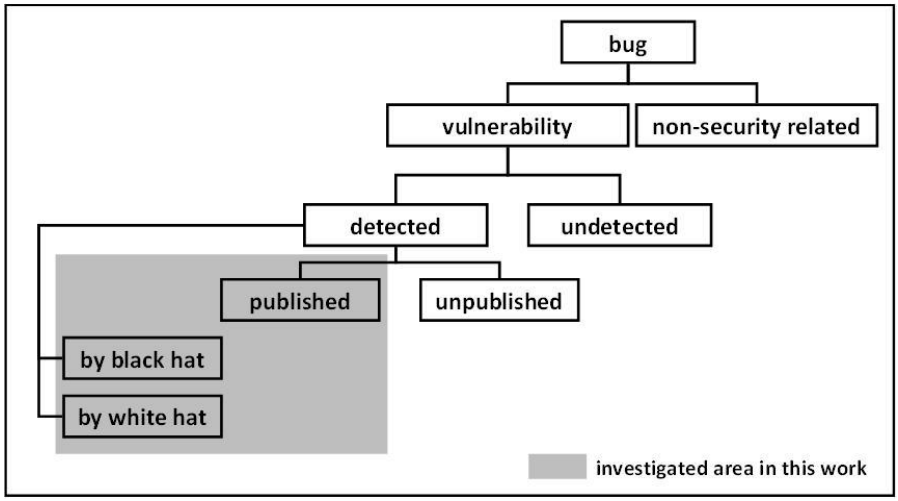

Figure 2. Classification of software bugs and vulnerabilities, source: [14; p. 2]

Once a vulnerability is published, at a 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 to not provide a patch, when it is the customers who suffer most cost of failure and when competitors behave alike. Arora et al. [30] analyze the timing of patch release and find that both the competition effect and disclosure threat effect hasten patch release, with competition having an even stronger effect. Cavusoglu et al. [31] apply game theory to compare liability

and cost-sharing as mechanisms for incentivizing vendors to patch their software and conclude that liability helps where vendors release less often than optimal, while cost-sharing helps where they release more often.

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 arguments still hold, the decision to not provide a patch might be additionally rooted in the assumptions that (a) a non-published vulnerability is hardly exposed to attacks, (b) any vulnerability disclosure reduces the vendor's reputation, and (c) 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 [32]. As the injection refers to new vulnerability, Figure 1 shows a dashed line.

The uncertainty of whether a vulnerability should be published and patched also applies to the decision of whether a software patch should be installed. The customers – be they private users or institutions – still have to determine the risk of installing the patch (immediately) for two reasons: First, the patch might contain even more critical vulnerabilities than the patched ones. Second, the benefit from having one or several vulnerabilities removed needs to be opposed to the risk that the patch installation makes applications dysfunctional, which can lead to considerably economic harm when production systems discontinue working or online shops are shut down, for example.

The previous discussion of the lifecycle 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, Anderson [33] draws on software reliability models and statistical thermodynamics to show that although, under ideal conditions, open and closed systems are equally secure, this symmetry can be broken due to economic phenomena, such as transaction costs and the behavior of vendors.

4. Research methodology

The research framework used in this paper is shown in Figure 3. In order to answer the research questions whether (particular styles of) open source development or closed source development lead to more effective patching behavior of vendors, it is most essential to select appropriate software packages and to have comprehensive and reliable data on vulnerabilities and patches available. Subsection 4.1. explains the software selection process in detail. As prior works [8;15] argue that data quality is still a serious issue for the empirical analysis of vulnerabilities and consequently also of patches, Subsection 4.2 reveals how this empirical study addresses the challenge to have reliable data available.

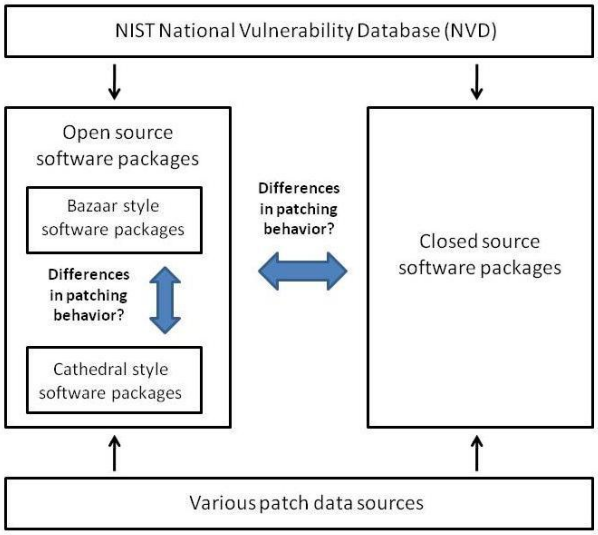

Figure 3. Research framework

4.1. Investigated software packages

In order to draw a picture of empirical open source and closed source software security is seems alluring to consider as many software packages and vulnerability data as possible. But this approach suffers from at least two problems. First, for many software packages only (too) few vulnerability data are 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 the qualitative approach and to manually select software packages for empirical analysis. The selection of software was driven by the goal to have different groups of widely deployed (open source and closed source) software available, which contemporaneously show diversity in functionality across groups (comprehensiveness) and homogeneity in functionality inside the groups (comparability). Consequently, it cannot be proved that the results of this empirical study also apply to other, less deployed packages, but the results provide a comprehensive overview of software that is widely used in private and institutional environments and that is thus in the focus of attackers and defenders.

Assuming that most software is usually attacked through the (client-server-based) Internet, I adopt the client-server perspective to frame the selection of software packages (see Figure 4). At the client side, the most widely deployed operating systems (OS) are Microsoft OS, *MAC OSX* and Linux derivations [34]. Among the Microsoft OP, *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 (arbitrarily) 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, which are widely used in both private and commercial environments. Regarding web browsers, *Internet Explorer* and *Firefox* are the most widely used programs [34], 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* and *OpenOffice*.

On the server site, 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 [35]. *Oracle* and *DB2* are two of the mostly used closed source DBMS [36], while for open source DBMS no reliable data could be found. Having explored many database-related websites, I decided to use DB2 and PostgreSQL, which are widely deployed. The specific versions of the software packages are given in Table 2.

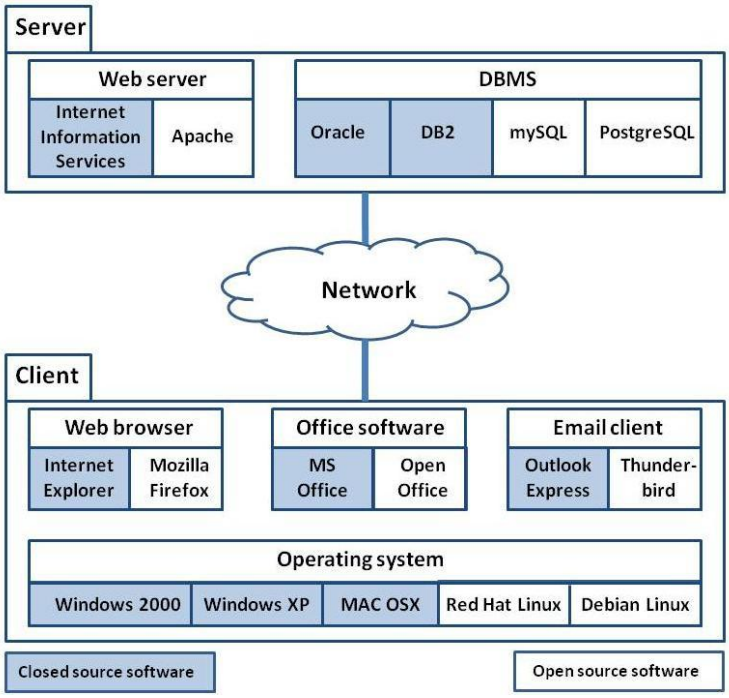

Figure 4. Selected open source and closed source software packages

4.2. Vulnerability data

The MITRE CVE group does not only provide a definition of vulnerabilities (see discussion above), but also provides a dictionary of vulnerabilities [21]. This dictionary contains for each vulnerability a standard identifier number (e.g. CVE-1999-0067), a brief description, and references to related vulnerability reports and advisories. As the data sources of CVE are manifold and include trustful 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 content team assures that each CVE candidate has been inspected by security professionals. For a detailed description of MITRE CVE see Appendix A. Overall, the CVE dictionary is a valuable resource for vulnerability analysis in terms of both quantity and quality. The CVE group recommends to use the NIST National Vulnerability Database (NVD) (http://nvd.nist.gov/), which is the only data pool that provides full database functionality for the complete MITRE CVE dictionary.

The NVD, formerly known as ICAT, contains information on all CVE identifiers. The NVD is updated immediately whenever a new vulnerability is added to the CVE dictionary of vulnerabilities. New vulnerabilities are then analyzed by NVD analysts on a first-in, first-out

basis and augmented with attributes (see below) usually within two U.S. government business days [37]. The NVD team then adds additional information, some of which is as follows [38]:

- Affected software and versions: The NVD applies the structured naming scheme CPE (Common Platform Enumeration) provided by MITRE. An example is "cpe:/o:redhat:enterprise_linux:3". The NVD team does not actively test products to determine affected products, but the NVD team consults MITRE, public vulnerability sites, vendors and security researches when analyzing CVEs. NVD also supports a vendor comment process that allows vendors to publish comments to the NVD regarding affected products. These cooperation activities also include the determination of severity base scores (see below).
- (Base) Score: The NVD provides vulnerability scores for almost all published vulnerabilities using the "Common Vulnerability Scoring System" (CVSS) 2.0 (http://www.first.org/ cvss/cvss-guide.html). 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. CVSS scores for vulnerabilities published prior to 11/9/2005 were approximated by the NVD team from prior CVSS metric data (CVEs were originally scored using CVSS Version 1, these scores were converted to CVSS Version 2 scores based on an approximation algorithm).
- Original release date (ORD): The ORD assigned to a CVE identifier does not necessarily mirror the actual date of disclosure due to two potential time gaps: 1) Time between the actual disclosure of a vulnerability (on the web or in mailing lists, for example) and its consideration in the "Assigned" phase of the MITRE CVE workflow. (2) Time between the "Assigned" date and the NVD publication date. This gap is usually not larger than some days [37], but as information on time gap (1) is available, the computation of patch times and exploit times would contain errors of unknown size.

For a detailed description of the NVD see Appendix B, which contains information that I gained through personal communication with the current NVD program manager. The following analysis of NVD vulnerabilities is based on NVD xml data feeds as available at 31 January 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 – the NVD team does not explicitly check versions [35] –, I doublechecked the affected software versions of each vulnerability on the websites of vendors, MITRE, and SecurityFocus. In very few cases of inconsistencies I excluded the particular vulnerability from any further analysis. This procedure was extremely time-consuming, but useful to assure the correctness of NVD information on affected software versions.

4.3. Patch data

While the analysis of vulnerabilities and their publication refers to the first three phases of the software vulnerability lifecycle and thereby mirrors software communities' behavior in terms of creating, detecting, and publishing vulnerabilities, the investigation of the provision of patches aims at identifying communities' behavior regarding actively addressing and finally removing vulnerability issues. 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 vendor's provision of patches, as the vulnerability publication dates contained in the NVD do not necessarily give the actual publication date (cmp. discussion above). In contrast to vulnerability publication data, reliable data on patches can be (manually) collected by directly looking up vendors' sites and vendor-neutral websites. 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, VUPEN Security, mySQL Forge, and Oracle Security Alerts and Patch Updates. For each CVE vulnerability, I searched these data sources in order to find a corresponding patch. Those vulnerabilities for which I could not find any patch information this way are regarded as unpatched. The newly compiled data pool contains patch data on the aforementioned browsers, email clients, web servers, office products, operating systems and database management systems.

5. Empirical results

Table 2 shows aggregated patch data for each software package. Vulnerabilities for which I could not find any patch information by February 28, 2009 are classified as "still unpatched". Figure 5 illustrates the proportions of unpatched vulnerabilities.

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, applying statistical analysis (Mann-Whitney U-test) on the proportions of unpatched vulnerabilities, no statistically significant differences between open and closed source software can be found: the two-tailed test provides a high number for p (p=0.48). Furthermore, all proportions greater than 30% are related to Microsoft products. Regarding open source software developed in bazaar or in cathedral style (see Appendix A), again, no statistically significant difference appears (p=0.79). Apparently, the proportion of still unpatched vulnerabilities largely depends on the specific vendor. I discuss this behavior in detail below.

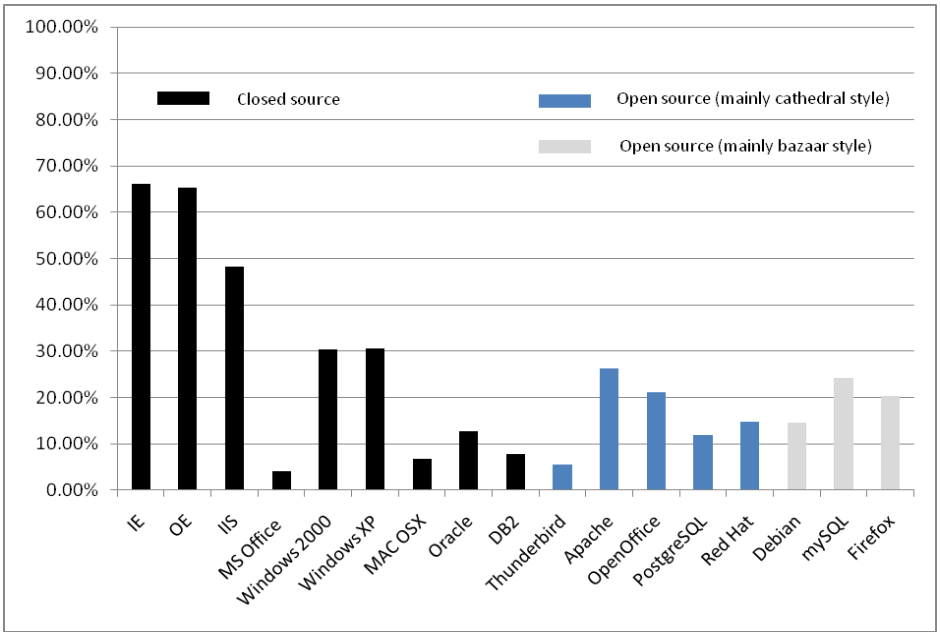

Figure 5. Proportions of unpatched vulnerabilities

Interestingly, the case of Microsoft also shows that even the same vendor can apply different patching behavior dependent 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, every second vulnerability of IIS is 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.

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 differences between open source and closed source software. The data in Table 2 reveal that, for all six Microsoft products, there is a strong bias towards patching the most 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 severe vulnerabilities, which reveals that software security is regarded a serious market issue. Apple (MAC OSX) 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 unpatched vulnerabilities are higher than those of the patched ones. To sum up, three out of four closed source software vendors leave only few vulnerabilities unpatched and the other vendor focuses on patching severe vulnerabilities.

Table 2. 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
	OpenOffice 2	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 OSX	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
	DB2 v8	13	1	7.69%	7.8	7.2	7.2

Regarding the medians of patched and unpatched vulnerabilities of open source vendors and their particular development style (bazaar vs. cathedral), 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. Consequently, I hypothesize that open source software development at least prevents "extremely bad" patching behavior.

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

6. Conclusion

This work presented the first comprehensive empirical study on the security of open source and closed source security. It compared 17 well known and widely deployed browsers, email clients, web servers, office systems, operating systems, and database systems regarding the patching behavior of the particular vendor. In order to assure high data quality, vulnerabilities were taken from the NIST NVD and manually doublechecked. Patch data for each vulnerability were collected by using vendor sites and vendor neutral platforms.

The empirical results showed that open source and closed source software do not significantly differ in terms of 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 patching activities. Rather, the policy of the particular development community or vendor determines the 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 disclosed vulnerabilities) or, even better, to avoid vulnerabilities at the outset.

7. Appendix

A. Investigated packages

Application type	Product	Vendor/Community	Devel. Type[1)]
Browser	Internet Explorer 7	Microsoft	Closed
	Firefox 2	Mozilla	Open (BS)
Email client	MS Outlook Express 6	Microsoft	Closed
	Thunderbird	Mozilla	Open (CS)
Web server	IIS 5	Microsoft	Closed
	Apache 2	Apache Software Foundation	Open (CS)
Office	MS Office 2003	Microsoft	Closed
	OpenOffice 2	Openoffice.org	Open (CS)
Operating System	Windows 2000 (all versions)	Microsoft	Closed
	Windows XP	Microsoft	Closed
	MAC OSX 10.4 (Tiger)	Apple	Closed [3)]
	Red Hat Enterprise Linux 4[2)]	Red Hat	Open (CS)
	Debian 3.1[2)]	Debian Project	Open (BS)
Database Management System	mySQL 5	Sun	Open (BS)
	postgreSQL 8	PostgreSQL Global Development Group	Open (CS)
	Oracle 10g	Oracle	Closed
	DB2 v8	IBM	Closed

BS: Bazaar style CS: Cathedral style

1) Regarding the identification of the particular open source development style (cathedral vs. bazaar) I checked the particular community websites. In some cases I found elements of both styles. The binary classification in the table reflects the author's assessment according to whether they are more "cathedral style" or "bazaar style".

2) The NVD lists linux kernel vulnerabilities separately from vulnerabilities of specific Linux distributions. Red Hat Enterprise Linux 4 uses Linux kernel 2.6.9, Debian 3.1 uses Linux kernels 2.4.27 or 2.6.8. I consider only those kernel vulnerabilities that were published after the release date of Red Hat Enterprise Linux 4 and Debian 3.1, respectively.

3) Some open source components are included.

B. Data sources: MITRE CVE and NIST NVD

MITRE CVE (Common Vulnerabilities and Exposures) [21]:

- CVE is a dictionary of publicly known information security vulnerabilities and exposures and is managed by the U.S. MITRE Corporation, which is a not-for-profit organization.
- The CVE dictionary contains for each vulnerability the standard identifier number (e.g. CVE-1999-0067) with status indicator, a brief description, and references to related vulnerability reports and advisories.

How the CVE dictionary is built**:**

1) Initial submission stage: MITRE has a content team whose primary task is to analyze, research, and process incoming vulnerability submissions from CVE's data sources (SecurityFocus.com weekly Newsletters, Network Computing and the SANS Institute - weekly Security Alert Consensus, ISS - monthly Security Alert Summary, NIPC

CyberNotes - biweekly issues, National Cyber Alert System), transforming the submissions into candidates.

2) Candidate stage: Candidates are normally created in one of three ways: (i) they are refined by the content team using submissions from CVE's data sources; (ii) they are reserved by an organization or individual who uses it when first announcing a new issue; or (iii) they are created "out-of-band" by the CVE Editor, typically to quickly create a candidate for a new, critical issue that is being widely reported. Candidates are proposed to the CVE Editorial Board for review and voting.

3) Entry stage: If the candidate has been accepted, the candidate is converted into an entry.

NIST NVD (National Vulnerability Database) [37;38]:

- NIST NVD, formerly known as ICAT, is the U.S. government repository of standards based vulnerability management data.
- It contains information on all CVE identifiers (both candidate status and entry status), adds additional information, such as affected product versions, vulnerability types and severity scores, and provides full database functionality for the MITRE CVE dictionary.

How the NVD is built:

1) Acquisition procedure: NVD is updated immediately whenever a new vulnerability is added to the CVE dictionary of vulnerabilities. New vulnerabilities are then analyzed by NVD analysts on a first-in, first-out basis and augmented with attributes (see below) usually within two U.S. government business days. NVD does not typically carry any persistent analysis backlog. The overall process has not substantially changed during past years.

2) Vulnerability scores: Vulnerabilities are scored by the NVD analysis team regarding their severity. The "Common Vulnerability Scoring System" (CVSS) 2.0 provides for different scores, with score values being between 0 and 10 (highest severity): The base score (used in my analysis) can be refined by considering temporal and environmental characteristics.

   i. The base score is an aggregation of six base score metrics (three are related to how the vulnerability is accessed, three describe the degree of loss of confidentiality, integrity, and availability). This score is mandatory and specified by vulnerability bulletin analysts and software vendors. The NVD team works closely with the CVSS working group and user community to come to a consensus on scoring some of the more commonly-occurring vulnerabilities.

   ii. The temporal score represents the characteristics of a vulnerability that change over time. It aggregates five modifiers (optionally, specified by vulnerability bulletin analysts and software vendors).

   iii. The environmental score represents the characteristics of a vulnerability that are relevant and unique to a particular user's environment. It aggregates three metrics (optionally, determined by users).

   The NVD scoring system changed over time: CVSS 2.0 scores for the CVE vulnerabilities published prior to 11/9/2005 were approximated by the NVD team from

prior CVSS metric data. The investigation of the NVD conversion script reveals that for all CVSS 2 characteristics corresponding CVSS 1 characteristics are available [35] and a "natural" conversion was conducted, which allows comparing scores converted into CVSS 2 with "new" CVSS 2 scores.

3) Affected software and versions: The NVD applies the structured naming scheme CPE provided by MITRE (an example is "cpe:/o:redhat:enterprise_linux:3".) to assign a CPE for each affected version. The NVD uses the MITRE descriptions and data obtained from product vendors to determine vulnerable product versions. On the rare occasions when information sources are not in agreement, the NVD works with MITRE and/or vendors to perform disambiguation.

4) Vulnerability types: The NVD analysis team assigns a type (e.g. buffer overflow) to a vulnerability based on the MITRE CWE (Common Weakness Enumeration). The NVD classification uses only a portion of the overall CWE structure; for the NVD-CWE mapping see http://nvd.nist.gov/cwe.cfm#cwes.

5) Original Release Date: This date corresponds to the date the CVE was originally published in NVD. In contrast, MITRE issues the "Assigned Date" to indicate when a CVE enters the "Assigned" phase of their CVE workflow. The NVD dating methodology has not changed over time.

8. References

[1] M. Schwarz and Y. Takhteyev, "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, 2008.

[2] C. Payne, "On the security of open source software", Information Systems Journal (12:1), 2002, pp. 61-78.

[3] Raymond, E.S. The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary, O'Reilly, Beijing, China, 2001.

[4] E. Levy, "Wide open source", http://www.securityfocus.com/news/19, 2000.

[5] J. Viega, "Open Source Security: Still a Myth", http://www.onlamp.com/pub/a/security/2004/09/16/open_source_security_myths.html, 2004.

[6] O.H. Alhazmi and Y.K. Malaiya, "Measuring and enhancing prediction capabilities of vulnerability discovery models for Apache and IIS HTTP servers", in: Proceedings of the 17th International Symposium on Software Reliability Engineering (ISSRE'06), Washington, DC, USA, 2006, pp. 343–352.

[7] O. Alhazmi, Y. Malaiya, and I. Ray, "Measuring, analyzing and predicting security vulnerabilities in software systems", Computers & Security (26:3), 2007, pp. 219-228.

[8] S. Frei, M. May, U. Fiedler B. Plattner, "Large-Scale Vulnerability Analysis, in: Proceedings of the ACM SIGCOMM 2006 Workshop, November 11, 2006, Pisa, Italy.

[9] R. Gopalakrishna and E.H. Spafford, "A trend analysis of vulnerabilities", Technical Report 2005-05, CERIAS, Purdue University, May 2005.

[10] S. Neuhaus, T. Zimmermann, C. Holler and A. Zeller, "Predicting Vulnerable Software Components", in: Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS 2007), Alexandria, VA, USA, October 2007, pp. 529-540.

[11] S.-W. Woo, O.H. Alhazmi, and Y.K. Malaiya, "An analysis of the vulnerability discovery process in web browsers", in: Proceedings of the 10th IASTED International Conference on Software Engineering and Applications, Dallas, TX, USA, November 13-15, 2006.

[12] S.-W.Woo, O.H. Alhazmi Y.K. Malaiya, "Assessing vulnerabilities in Apache and IIS HTTP servers, in: Proceedings of the 2nd International Symposium on Dependable, Autonomic and Secure Computing, Indianapolis, IN, USA, September 29-October 01, 2006, pp. 103-110.

[13] E. Rescorla, "Is finding security holes a good idea?", in: Proceedings of the Third Annual Workshop on Economics and Information Security, Minneapolis, Minnesota, May 13-14, 2004.

[14] G. Schryen, "Security of open source and closed source software: An empirical comparison of published vulnerabilities", in: Proceedings of Americas Conference on Information Systems, San Francisco, California, August 6 - 9, 2009.

[15] A. Ozment, “The Likelihood of Vulnerability Rediscovery and the Social Utility of Vulnerability Hunting”, in: Proceedings of the Fourth Workshop on the Economics of Information Security, Cambridge, Massachusetts, June 2-3, 2005, pp. 1-21.

[16] Open Source Initiative (OSI), “The Open Source Definition”, http://www.opensource.org/docs/osd, 2006.

[17] J. M. Gonzalez-Barahona, “Free Software/Open Source: Information Society Opportunities for Europe?”, Working group on Libre Software, http://eu.conecta.it/paper/cathedral_bazaar.html, 2000.

[18] Free Software Foundation (FSF), “The Free Software Definition”, http://www.fsf.org/licensing/essays/free-sw.html, 2007.

[19] G. Schryen and R. Kadura, “Open Source vs. Closed Source Software: Towards Measuring Security”, in: Proceedings of the 2009 ACM Symposium on Applied Computing, Honolulu, Hawaii, March 8-12, 2009, pp. 2016-2023.

[20] R. Anderson, “Why Information Security is Hard – An Economic Perspective”, in: Proceedings of the Seventeenth Computer Security Applications Conference, New Orleans, Louisiana, December 10-14, 2001, pp. 358-365.

[21] MITRE, “Common Vulnerabilities and Exposures”, http://cve.mitre.org, 2009

[22] A. Ozment, “Improving Vulnerability Discovery Models: Problems with Definitions and Assumptions”, in: Proceedings of the Third Workshop on Quality of Protection (QoP’07), Alexandria, VA, USA. October 29, 2007.

[23] US-CERT, “Vulnerability Notes Database Field Descriptions”, http://www.kb.cert.org/vuls/html/fieldhelp, 2009.

[24] W.A. Arbaugh, W.L. Fithen and J. McHugh, “Windows of vulnerability: A case study analysis”, IEEE Computer (33:12), 2000, pp. 52–59.

[25] 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.

[26] A. Arora, R. Krishnan, A. Nandkumar, R. Telang, and Y. Yang, “Impact of Vulnerability Disclosure and Patch Availability – An Empirical Analysis”, in: Proceedings of the Third Workshop on the Economics of Information Security, Minneapolis, Minnesota, May 13-14, 2004, pp. 1-20.

[27] D. Nizovtsev and M. Thursby, “To disclose or not? An analysis of software user behavior”, Information Economics and Policy (19:1), 2007, pp. 43-64.

[28] A. Arora, A. Telang, and H. Xu, “Optimal Policy for Software Vulnerability Disclosure”, in: Proceedings of the Third Annual Workshop on Economics and Information Security, Minneapolis, Minnesota, May 13-14, 2004, pp. 52-59.

[29] J.P. Choi, C. Fershtman, and N. Gandal, “Network Security: Vulnerabilities and Disclosure Policy”, Discussion paper, http://www.msu.edu/~choijay/Internet_Security.pdf, 2007.

[30] A. Arora, C.M. Forman, A. Nandkumar, and R. Telang, “Competitive and strategic effects in the timing of patch release”, in Proceedings of the Fifth Workshop on the Economics of Information Security, Cambridge, UK, June 26-28, 2006.

[31] H. Cavusoglu, H. Cavusoglu, and J. Zhang, “Economics of Security Patch Management”, in: Proceedings of the Fifth Workshop on the Economics of Information Security, Cambridge, UK, June 26-28, 2006.

[32] S. Beattie, S. Arnold, C. Cowan, P. Wagle, C. Wright, and A. Shostack, “Timing the Application of Security Patches for Optimal Uptime”, in: Proceedings of Sixteenth Systems Administration Conference, Philadelphia, Pennsylvania, November 3–8, 2002, pp. 233-242.

[33] R. Anderson, “Open and Closed Systems are Equivalent (that is, in an ideal world)”, in: Perspectives on Free and Open Source Software, Feller, J., B. Fitzgerald, S.A. Hissam, and K.R. Lakhani (Eds.), MIT Press, Cambridge, 2005, pp. 127–142.

[34] NetApplications, “Global Market Share Statistics, http://marketshare.hitslink.com, 2009.

[35] Netcraft, “Web Server Survey”, http://news.netcraft.com/archives/web_server_survey.html, 2009.

[36] Gartner, “Gartner Says Worldwide Relational Database Market Increased 14 Percent in 2006”, http://www.gartner.com/it/page.jsp?id=507466, 2007.

[37] NIST, Personal communication with C. Johnson, National Vulnerability Database - Program Manager, Computer Security Division Personal communication, May 2009.

[38] NIST, National Vulnerability Database, http://nvd.nist.gov, 2009.