OPEN SOURCE VS. CLOSED SOURCE SOFTWARE: TOWARDS MEASURING SECURITY

Guido Schryen
International Computer Science Institute
1947 Center Street. Suite 600
Berkeley, CA 94704, USA
+1 510 666 2972
schryen@gmx.net

Rouven Kadura
RWTH Aachen University
Templergraben 64
52062 Aachen, Germany
+49 241 8096184
kadura@inbox.com

ABSTRACT

The increasing availability and deployment of open source software in personal and commercial environments makes open source software highly appealing for hackers, and others who are interested in exploiting software vulnerabilities. This deployment has resulted in a debate "full of religion" on the security of open source software compared to that of closed source software. However, beyond such arguments, only little quantitative analysis on this research issue has taken place. We discuss the state-of-the-art of the security debate and identify shortcomings. Based on these, we propose new metrics, which allows to answer the question to what extent the review process of open source and closed source development has helped to fix vulnerabilities. We illustrate the application of some of these metrics in a case study on OpenOffice (open source software) vs. Microsoft Office (closed source software).

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics – *product metrics, process metrics*

General Terms

Measurement, Security

Keywords

Open source software, Closed source software, Security, Metrics

1. 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, then, come to regard software as a good we have to pay for – be it for either personal or commercial use – 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 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 family of software of this kind is referred to as the umbrella term "open source software". When discussing this alleged innovation in software distribution, we are reminded by Glass [10] that, essentially, free and open source software dates right back to the origins of the computing field, as far back in fact as the 1950s, when all software was free, and most of it open.

The current application fields of open source software are manifold. Internet programs, such as the mail transfer agent Sendmail, the Web server Apache, the operating system Linux, the database MySQL, and the office package OpenOffice are some of the most popular examples. Comprehensive repositories for open source applications, which are already successfully competing with today's binary-only software (closed source software), are provided by the open source software development websites http://sourceforge.net and http://freshmeat.net, the latter maintaining a large index of Unix and cross-platform software.

The increasing availability and deployment of open source software in personal and commercial environments makes open source software appealing for hackers, and others who are interested in exploiting software vulnerabilities. These security flaws become even more dangerous when software is not applied in a closed context, but interconnected with other systems and the Internet (this argument is also valid for closed source software). Naraine [22] reports a study by The Mitre Corp., according to which there are more than 230 open source software packages already in use, even for critical operations, within U.S. federal government agencies and departments. In order to appropriately tackle security concerns regarding the applied packages, the U.S. Department of Homeland Security initiated the so-called Vulnerability Discovery and Remediation, Open Source Hardening Project, which was part of a broad federal initiative to perform daily security audits of approximately 40 open-source applications, including Linux, Apache, and MySQL. All these developments show that open source software has definitely arrived in the world of important and critical software environments that need security protection against attacks. Interestingly, Li et al. [19] find that the portion of security vulnerabilities related to the total bugs fixed has even increased in both software "Mozilla" and "Apache". The discussion on open source security becomes even more relevant when open source software packages are themselves deployed as security

SAC'09, March 8-12, 2009, Honolulu, Hawaii, U.S.A.
Copyright 2009 ACM 978-1-60558-166-8/09/03…$5.00.

instruments, such as virus scanners, intrusion detection systems, password safes or "single sign-on" systems [4]. However, the discussion of whether obscurity outperforms tranparency in terms of security is as old as the frequently referenced work of Kerckhoffs [15].

Picking up the discourse on comparing the security of open source software with that of closed source, one might argue that the former is inherently more secure due to its communal writing and review process. On the other hand, Fisher [7] reminds us that, in 2002, researchers found several vulnerabilities in the open source software "OpenSSL toolkit", all of which were buffer overruns – the most common and preventable flaws in software. Developers can also place back doors in open source software deliberately. Although there is a plethora of articles in the popular on-line press, the observation of Payne [26] that there has been little discussion of this in the academic literature is still valid. Frequently, discussions and arguments are polarizing, and we believe that Herbert Thompson hit the mark when saying "When folks talk about Linux and Windows security, a lot of religion gets involved." ([21], p. 27). Furthermore, we also need to consider that a security validation might be biased depending on the person, role or organization that performs the security analysis. For example, Messmer [21] reports that Security Innovation caused an uproar when it asserted in a study that a Web server based on open source code had twice as many security vulnerabilities recorded for it in 2004 as a comparable Microsoft-based Web server did. According to Messmer, this study was financed by Microsoft.

An unbiased discussion of open source and closed source security is necessary for a validation of the arguments of both open source advocates and closed source advocates. More specifically, we should not primarily address the question of whether open source or closed source software is securer, but should rather focus on the conditions under which open source development and closed source development contribute to enhanced security, in order to give hints about the reasons for flaws, and on how to prevent them in the future. For example, Li et al. [19] empirically find for Mozilla and Apache that, against the belief that buffer overflows are the most common form of security vulnerabilities, semantic bugs cause more than 70% of vulnerabilities.

Summing up, it is helpful, if not necessary, to transparently measure and rate the security of software – be it open source or closed source software [33]. As Bellovin [5], p. 96, quotes Lord Kelvin, "*If you can not measure it, you can not improve it.*" However, measuring security is a challenging task, because security is somehow invisible: the more secure a system is, the less uproar occurs. Despite an increasing number of quantitative research papers on measuring software security in the past years, it is still true what Witten et al. [32], observed in 2001: what the discussion on software security specifically lacks is appropriate metrics, methodology and hard data.

This paper addresses this research gap and contributes to the quantification of (open source and closed source) software security by (a) analyzing limitations of metrics and models defined in previous research, (b) proposing new metrics, which measures to what extent the review process of open source and closed source development has helped to fix vulnerabilities, and (c) applying the metrics in a case study on OpenOffice (open source software) vs. Microsoft Office (closed source software).

The rest of this article is organized as follows: In Section 2, we provide an overview of the recent discussion. Then, in Section 3, we analyze models proposed in the literature. Section 4 presents and discusses the new metric. The data used for the application of the proposed metric in the case study are presented in Section 5. Section 6 provides our empirical results, before we conclude.

2. RECENT DISCUSSION

The discussion on open source and closed source software is affected by the presence of several different understandings. It becomes even more unclear when several open source licenses are mentioned, or further notions, such as "free software", pop up. However, (not only) in the context of the impacts of software models on security, does it seem reasonable to precisely define what open source software is and whether we can identify several categories which need to be treated differently in the security context. Therefore, we briefly clarify these issues. This also helps us to unfold and discuss the security arguments in favor of and against open source software before we present proposed models and empirical findings on measuring software security.

2.1 Terms and Definitions

In any of the understandings the authors are aware of, 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 [25]. The definition particularly 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 modify the original version. For example, one option would be to allow anyone to include source code and to upload it to the software repository (this style of development is referred to as "bazaar style" by Raymond [27], another would be to supervise the modification process by leaving the integration of modification proposals up to "wizards"; this traditional, hierarchical development style is denoted as "cathedral" by Raymond [27]. The implementation of this modification procedure might have an impact on the security of the software, so that a detailed discussion of open source security would need to consider this issue. Summing up, it is important to distinguish between "the product" open source software and its development process. With regard to the latter, we focus on the implementation phase and do not regard other phases in the software development process.

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 [9] provides a definition of " '*free software' [as] a matter of liberty, not price.*" In contrast to the OSD definition, the FSF explicitly focuses on the option of releasing the improvements to the public (freedom 3), 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. Does it comprise all that software that is non-open in a particular context, or does it simply mean that software is distributed in a binary-only form? It might be useful in the context of security evaluation to further specify different types of "closed" software.

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 approaches through the developer's lens and specifies the type of development. Complementarily, one could also adopt a software user's point of view by differentiating between software that needs to be paid for and software for which no fee applies. This dimension takes the pricing model into account. The resulting classification scheme corresponds to a two-dimensional matrix, which is illustrated (with real-world examples) in Table 1.

Table 1. Classification of software

	Open source	**Closed source**
Free of charge	Linux, Apache Web server	Adobe Acrobart Reader
Subject to charge	MySQL	MS Windows OS

2.2 The Debate on Open Source Security

In the debate on the security of open source and closed source software, a set of arguments is repeatedly presented. We present and discuss those arguments that seem to arise most frequently.

While there is consensus 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 peer review process [26] and argue in the sense of Raymond [27] that, "*Given enough eyeballs, bugs are shallow.*" (p. 19). This strength of the review process is assumed to make finding bugs easier and more likely. Beside the argument of an increased number of reviewers, proponents also claim that, for each reviewer, vulnerabilities in closed source software are harder to find than those in software whose source code is readable. However, opponents comment that only techniques differ ([27], p. 67f). For example, closed source software can be disassembled. They further worry that open source code might attract skilled programmers who are actively seeking flaws but who eschew any efforts to find flaws in closed source software. In this context, source code of new software (version), which has not been inspected by many reviewers, is assumed to be particularly endangered.

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. With regard to the availability of an increased number of reviewers, it is countered that not all reviewers tend to have similar experience and expertise. In contrast, experienced reviewers in companies are believed to be even better skilled in finding flaws. The reason for this bias is that often reviewers do not only need to know programming languages but also need to have further skills, such as network or cryptographic skills. For example, Payne [26] mentions a vulnerability found in some implementations of the Secure Shell remote login system protocol version 1.5. Finding this vulnerability required not only an understanding of the protocol itself, but also of advanced issues relating to cryptology. Furthermore, it is queried whether the actual number or reviewers is really as high as assumed: Levy [18] remarks "*Sure, the source code is available. But is anyone reading it?*", and Viega [31] guesses that many potential reviewers do not inspect the code because they believe that others have already done so. Summarizing, advocates of closed source software believe that the closeness of software, which follows the principle "security by obscurity", allows security flaws to be hidden, at least until a patch is publicly available [8]. The authors doubt that this argument is a strong one, since it is difficult, if not impossible, to hide the source code during the time the software is in operation. A salient example is the accidentally published source code of Diebold voting machines on the Internet in 2003 [29]. This source code had been used by voting machines in 37 states of the U.S. Although this source code was certainly involved in critical operations, it was published, even without any criminal efforts being necessary. While this source code was readable for anybody, vulnerabilities could also be detected by scientists, which initiated a public debate. However, in cases where a source code is only available to a few criminals, code hiding may even be counterproductive [8].

With regard to the detection of security flaws in software, it should be equitably noted that not all vulnerabilities are revealed by the source code, be it open or closed. Some flaws might be due to design decisions, but documentations on design are not always available. Other vulnerabilities can infiltrate software if the compiler used to generate binary code is insidiously modified. In this case, the source code does not reveal these vulnerabilities. Thompson [30] demonstrates this principle with C code examples, where even the source code of the compiler does not disclose any malicious elements, although these are integrated into the binary version of this compiler.

Payne [26] argues that security flaws in open source software can be fixed more quickly than those of closed source software, because the user community is not dependent on a company's schedule to release a patch. It can rather control the activities to fix vulnerabilities by itself. However, Payne [26] also notes that the impact of the availability of source code on security might also depend on the open source model used. For example, the (open source) cathedral model would allow essentially anyone to detect vulnerabilities, but not to remove them, because the patching process is regulated and needs time, which can then be used by attackers to exploit the (unpatched) vulnerabilities.

The discussion on security presented above involves a lot of "religion"[21] and is also characterized by general attitudes towards open source and closed source software. However, in order to enlighten the impact of open source on security, we propose the application of measurements, which allow for a fair comparison of open source and closed source software. In the next section, we discuss metrics that have been proposed in the literature and that would support such measurements.

3. REVIEW: QUANTITATIVE MODELS

In the literature, a number of quantitative models for the measurement of security of software systems have been proposed. These models have often be related to reliability and dependability in terms of nomenclature and methodologies [12, 13, 16]. In this section we briefly present the most important security-related models, focussing on security breaches and vulnerabilities. Models that address the economics of disclosing vulnerabilities (see, for example, [23, 28]), are beyond the scope of our work. Finally, we identify the need for further research by summarizing the drawbacks and limitations of existing models and metrics.

Security breaches are incidents, which are due to security vulnerabilities. Adopting a quantitative model of reliability, Littlewood et al. [20] and Kimura [16] use a probabilistic model for the empirical security of software by representing the cumulative number of security breaches as a function with the elapsed time as an independent variable. The model assumes the random variable time up to the next intrusion to be exponentially distributed. However, the authors make no assumptions on the development of the rate parameter λ.

In cases where the total effort in finding vulnerabilities is not linear in time, for example due to a changing number of reviewers with different skills, the elapsed time as the independent variable seems inappropriate and would need to be substituted by the total effort [20]. Another modification of the basic model refers to evaluating the security breaches by considering the cumulative reward gained by the attackers. Jonsson and Olovsson [13] perform a practical intrusion test on a distributed UNIX computer system and collect data related to the difficulty of causing security breaches. On the basis of these data, they formulate the hypothesis that the occurrence of security breaches can be split into three phases based on the attackers' behavior: the learning phase, the standard attack phase, and the innovative attack phase. They further find statistical evidence that the times between consecutive breaches during the standard attack phase are exponentially distributed with a constant rate parameter λ using the (independent) variable attacking worker time. Thus, their findings support the (homogeneous) model of Littlewood et. al [20].

Similar to the observations of Johnsson and Olovsson [13] regarding the development of security breaches, Alhazmi et al. [2, 3] assume that the development of vulnerability discovery, which is a precondition for any intentionally induced security breach, can be split up into three different phases, in which the usage environment and vulnerability detection effort change. In phase 1, the software testers gather sufficient knowledge of the system to break into it successfully. In phase 2, the discovering of vulnerabilities will be most rewarding for both white hat and black hat finders. Finally, in phase 3, the 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 due to a finite number of vulnerabilities. Let y(t) be the total number of vulnerabilities found in period [0,t], A a constant of proportionality, and B the total number of vulnerabilities that would eventually be found in the software, then Alhazmi et al. (2005) consequently assume the vulnerability discovery rate to be given by the differential equation $\frac{dy}{dt} = Ay(B-y)$, resulting in $y = \frac{B}{BCe^{-ABt}+1}$. Using data for both commercial (five versions of Windows) and open-source systems (two versions of Red Hat Linux), Alhazmi et al. [2] find statistical evidence for their model for both closed source software and open source software. Interestingly, following the assumption of the model that the total number of eventually found vulnerabilities is given by B, it provides a procedure for determining B and, thereby, for determining the number of still undetected vulnerabilities. Comparing their figures of B (rounded up) with the current numbers on detected vulnerabilities (bracketed), as provided by the U.S. National Vulnerability Database Version 2.0, we get these figures: Windows 95: B=49 (46), Windows 98: B=66 (91), Windows XP: B=88 (257), Windows NT: 153 (234), Windows 2000: 163 (345), Red Hat Linux 6.2: 123 (64), and Red Hat Linux 7.1: 163 (36). The gaps between predicted and current figures show that the number of detected vulnerabilities of some systems are strongly underestimated in the model of Alhazmi et al. [2]. Therefore, their model needs to be re-evaluated with particular regard to approximating the development of detected vulnerabilities in phase 3.

As mentioned above, time-based models become inappropriate when the total effort that is spent on detecting vulnerabilities is not linear in time. A model that considers this issue is presented by Alhazmi and Malaiya [1], who assume the effort to detect vulnerabilities of a software system to depend upon the number of computers on which the particular software is installed. More specifically, they define the effort E as $E = \sum_{i=0}^{n}(U_i \times P_i)$, where U_i is the total number of users of all systems at the period of time I, and P_i is the percentage of the users using the system. They further assume, in analogy to their time-based model [2], that the vulnerability detection rate is proportional to the fraction of remaining vulnerability. On the basis of these assumptions, they hypothesize that the number of vulnerabilities is given by $y = B(1-e^{-\lambda_{vu}E})$, where λ_{vu} is a parameter and B represents the number of vulnerabilities that would eventually be found. They find statistical evidence for the validity of this effort-based model for the operating systems Windows 98 and NT 4.0.

Like Littlewood et al. [20], Rescorla [28] adopts a model provided by the literature on software reliability. More specifically, he uses the probabilistic G-O model presented by Goel and Okumoto [11], which models the number of vulnerabilities over time with a non-homogenous Poisson process. This model assumes the expected value of the Poisson process to be proportional to the number of undiscovered vulnerabilities at time t. The model also assumes that all vulnerabilities will eventually be found. On the basis of the non-homogenous Poisson process that the G-O model features, Rescorla [28] fits an exponential of the form $Ae^{-t/\Theta}$ to the curve of vulnerability. Then, the total number of vulnerabilities N can be computed by $N = A\Theta$, where A is a constant.

However, in his empirical analysis of vulnerabilities of (both open source and closed source) operating systems, namely Windows NT 4.0, Solaris 2.5.1, FreeBSD 4.0 and RedHat Linux 7.0, Rescorla [28] finds no (strong) statistical evidence that the G-O model appropriately approximates the number of detected vulnerabilities over time.

Having reviewed the literature on the quantitative security analysis in the context of "open source versus closed source software", we can identify the following problems and limitations:

- There is only little literature on measuring software. Those metrics and models that have been applied to security are mostly adopted from the research field of reliability. Particularly, to the knowledge of the authors, no models have been developed that address the discourse on open source versus closed source security. There is a strong need for the development of metrics and models dedicated to measuring and comparing software security.

- The set of empirical investigations is small and mainly focuses on the analysis of operating systems. We assume that these limitations are strongly related to the scarcity of security data.
- Beside the problem of data scarcity, many authors struggle with the availability and quality of data, particularly in terms of incompleteness and a low level of granularity.
- Up to now, software security has been addressed like an "atom". Only very few authors split it up into components. Particularly in the context of security assessment of open source development versus closed source development, it seems reasonable to zoom in on the "bundle security assessment" to analyze the extent to which elements of security are supported by these different types of software development. One option would be to follow Payne's [26] path by separately considering security requirements, such as availability and confidentiality. However, we propose the following of a more software-technological dimension, which differentiates according to the type of vulnerability source. For example, vulnerabilities can occur due to software design, due to implementation faults, such as buffer overflows, or due to software environment problems, such as the usage of faulty libraries or operating system calls. By employing this classification, we would be able to assess the appropriateness of the software development type with regard to security (maintenance) in more detail.
- An assumption of many models is the finite number of vulnerabilities that a software features or that are detected over the software's lifetime. This assumption needs to be scrutinized when we accept the option that patches not only eradicate vulnerabilities, but also create new ones.
- Most existing models on security measurement are related to the number of detected vulnerabilities or exploitations. However, this is only one aspect of the quantification of software security, which needs to be complemented by further evaluations. For example, the assessment of the severity of vulnerabilities is no less important.
- The demand for developing models that address software security has led the security measurement community underemphasizing the discussion of metrics to be used in the models.

The next section addresses the challenge of quantifying software security by proposing a new metric that allows to measure to what extent the review process of open source and closed source development has helped to fix vulnerabilities.

4. NEW SECURITY METRICS

As vulnerabilities are the root of exploitations and security breaches, the measurement of vulnerabilities is the right point at which to start quantifying software. However, the number of vulnerabilities does not necessarily reflect the level of security of a program. For example, if program A features only one vulnerability that is easy to discover, can be exploited systematically and causes severe damage, then A can be felt to be less secure than a program B that features ten vulnerabilities, each of them being extremely hard to discover, can be exploited only in the presence of specific conjunctures, and does not cause any severe harm. Therefore, we propose weighting vulnerabilities. Let n be the number of vulnerabilities in time window [0;t] and vs_i, i=1..n, be the (normalized) severity of vulnerability i with $vs_i \in [0;1]$. Then, the cumulated weighted vulnerability CWV(t) in time window [0;t] is calculated by summing up all vs_i, i.e. $CWV(t) = \sum_{i=1}^{n(t)} vs_i$. A practical example for the "severity of vulnerabilty" concept is the NIST Common Vulnerability Scoring System (CVSS), which assigns a (aggregated) score between 0 and 10 to each vulnerability (normalization is straightforward here). If we further categorize the vulnerabilities according to their type j, such as "buffer overflow" or "faulty library", we can compute type-specific CWVs by $CWV_j(t) = \sum_{i=1}^{n(t)} \delta_{ij} \times vs_i$ with $\delta_{ij} = 1$, if vulnerability i belongs to type j and $\delta_{ij} = 0$ else.

This segregation of vulnerabilities according to their type allows us to identify the most critical (types of) security defects, so that we can discuss the impact of open source and closed source development on security broken down into defect types. We would like to stress that the categorization of vulnerabilities can occur along different dimensions; for example, it can be based on the type (buffer overflows, cross-site scripting etc.) that cause vulnerabilities, but also on the resulting impact (violation of integrity, confidentiality etc.) of intrusions or on the impact on business value [6]. In the literature, mainly the first option is adopted, but we also find it interesting to discuss the other paths; however, this discourse is out of scope of this paper.

However, the availability of scores and the resulting opportunity to derive conclusions on a metric scale level is dangerous, when scores were improperly determined on the basis of ordinal rankings, thereby reflecting only a seemingly accuracy. The Common Vulnerability Scoring System Version 2 Calculator, which is applied in the CVSS, implements such a misleading procedure. A way out of this problem would be to remain on ordinal level by providing vulnerability severity classes (for example low, medium, high) and to simply count the vulnerabilities for each class without aggregating the numbers. We, then, obtain the cumulated unweighted vulnerability $CUV^k(t)$, with k being the severity class, by $CUV^k(t) = \sum_{i=1}^{n(t)} \delta_{ik}$ with $\delta_{ik} = 1$, if vulnerability i belongs to severity class k and $\delta_{ik} = 0$ else. Analogously to the calculation of CWV, we can also determine CUV^k specific to vulnerability type j (for example buffer overflow) by $CUV_j^k(t) = \sum_{i=1}^{n(t)} \delta_{ijk}$ with $\delta_{ijk} = 1$, if vulnerability i belongs to severity class k and to vulnerability type j, and $\delta_{ijk} = 0$ else.

However, we still need to carefully observe the procedure with which classes are used. For example, the NIST National Vulnerability Database does not only provide a score for each vulnerability, but also a vulnerability class/ranking. Unfortunately, *"[...] these qualitative rankings are simply mapped from the numeric CVSS scores*" (http://nvd.nist.gov/cvss.cfm).

The metrics discussed so far relate to the vulnerabilities that have been revealed during the process of reviewing software. However, they do not consider the extent to which the review process has helped to fix the vulnerabilities. As this issue is particularly relevant to the discussion of whether open source software or closed source software is more secure in practice, we now address it in more detail.

With regard to the elimination of vulnerabilities by the provision of (security) patches, it seems less reasonable to measure the number or intensity of patches, because this provides no information on the number of covered vulnerabilities or on the ages of covered vulnerabilities. It seems rather appropriate to compute (statistical data on) the reaction time between detection and elimination of a vulnerability, weighted by the level of severity of the vulnerability. It might also seem reasonable to record how many of the detected vulnerabilities are unpatched: Let i be the index of the event that a vulnerability is either announced or patched, t_i be the corresponding point of time, pv_{t_i} be the (possibly severity-weighted) number of detected and patched vulnerabilities in the time window $[0;t_i]$, and let uv_{t_i} be the corresponding (possibly severity-weighted) number of unpatched vulnerabilities. Then, we define the patch index at time t_n (PI_{t_n}) by $PI_{t_n} = \frac{1}{t_n}\sum_{i=1}^{n-1}(t_{i+1}-t_i)\times\frac{uv_{t_i}}{pv_{t_i}+uv_{t_i}}$.

The sum corresponds to the shaded area in Figure 1, the quotient $1/t_n$ normalizes $PI_{t_n} \in [0;1]$. It should be noted that t_1 corresponds to the point in time where the first vulnerability is detected. Because of the normalization being inherent in PI , PI=0 would imply that, for all announced vulnerabilities, a patch is already provided at the day of announcement. In contrast, PI=1 would imply that none of the announced vulnerabilities has been patched.

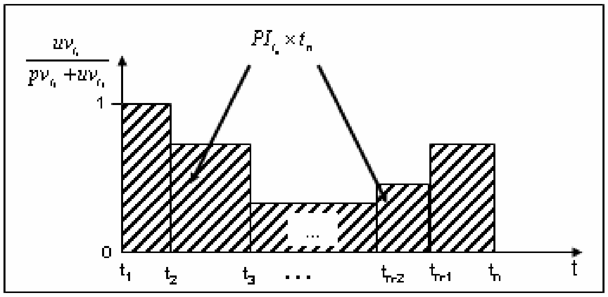


Figure 1. Visualization of patch index (PI)

It should also be noted that the proposed patch index does not reflect a security level at a specific point of time, but rather mirrors the level of community patching activities with regard to both the number of unpatched vulnerabilities in relation to all vulnerabilities and the time having been consumed for fixing; therefore, the patch index is relative in nature. At the beginning, the level does not provide any valuable data, but it becomes a significant factor after some time has gone by and several vulnerabilities have been detected. However, the proposed patch index is time-invariant what needs to be discussed or modified in future research:

- The shaded rectangles in Figure 1 are considered regardless of their horizontal position. This issue might lead to overemphasizing the meaning of early (unpatched) vulnerabilities, particularly when the sizes of consecutive rectangles are comparably small.
- The treatment of exposed vulnerabilities is time-invariant in that the patch index does not consider whether the exposed vulnerabilities were just recently announced or whether they are already known for a long time.

5. DATA

In order to exemplify and apply the proposed patch index, we apply it to data gained for the closed source software "Microsoft Office" (considering only Word, PowerPoint and Excel and starting with the version released in 2002) and the open source software "OpenOffice" (excluding the database program introduced in version 2.0) for the period from 1 October 2001 to 11 March 2008. The reasons for choosing these software bundles are rooted in the fact that (1) they provide essentially the same functionality, (2) comprehensive security data is available, and (3) they are well-known in the software (security) community.

We consider only those vulnerabilities that have been accepted as Common Vulnerabilities and Exposures (CVE) entries by the CVE editorial board, which was itself created by the MITRE corporation (http://cve.mitre.org). Each of these vulnerabilities has a unique identifier, e.g. CVE-1999-0067, which is used as a reference in many other vulnerability databases (http://cve.mitre.org/compatible/vulnerability_management.html). Among these databases, we use one of the most comprehensive, the NIST "National Vulnerabilty Database" (http://nvd.nist.gov/), which provides full CVE database functionality and offers sophisticated search options. We obtain further details of the vulnerabilities from the MITRE website, the US-CERT Vulnerability Notes Database, Microsoft Security Bulletins, OpenOffice.org and "The Open Source Vulnerability Database" (http://osvdb.org).

6. EMPIRICAL RESULTS

We first address the number of vulnerabilities found for each software. Table 2 shows the numbers, categorized according to their severity. The table entries correspond to what, in Subsection "Security vulnerabilities", is referred to as "cumulated unweighted vulnerability CUV^{k}", with k being the severity class. The severity score follows the NIST Common Vulnerability Scoring System (CVSS), the categorization of types also follows NIST, which adopts a subset of the Common Weakness Enumeration (CWE) list that provides a comprehensive categorization of vulnerability types and is maintained by the MITRE corporation.

The central findings regarding the announced vulnerabilities are:

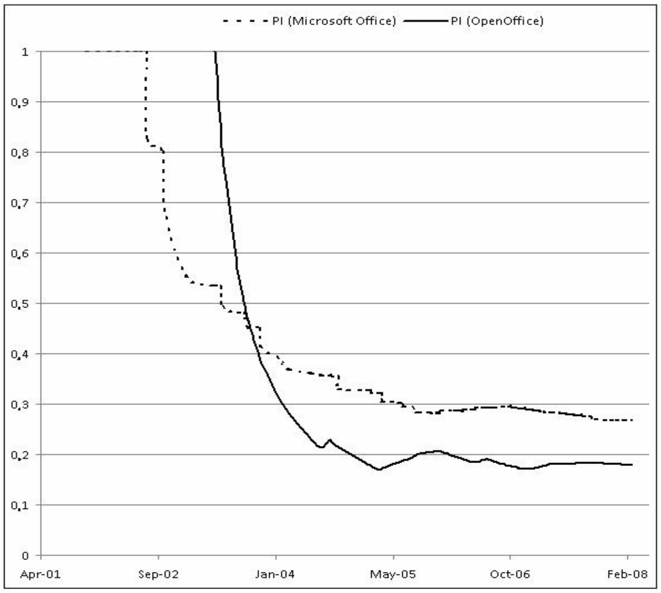
- MS Office has attracted about 7 times more vulnerabilities than OpenOffice has. However, we have to consider that probably more vulnerabilities in OpenOffice than in MS Office might have been existed, detected, potentially discussed in forums, and finally removed, before they could have become a CVE vulnerability.
- Both software bundles have suffered only minor low-severity vulnerabilities; medium- and high-severity vulnerabilities have occurred almost equally often.

Table 2. Number of vulnerabilities (CUV^k) of MS Office (M) and OpenOffice (O)

Vulnerability severity class k						Sum	
Low (0-3.9)		Medium (4.0-6.9)		High (7.0-10.0)			
M	O	M	O	M	O	M	O
3	2	50	6	55	8	108	16

Having analyzed the announced vulnerabilities, we now address the question of the extent to which the review process of open source development has helped to fix vulnerabilities. For this purpose, we apply the patch index, as defined in Section 4. We do not apply any weighting of vulnerabilities, because CVE data are essentially on ordinal scale level, as discussed in Subsection 4. Figure 2 shows the development of the patch index for both software bundles.

Figure 2. Patch indices of MS Office and OpenOffice



Both curves feature a strong decrease at the onset, before leveling off. The strong decrease of both curves is due to the fact that, in the beginning, the presence of unpatched vulnerabilities is "overemphasized", as the total number of vulnerabilities is low. In order to weaken this early impact on the overall development of the patch index, it might be reasonable to integrate some kind of "weighting vulnerabilities" in future work.

Interestingly, although the total numbers of vulnerabilities found in MS Office is about 7 times higher than the OpenOffice-related number, the (levelled off) patch index of MS Office does not reveal a comparably weak performance in patching vulnerabilities. With regard to MS Office, on average, about 27% of all announced vulnerabilities have not been patched, the corresponding value of OpenOffice is 18%. However, these results do not necessarily mean that MS Office vulnerabilities are slower patched than those of OpenOffice. By contrast, a simple statistical analysis of patch times reveals that, on average, MS Office vulnerabilities (the median is 67.5 days, the mean is 87 days) are more quickly patched than OpenOffice vulnerabilities (the median is 85 days, the mean is about 87.4 days). The reasons for these divergent results are that (1) the patch index is invariant in which vulnerability is patched and (2) the patch index also considers both the total number of vulnerabilities detected and the total number of vulnerabilities patched. Interestingly, in the period under consideration, the overall proportions of unpatched vulnerabilities are almost equal (MS Office: 14/108 ≈ 13%, OpenOffice: 2/16 = 12.5%).

This investigation demonstrates that – in contrast to statistical data about patch times – the proposed patch index is capable of considering both the extent to which a certain type of software development creates vulnerabilities and removes vulnerabilities. Therefore, the patch index represents a metric that allows for comprehensively measuring and comparing practical software security. However, the observed patch times need to be interpreted very carefully for two reasons: (1) Data refer to one investigation only. (2) Patch times for both closed and open source development heavily depend on the concrete patching procedures in the responsible organizations or communities. Therefore, it would be necessary to consider whether the particular open source software development is realized in bazaar or in cathedral style (according to OpenOffice.org [24], cathedral style seems to dominate the development of OpenOffice).

7. DISCUSSION AND OUTLOOK

Discussions in the literature show that the increasing availability and the deployment of open source software in personal and commercial environments has resulted in a debate "full of religion" on the security of open source software compared to that of closed source software. This debate reveals the much more comprehensive problem of assessing security, which has traditionally only rarely been conducted on a quantitative level. Although some methods and metrics have been proposed and applied in empirical research, the methodology is at an early stage and is mainly adopted from the fields of reliability and dependability, without careful investigation into the extent to which it can be adopted or the question of whether new models and metrics are required. For example, one assumption of some models is the finite number of vulnerabilities that software features or that are detected over the software's lifetime. This assumption needs to be scrutinized when we accept the option that patches not only eradicate vulnerabilities, but also create new ones. Furthermore, most existing models on security measurement are related to the number of detected vulnerabilities or exploitations. However, it is only one aspect of the quantification of software security. For example, the assessment of the severity of vulnerabilities is no less important. Beyond the problems related to the scarcity of appropriate models, we also face the crux that the set of empirical investigations is small and mainly focused on the analysis of operating systems. We assume that these limitations are strongly related to the scarcity of security data.

This paper starts to bridge the sketched gaps by proposing metrics that allow for quantitatively comparing software development styles with regard to resulting security. The application of the proposed metrics, for which reliable data are available, shows that, overall, OpenOffice has been more secure than MS Office in terms of vulnerabilities. We suggest refining the metrics and extending the analysis onto more software bundles. A further step beyond such activities would be automated security evaluation, which enables us to continuously monitor (the development of) software security and, particularly, to empirically answer the question of when, and to what extent, which software development style leads to more secure software. More specifically, we find it appropriate to analyze which vulnerability types (with regard to their roots) are best addressed by which software development style. We would then try to group software into components, each of which is homogeneous in the roots of potential vulnerabilities. For example, all input validation tasks could be integrated in an I/O module. This module would then be developed according to the development style that best addresses the roots of such vulnerabilities. It should be noticed that the categorization of vulnerabilities does not necessarily needs to be done along the roots of vulnerabilities. It can also occur along the resulting impact (violation of integrity, confidentiality etc.) of intrusions or along the impact on business value [6].

8. REFERENCES

[1] Alhazmi, O. and Malaiya, Y. Quantitative Vulnerability Assessment of Systems Software", *Annual Reliability and Maintainability Symposium*, 1995, 615-620.

[2] Alhazmi, O., Malaiya, Y. and Ray, I. Security Vulnerabilities In *Software Systems: A Quantitative Perspective in Data and Applications Security 2005,* LNCS 3654, 2005, 281-294.

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

[4] Ardagna, C.A., Damiani, E., Frati, F. and Reale, S. Adopting Open Source for Mission-Critical Applications: A Case Study on Single Sign-On. In Damiani, E., Fitzgerald, B., Scacchi, W., Scotto, M. and Succi, G. (Eds.) *Open Source Systems*, Springer, Boston, 2006, 209-220.

[5] Bellovin, S.M. On the Brittleness of Software and the Infeasibility of Security Metrics. *IEEE Security & Privacy*, 4, 4 (2006), 96.

[6] Chen, Y., Boehm, B. and Sheppard, L. Value Driven Security Threat Modeling Based on Attack Path Analysis. In *Proceed. of the 40th Hawaii International Conference on System Sciences*, 2007.

[7] Fisher, D. Open source: a false sense of security? *eWeek, 19,* 39 (2002), 20-21.

[8] Ford, R. Open vs. Closed Software. *ACM Queue,5,* 1 (2007).

[9] Free Software Foundation (FSF) *The Free Software Definition*, 2007.

[10] Glass, R.L. A look at the economics of open source. *Comm. of the ACM, 47*, 2(2004), 25-27.

[11] Goel, A.L. and Okumoto, K. Time-Dependent Error-Detection Rate Model for Software and Other Performance Measures, *IEEE Transactions on Reliability, 28,* 3(1979), 206-211.

[12] Jonsson, E. and Olovsson, T. On the Integration of Security and Dependability in Computer Systems. In *IASTED International Conference on Reliability, Quality Control and Risk Assessment*, 1992.

[13] Jonsson, E. and Olovsson, T. A Quantitative Model of the Security Intrusion Process Based on Attacker Behavior. In *IEEE Transactions on Software Engineering, 23,* 4 (1997), 235-245.

[14] Jonsson, E., Strömberg, L. and Lindskog, S. On the functional relation between security and dependability impairments. In *Proceedings of the 1999 Workshop on New Security Paradigms*, 2000, 104-111.

[15] Kerckhoffs, A. La cryptographie militaire. *Journal des sciences militaires* IX, 1883, 161–191.

[16] Kimura, M. Software vulnerability: definition, modelling, and practical evaluation for e-mail transfer software. *International Journal of Pressure Vessels and Piping, 83,* 4 (2006), 256-261.

[17] Laprie, J. C. (ed.) *Dependability: Basic Concepts and Terminology*, Springer, Austria, 1992.

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

[19] Li, Z., Tan,L., Wang, X., Lu, S., Zhou, Y. and Zhai, C. Have things changed now? – An empirical study of bug characteristics in modern open source software. In *Proceedings of the 1st workshop on Architectural and system support for improving software dependability* , 2006, 25-33.

[20] Littlewood, B., Brocklehurst, S., Fenton, N., Mellor, P., Page, S., Wright, D., Dobson, J., McDermid, J., and Gollmann, D. Towards Operational Measures of Computer Security, *Journal of Computer Security,2,* 3 (1993), 211-229.

[21] Messmer, E. Open source vs. Windows: security debate rages. *Network World, 22,* 26 (2005), 26-27.

[22] Naraine, R. DHS backs open-source security, *eWeek, 23,* 3 (2006), 20.

[23] Nizovtsev, D. and Thursby, M. To disclose or not? An analysis of software user behaviour. *Information Economics and Policy, 19,* 1 (2007), 43-64.

[24] OpenOffice.org *Active Projects of OpenOffice.org, 2008,* http://projects.openoffice.org/index.html#components.

[25] Open Source Initiative (OSI) *The Open Source Definition*, 2006.

[26] Payne, C. On the security of open source software. *Inform. Systems Journal, 12,* 1 (2002), 61-78.

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

[28] Rescorla, E. Is finding security holes a good idea? *Workshop on Econ. and Info. Security*, 2004.

[29] Rubin, A. *Brave New Ballot,* Morgan Road Books, New York, USA, 2006.

[30] Thompson, K. Reflections on Trusting Trust. *Comm. of the ACM, 27,* 8 (1984), 761-763.

[31] Viega, J. *The Myth of Open Source Security*, 2000.

[32] Witten, B., Landwehr, C. and Caloyannidis Does open source improve system security?, *IEEE Software, 18,* 5(2001), 57-61.

[33] Wolfe, M. See Past Self-Proclaimed Experts' Open-Source Security Evaluations, *Comm. of the ACM, 50,* 5 (2007).