

# **Vulnerabilities and Patches of Open Source Software: An Empirical Study**

Kemal Altinkemer, Jackie Rees, and Sanjay Sridhar

Krannert Graduate School of Management  
and  
The Center for Education and Research in Information Assurance and Security (CERIAS)  
Purdue University  
West Lafayette, IN 47907-2056  
{kemal, jrees, ssridhar}@purdue.edu

## **Vulnerabilities and Patches of Open Source Software: An Empirical Study**

### **Abstract**

Software selection is an important consideration in managing the information security function. Open source software is touted by proponents as being robust to many of the security problems that seem to plague proprietary software. This study empirically investigates specific security characteristics of open source and proprietary operating system software. Software vulnerability data spanning several years are collected and analyzed to determine if significant differences exist in terms of inter-arrival times of published vulnerabilities, median time to release patches, type of vulnerability reported and respective severity of the vulnerabilities. The results demonstrate that open source and proprietary operating system software are each likely to report similar vulnerabilities and that open source providers are only marginally quicker in releasing patches for problems identified in their software. The arguments favoring the inherent security of open source software do not appear to hold up to such analysis. These findings provide guidance to security managers to focus on holistic software security management, irrespective of the proprietary-nature of the underlying software.

**Keywords:** open source software, information security

# **Vulnerabilities and Patches of Open Source Software: An Empirical Study**

## **1. Introduction**

Software selection is an important consideration for managing the information security function. Utilizing software packages with strong security features can reduce the information security risk exposure of an organization. Additionally, the underlying quality and security of a technology under consideration has become increasingly important in total cost of ownership (TCO) and other calculations of business value, the logic being that better quality software is cheaper to manage in the long run [24]. Open source software (OSS) has been cited as a possible solution to the information security problems and vulnerabilities often reported in proprietary software. Open source software is software that by license provides unlimited access to the source code, so that the source code can be examined and modified according to the user's wishes. There are also prescriptions for the distribution of the software and subsequent modifications [23]. According to a recent Wall Street Journal article, supporters of open source software, particularly Linux, a PC-based operating system, viewed such software as less vulnerable to viruses and worms, making OSS a favorable choice over other operating systems, such as Windows XP [6]. Open source software is often developed and maintained by large numbers of volunteers. One of the oft promoted features of open source software is that the wide availability of source code, and hence the large number of critical eyes examining the source code, results in more robust and therefore more secure software and applications [23]. Many critics also suggest the push to market pressures among proprietary software vendors result in more problematic end products, including an increased number of software bugs and other problems. In light of these commonly-held beliefs, there is a growing perception that open source software, for example the various

instantiations of the Linux operating system and various software applications, is inherently more secure, due to the freely available source code and greater levels of critical scrutiny.

Information security activities, in theory, are driven by risk management principles. Anti-virus software, firewalls, access control, and intrusion detection systems are certainly important in managing the risk exposure of the organization. Increasingly, organizations are looking to the information technology infrastructure itself, to ensure that components are utilized that provide the most functional and secure platform possible while minimizing the total cost of ownership [27]. Given the general low acquisition costs and perceived higher levels of security, OSS such as Linux, Apache, and MySQL has received significant attention from the popular press. However, it remains unclear whether or not the deployment of OSS results in greater levels of security and therefore lower levels of risk to the organization. The question of reduced TCO remains for future research.

The objective of this research is to empirically determine if there is any validity to the claim of OSS proponents that OSS is more secure, using vulnerability and patch data available for both open source and proprietary software. While well-designed and secure software is much more than the collection of vulnerability and patch release data, these metrics provide at least a preliminary look into potential problems with specific technologies. In this research we focus on operating systems software, such as Linux, Microsoft XP and Apple OS. Data have been collected for a number of years, including the most recent available. We investigate and address the counter-criticism that the reason open source software appears to be less vulnerable is that it simply is less of a target, both in terms of market share and length of time in the market.

The rest of the paper is organized as follows: Section 2 examines the background of the open source versus proprietary software debate. We look at the context used to examine the data,

including the software reliability literature. We also examine previous work in this area and differentiate the current research from this work. The research questions and framework are presented in Section 3. The methodology and information about the data is discussed in Section 4. Section 5 provides the results and discussion while Section 6 concludes the study and presents ideas for future research.

## **2. Related Background**

Open source software is defined as software whose source code is publicly available for free or a nominal charge. Depending on the specific license agreement, the source code may be modified and redistributed. Software whose source code is not openly published, usually are commercial software, which we refer to as proprietary software. As the executable code may be free or of nominal charge for either propriety or open source software, we limit the distinction to the source code. The software development and maintenance work carried out for open source software, is often voluntary and performed by any number of hobbyists and/or other interested parties. The incentives and motivation for volunteering time and effort to open source projects has been studied by [13] and [19]. In general, career concerns and peer recognition are motivating factors for those involved in open source projects. Open source software was modeled using the theory of public goods by [16] who reported inefficiencies in open source development and distribution as compared to proprietary software.

A vulnerability is any bug or error in a user-application or a network application that can be exploited to compromise a system, or cause a security breach. Vulnerabilities can be classified according to the types of problems or breaches it can cause, or the potential damage it may inflict. The problems or threats created by vulnerabilities are qualified based on the nature of possible security attacks and the level of severity of the attacks. Typically, developers working on software

take preventive measures to counter these threats due to known vulnerabilities by releasing what is known as a fix or a patch to eliminate the vulnerability.

There are numerous issues with patching vulnerabilities. First of all, the vulnerability must be discovered and reported. There are different mechanisms by which vulnerabilities can be reported. For example, CERT [8] is a partnership between the federal government and public and private companies, and acts as an infomediary between those who identify vulnerabilities and software users. The ICAT Metabase is a database maintained by the National Institute of Standards and Technology (NIST) (<http://icat.nist.gov/icat.cfm>) [14], that stores and maintains vulnerability and patch information, collected from various other security advisories maintained by the Center for Education and Research in Information Assurance and Security (CERIAS), Security Focus, Bugtraq, etc. There also exist market-based infomediaries such as iDefense (<http://www.iDefense.com>), providing incentives like monetary rewards for vulnerability identifiers. Arora, Telang and Xu in [1] and Kannan and Telang in [17] examine a number of scenarios to study optimal vulnerability disclosure policy and find that the social welfare is maximized by the use of a neutral intermediary. However, there is a contingent of users who believe in immediate and full disclosure (<http://lists.netsys.com/mailman/listinfo/full-disclosure>). The chief concern is that if a particular vulnerability is known, an attack can be executed before affected systems are somehow patched or secured.

Next, a patch must be created, delivered or made accessible to all relevant users of the affected systems. The current system of patch delivery depends on the particular source or vendor of the technology. Some might be automatically located and applied via the Internet, whereas others might require far more work and expertise to locate. Third, the patch must be applied once it is obtained. While some patches and updates are simple to download and install, others have

been known to cause further problems and system instability, for example Microsoft's Service Pack 2 (SP2) [28]. It has been conjectured that such patches might introduce further vulnerabilities as they are sometimes quickly put together without sufficient development and testing procedures. Also, the inherent complexity of many software applications and operating systems makes it practically impossible to completely test the impact of any particular modification or improvement.

The proliferation of patches in itself has resulted in a new area of practice and research, namely patch management. Certainly, keeping up with the volume of patches for various systems is no easy task for organizations. Intuitively, well-developed software that was engineered for both functionality and security should result in fewer bugs, and therefore, fewer vulnerabilities and subsequent patches. From a TCO perspective, patching is quite expensive. Much effort is required to determine which patches should be downloaded and applied at what times, to which systems and follow-up must be performed to ensure adequate systems operations and stability [27]. From a risk management perspective, the application of large numbers of patches implies that further problems, complexities and instability could be introduced to a system, therefore increasing the likelihood of more problems or failure. Both TCO and risk management dictate that fewer patches as a result of fewer vulnerabilities in software is desirable.

Open source software advocates claim that OSS is much less prone to attacks from viruses and other information security problems as there are fewer vulnerabilities to exploit. Others have proposed that the issue is more of simply being an interesting target to attack. For example, as Microsoft is the dominant player in the personal computer operating system market, [12] they are the most interesting target for attackers and others with malevolent agendas. As Linux makes further inroads into the PC operating system market space, will the various

instantiations of Linux become popular targets for virus writers and other attackers as well? The next section sets forth several research questions, along with the research framework in which we attempt to address the research questions. The vulnerabilities and patching behaviors are then examined using these questions, in Section 4.

### 3. Research Questions

The premise for this research has evolved from the concept of software reliability. Prior research examined software reliability from a technical viewpoint, and was often formulated as analytical models [15, 22, 26, 29]. In [29], Sumita and Shantikumar modeled software reliability as a multiple birth and death process. The major assumption is that a “fix” may introduce zero or multiple bugs and eliminate zero or multiple several bugs. Software reliability studies have focused on such issues as mean time to failure and release times. For example, Jelinski and Moranda in [15] and Shooman in [26] presented a well-known reliability model that has its roots in hardware reliability.  $R(t)$  is the probability that no errors will occur from time 0 to time  $t$ ; this is the reliability function.  $F(t)$  is the failure function; the probability that an error will occur in the time interval 0 to  $t$ .

$$F(t) = 1 - R(t) \quad (1)$$

The probability density function of  $F(t)$  is

$$f(t) = \frac{dF(t)}{dt} = \frac{-dR(t)}{dt} \quad (2)$$

A hazard function  $z(t)$  can be defined as the conditional probability that an error occurs in the interval  $t$  to  $t + \Delta t$ , assuming that the error did not occur before time  $t$ . If  $T$  is the time that the error does occur,

$$z(t)\Delta t = P\{t < T < t + \Delta t \mid T > t\} \quad (3)$$

This expression is equivalent to



$$z(t)\Delta t = \frac{P\{t < T < t + \Delta t\}}{P\{T > t\}} = \frac{F(t + \Delta t) - F(t)}{R(t)} \quad (4)$$

Dividing both sides by  $\Delta t$  and taking the limit as  $\Delta t$  approaches zero,

$$z(t) = f(t) / R(t) = [-dR(t) / dt] / R(t) \quad (5)$$

Solving for  $R(t)$  by setting  $R(0) = 1$ ,

$$R(t) = \exp\left(-\int_0^t z(x)dx\right) \quad (6)$$

and mean-time-to-failure (MTTF) is

$$MTTF = \int_0^{\infty} R(t)dt \quad (7)$$

Numerous extensions exist to this original model. Additionally, other models have been developed using different assumptions, such as error seeding models and complexity models [22]. Banker, Datar, Kemerer, and Zweig in [3] point out that a major limitation of these models is that they offer little explanatory power. They proposed a conceptual model which attempts to better understand why errors occur or are introduced and determined that factors such as programmer experience, system volatility and complexity and frequency of minor modifications to the system impacted the error rates.

We are interested in software reliability from an information security viewpoint. Therefore, we consider the errors found in software to represent a superset of vulnerabilities that could lead to exploits and other security threats. We do not know the exact size of the vulnerability set relative to the set of all errors, however given the advances in software engineering practices and automation, we assume that security vulnerabilities represent the large majority of errors discovered in software post-release. Therefore, the discovery of software errors or bugs and vulnerabilities and the subsequent release of patches appear as ideal starting points for examining the security and reliability of various types of software. Arora, Nandkumar, Krishnan,

and Telang in [2] looked at vulnerabilities and frequency of attacks, focusing on vulnerability disclosure policies. From a decision-making perspective, two papers ([4], [7]) examined the optimal time to apply a software patch. Evaluating software quality is critical in cost-effective software maintenance, [5], as too little quality translates into too much software life-cycle costs. In this research, we take an additional look at vulnerability disclosures and their fixes, in an attempt to compare open source and proprietary software, along the lines of software quality and reliability and gain insights into the development process of open source software. How well operating system vendors deal with security problems is bigger than just quick patch release and how easily the vendor enables administrators to apply those patches.

“...the key questions in judging operating systems are: how quickly does an operating system vendor fix public security vulnerabilities; how severe are those problems, compared with other vendors.....” ([21], p. 1)

Given that OSS source code is much more widely available, it is commonly believed that more “critical eyes” are examining open source software in its development and debugging [10]. If more individuals are involved in the development and maintenance of open source software, repairs or patches for problem in the software should be issued in less time than for proprietary software products. Based on this belief, we expect open source to be faster in its response to fixing bugs. Formally, we state Hypothesis 1 as follows:

***Hypothesis 1: Open source software developers issue patches faster than proprietary software vendors.***

If patches are issued in closer time units to the discovery of the vulnerability, we can state that open source software is more secure from a patching perspective. This is due to the likelihood of a hacker being able to exploit an unpatched vulnerability as decreasing as the time required to

issue a patch decreases. Conversely, the longer a particular vulnerability goes unpatched, the likelihood of a hacker exploiting such a vulnerability increases. Therefore, if patches are released relatively quickly, the more secure the software should be. This assumes that the patches are successfully applied immediately after release. Failure of software owners to apply patches is a topic for future research.

From a reliability/queuing perspective, the discovery of a vulnerability can be viewed as an arrival event [15, 26, 29]. So, the inter-arrival times of vulnerabilities indicate the frequency of vulnerabilities that are discovered in unit time. Building on our initial belief about the inherent security of open source, we posit that the mean inter-arrival time of vulnerabilities will be greater for open source software. In other words, we expect to see fewer bugs being discovered in open-source systems, in unit time, reflecting their increased robustness over proprietary software. This leads to our second hypothesis,

***Hypothesis 2: In unit time, there are fewer vulnerabilities in open source software compared to proprietary software.***

The motivational profiles of open source and proprietary software developers have been widely studied based on theories of volunteerism and labor economics [13], [18]. Open source software developers have been found to be more motivated to contribute and respond to the development process of open source software due to the apparent visibility of their contribution. This visibility provides an incentive to participate and consequentially brings in the desired recognition. We also anticipate that open source developers would be more adverse to negative perceptions of high-severity vulnerabilities than proprietary developers. Thus, we believe that the open source community will respond faster to high-severity vulnerabilities than the proprietary software developers. Hypothesis 3 follows:

***Hypothesis 3:*** *Open source software patches high-severity vulnerability faster than proprietary software.*

Borrowing again from the motivational profiles of open source software developers, we hypothesize that open source developers will respond faster to confidentiality, integrity, availability, and security-protection types of vulnerabilities. This research question was then split into the following four sub-hypotheses in order to examine individual relationships between type of vulnerability and patch response times.

***Hypothesis 4a:*** *Open source software developers patch confidentiality-type vulnerabilities faster than proprietary software developers.*

***Hypothesis 4b:*** *Open source software developers patch integrity-type vulnerabilities faster than proprietary software developers.*

***Hypothesis 4c:*** *Open source software developers patch availability-type vulnerabilities faster than proprietary software developers.*

***Hypothesis 4d:*** *Open source software developers patch security protection-type vulnerabilities faster than proprietary software developers.*

Hypothesis 5 is based on the exploit range of the vulnerabilities. High-severity vulnerabilities are usually remote-exploits by nature. Therefore, similar to hypothesis 3 we anticipate that open source developers, owing to their increased individual recognition and implicit rewards, act faster in developing and issuing patches for remote-exploit vulnerabilities. So, following on hypothesis 3, we have,

***Hypothesis 5:*** *Remote-exploit vulnerabilities are patched faster by open source developers than their proprietary counterparts.*

#### **4. Data & Methodology**

Data for our analysis has been obtained from the ICAT database [14] maintained by the security division of National Institute of Standards and Technology (NIST) and the Common Vulnerabilities and Exposures (CVE) list maintained by the Mitre Corporation [20]. The data set contains vulnerabilities/bugs discovered and patched/fixed between the period of January 2001 and October 2003. The October 2003 cutoff date was selected to control for Microsoft's adoption of a monthly patching cycle commencing November 2003. The data is classified according to vulnerability type, severity of potential damage, and their exploited range. These classifications are made by the originators of the data sources, rather than by the researchers of this study.

As mentioned earlier, the focus of our study is on operating system software. The list of operating systems appearing in the sample is provided in Table 1. Hence the dataset is classified as open source and proprietary, based on the underlying operating system software where the vulnerability has been discovered. The dates of discovery of the vulnerabilities and the dates when these bugs were fixed are recorded to obtain the response times of vendors to release a patch.

**<Insert Table 1 Approximately Here>**

Depending on the perceived consequence of a security breach due to a prevailing vulnerability, borrowing from the well-known Confidentiality, Integrity and Availability (CIA) framework [8], the data is classified into confidentiality, integrity or availability bugs. A fourth classification, namely security protection (S) is also used to qualify a more general consequence of a security breach. Thus, a vulnerability that has been discovered could be classified into multiple types (C, I, A, and/or S) depending upon the possible consequences of the same. A confidentiality vulnerability implies that the vulnerability enables an attacker to steal data directly from the system. An integrity vulnerability indicates that the data residing or passing through the system

can be modified or deleted. An availability vulnerability is one that could deny authorized users access to the system resource. A security protection vulnerability is defined as a situation where an unauthorized user could gain system privileges not allowed by the access control policies of the system [14]. The potential damage caused by the vulnerability is categorized into three types depending on their severity: High, Medium, and Low (H, M, and L). These classifications are contained within the database and were not determined by the authors. A “high” severity vulnerability is one that could allow an unauthorized user to gain a root or user account or a local user to take over control of a system, or if there is an associated CERT advisory. A “medium” severity vulnerability is one that is neither classified as “high” or “low”, and a “low” severity vulnerability is one where the information gained is not valuable in and of itself but could possibly be used for other exploits or is considered inconsequential for most organizations [14]. Additionally, the exploit range of these bugs is defined to be either remote or local, again determined by the database compilers, rather than the authors of this study. A remote exploit range can allow attacks to be launched across a network, whereas a local exploit range requires previous access to a system in order to directly attack it [14].

Approximately 600 vulnerabilities were identified and listed in the CVE during the 2001-2003 time period. For several of the vulnerabilities, we were unable to get information on the exact date of discovery of the vulnerability. Additionally, some of the vulnerabilities affected both open source as well as proprietary software, in which case, it was difficult to determine the date of discovery or the date when the vulnerability was fixed, for both. Also, there were some vulnerabilities that did not have information on the nature of the vulnerability, their type and/or their exploit range. We removed such data points and ended up with a list of 412 vulnerabilities. A bootstrap procedure was performed, explained in the following section, to impute those missing

values where we only needed to estimate the date of discovery of the vulnerability. There were 148 such data points. The resulting data set contains 560 vulnerabilities. Sources for the information on dates of vulnerability discovery and patch application typically included the CERIAS website (<http://www.cerias.purdue.edu>), ISS X-Force, Security Focus, CERT, Microsoft Security Bulletins, Bugtraq, KDE, Neohapsis, etc. The dates of discovery of the vulnerabilities are as reported by these sources when the vendor is notified. For information on the dates of vendor response, we have taken them to be the dates when the security advisories were made public by the vendor, unless when the actual dates of vendor response were available.

Given the difficulty in obtaining such confidential details related to information security, and the resources available to us, getting exact dates of occurrence of these events was not an easy task. However, we have been consistent in this regard, during our entire data collection process and hence still believe that our preliminary findings would be validated. Next, we describe the procedure that was used to impute the missing data points.

### **Estimating Missing values**

As mentioned earlier, a large number of vulnerabilities had missing values related to the date of discovery of the vulnerability. If these data points were deleted, our analysis would be possibly biased. In order to eliminate this bias, we estimated the missing values using a simple bootstrap procedure [11].

For simplicity, we initially built our full data set by drawing discovery dates randomly from our existing data set, with replacement. This inherently assumes that our full data set of patch times and inter-arrival times had a similar distribution to our existing data set. Then this random drawing was performed for 1000 iterations to obtain mean estimates. The overall patch times and inter-arrival times of vulnerabilities did not show much deviation from our original

result. Our data sources contained information on the vulnerability type, the date it was patched and also when the CVE was assigned to these vulnerabilities. Now, to impute the patch times and inter-arrival times, we incorporated the following constraints:

- 1) We drew the values randomly from an exponential distribution where the mean was equal to the mean values obtained from our existing data set.
- 2) An upper limit and lower limit for each value randomly drawn was determined as an additional constraint, from the information contained in our data sources.

We, therefore present the results from our preliminary analysis for our final data set.

## **5. Results and Discussion**

**<Insert Table 2 Approximately Here>**

Table 2 shows the mean time to patch for open-source and proprietary software, across different time periods. Goodness-of-fit tests were performed for the Normal, Lognormal, and Weibull distributions. Different parameter values of the Gamma distribution were tested and the exponential distribution was found to have the best fit. Note that we do not have adequate data points for the year 2000. Hence we can only make limited inferences here. However, if we look at years 2001-2003, we clearly see a smaller mean time to patch in the case of open-source systems. To compare the means in open-source and proprietary software statistically, we use the non-parametric Kruskal-Wallis test [11]. The parametric t-test cannot be used due to the violation of the normality assumption in our data set. The tests were performed for the data sets without the missing values as well as the complete data set and the results obtained were similar. Here we present the results for the complete data set containing the imputed values. We compute the chi-square values to determine the statistical significance of the difference between the means in open-source and proprietary software [11]. The last column in Table 2 shows the p-values for



comparing the distribution of the two samples. The results that are significant at 10% level are indicated in bold. We see that Hypothesis 1 is supported only in some cases. This finding does contradict the improvements in security management statements made by some open source proponents. We provide market share data [9] for both open source and proprietary operating system software in client applications, in Table 3, for comparison purposes. The data shown does not reflect any major trend reversals. Contrary to popular belief, proprietary operating system software market share only seems to grow larger. Do note that our dataset is restricted only to vulnerabilities in client applications and not server applications, where the move to open source software might be more pronounced.

**<Insert Table 3 Approximately Here>**

Figure 1 illustrates the exponential distribution of the counts of the patch times for both open source and propriety operating system software in our sample. The overall mean time to patch for open source software was 35.75 days, about 14% less than the mean times for proprietary software (41.22 days). Figure 2 displays the fit of the exponential distribution to the patch-times data for the two types of operating systems.

**<Insert Figure 1 Approximately Here>**  
**<Insert Figure 2 Approximately Here>**

Figure 3 provides the exponential distribution of the counts of the inter-arrival times of the vulnerabilities in the sample and similarly, Figure 4 displays the fit of the exponential distribution to the vulnerability inter-arrival data for the two sets of operating system software.

**<Insert Figure 3 Approximately Here>**  
**<Insert Figure 4 Approximately Here>**

The numbers in Table 4 show the difference in the mean inter-arrival times of vulnerabilities for the two software-types. There is not much difference in the mean inter-arrival

times and though proprietary software exhibits a lot more variation, the difference in the means is not statistically significant ( $p\text{-value} = 0.3625$ ) at the 10% level. Hence, at this point, we cannot conclude that open-source software exhibit greater robustness. Again, this is contrary to some widely-held beliefs in the open source community. We, therefore, reject Hypothesis 2.

**<Insert Table 4 Approximately Here>**

Table 5 shows the relative performance of the two software systems based on the type of severity. As we had hypothesized earlier, we find a significant difference in the performance for high-severity vulnerabilities. This finding implies that the individual recognition within the open source community appears to positively impact the reduction in high-severity vulnerabilities within the data set. Therefore, Hypothesis 3 is supported. The medium and low-severity vulnerability meanwhile do not show differences that are statistically significant. This could mean that, given time constraints, providers of both open source and proprietary operating system software, choose to focus on high-severity, rather than lesser vulnerabilities.

**<Insert Table 5 Approximately Here>**

The classification based on the type of the vulnerability (C, I, A or S) is shown below in Table 6. We do see a difference in the patch times of confidentiality-type vulnerabilities, but it is in favor of proprietary software systems. However, the difference is not statistically significant. Integrity and Availability-type vulnerabilities also do not show any significant difference. Thus, Hypotheses 4a, 4b and 4c are not supported. A more general type of vulnerability, namely the Security protection, meanwhile, does show a significant difference in the patch times, which is consistent with our results for Hypothesis 1. Hypothesis 4d is supported. Yet, looking at the means, it is difficult to ascertain if one type of software is better than the other. We conclude

that the results are mixed for the C, I, A, and S-type vulnerabilities, and that more research is indicated in this area.

**<Insert Table 6 Approximately Here>**

Consistent with Hypothesis 5, we find that open source software developers patch faster than the proprietary software developers in the case of remote-exploit vulnerabilities, but the difference is not statistically significant. Thus, Hypothesis 5 is not supported. The corresponding results are presented in Table 7. Again, this finding indicates that more research would be beneficial in this area.

**<Insert Table 7 Approximately Here>**

Figure 5 shows the moving averages for the patch times and the vulnerability inter-arrival times. Moving averages have been calculated to eliminate or minimize the fluctuations of the patch time values and inter-arrival times observed to recognize underlying trends in them. Moving averages for several event-intervals were computed and as a result, adequate smoothing has been done, considering 120 event-intervals to show the trajectory of the two curves. Since the two parameters were found to be exponentially distributed, an exponential moving average was also computed and Figure 6 shows the relative convergence of the two curves for the patch times and inter-arrival times. The graphs indicate neither operating system software dominating the other. No clear trends emerge and if any, proprietary software systems appear to be catching up with open source software. This once again demonstrates the need for holistic software security management.

**<Insert Figure 5 Approximately Here>**

**<Insert Figure 6 Approximately Here>**

Summarizing, the results provide some perspective into the ongoing debate of security between open source and proprietary software systems. The paper makes a contribution to this growing literature on software security and reliability. The analysis reveals yet again that the security issue is still an open question. Critics who argue in favor of the “more critical eyes” theory may do well to look at the observations again. While open source software developers have lower average numbers of certain vulnerabilities, the lack of statistical significance on several measures indicate a close race between the open source and proprietary communities, with the proprietary developers learning many lessons from and responding to the open source community. Though time-to-patch, or the quickness of response from the vendor, is not the only measure of software quality, in the minds of the user, it is definitely one of the proxies for software security. The faster the patch is available, the more confident users feel towards the software’s security. Again, research [25] does show that securing a system does not end with making a patch available. Most users are found to be less responsive (call it user-responsiveness) to apply the patch that is available to them. Yet, restricting our study to purely technical concerns, and to just the vulnerabilities found in operating system applications, the results are not conclusive.

As mentioned earlier, given the limited data available to us, we do not differentiate our data sources based on the different disclosure policies they follow. Incorporating these policy differences would ensure a more complete framework is in place. Going back to the argument of proprietary software having more discovered vulnerabilities (due to their push to market pressures) and therefore being less secure, our results show that this is not the case. Both kinds of software contain almost the same number of discovered vulnerabilities and further, the yearly data reveals no trends indicating any one holding a clear advantage over the other, from a security standpoint.

This study contains a number of limitations. There is concern that despite using comprehensive and public data sources, not all vulnerabilities discovered are reported. CERT follows a policy of allowing vendors the opportunity to formulate a patch to address the vulnerability before the vulnerability is publicly announced. CERT typically allows vendors 45 days to create and test patches but the possibility exists that this time frame could be shorter but most likely longer, which might introduce bias into our sample. Another possible source of bias is the shifting user interest in open source software, particularly in years 2000 and 2004 (although we removed the 2004 data points from the sample due to Microsoft's policy change). The changes in the number of users could impact the number of vulnerabilities discovered. Also, while not reflected in the time frame of the sample selected, changes in patch release policies by major vendors such as Microsoft could cause changes in the average patch inter-arrival times after the new policies are implemented [30].

## **6. Conclusion and Future Research**

Our study contributes to the literature on software security. The key motivation remains to carry the software reliability model forward. Quite a few data points had to be imputed, since the information was incomplete. We seek to further examine this data through an analytical framework from the perspective of security as reliability. The insights available from the software reliability literature will be applied accordingly. Another issue worth pursuing is that of staffing questions for software development. Some critics state that open source software has too many developers than is efficient, and thus wastes resources. Others purport that perhaps open source software has naturally evolved to rely on the relatively large numbers of programmers used to manage the software. We hope to eventually shed light on this issue as well.

Our analysis serves as a good starting point to compare open source and proprietary software systems from a security perspective. However, the study is not complete. Data classification based on different versions of the operating system software would provide insights on the level of vulnerability of the software with respect to the software life cycle. For example, the life cycles of different versions of operating system software is inherently across proprietary and open source systems and even across various vendors. This unfortunately makes a closer look at the number of vulnerabilities at various stages of the software releases more difficult but nonetheless very interesting. For example, proponents of proprietary software would argue that there would be more vulnerabilities only in the initial versions of their software and that the vulnerabilities would only decrease in the later versions. Investigating such issues would have interesting implications about the social environment in which software is developed. This would provide definite directions on the subsequent adoption of these software versions. Currently, our study focuses primarily on operating system vulnerabilities. We hope to obtain similar data on server applications as well. Finally, incorporating the information on vendor disclosure policies for our different data sources and also the user patterns in applying a patch once it is available, will help us build on the framework we are trying to establish through this study. How good a patch is, or whether frequent patching is cost-effective are still questions open to research. An interesting extension of our analysis would be to analyze the quality of a patch, in terms of the number of new vulnerabilities it introduces into the system. Though data might be difficult to obtain, an analytical birth-and-death model could capture the mechanics of the vulnerability-patch lifecycle adequately. Another interesting angle to explore is the role of product maturity in vulnerability reporting. Software, just like other products, experiences a maturity life cycle. Making the comparisons of

earlier in the paper but being able to control for differences in product maturity, could provide further insights into this interesting and very relevant problem.

## References

1. Arora, A.; Telang, R.; and Xu, H. Optimal Policy for Software Vulnerability Disclosure. Working paper, Carnegie Mellon University, 2004.
2. Arora, A.; Nandkumar, A.; Krishnan, R.; and Telang, R. Impact of Patches and Software Vulnerability Information on Frequency of Security Attacks – An Empirical Analysis. Working paper, Carnegie Mellon University, 2004.
3. Banker, R. D.; Datar, S. M.; Kemerer, C. F.; and Zweig, D. Software Errors and Software Maintenance Management. *Information Technology and Management* 3, 1-2, (2002), 25-41.
4. Beattie, S.; Arnold, S.; Cowan, C.; Wagle, P.; Wright, C.; and Shostack, A. Timing the Application of Security Patches for Optimal Uptime. In *Proceedings of LISA Sixteenth Systems Administration Conference*, 2002, pp. 101-110.
5. Boehm, B.W., Brown, J.R., and Lipow, M. (1976) Quantitative Evaluation of Software Quality, *Proceedings of the Second International Conference on Software Engineering*, 592-605.
6. Bulkeley, W. M. Can Linux Take Over the Desktop? *Wall Street Journal*, May 24, 2004.
7. Cavusoglu, H. and Raghunathan, S. Optimal Timing Decisions for Application of Security Patches. Working paper, Tulane University, 2004.
8. CERT. Securing Network Servers. Prepared by CERT, Retrieved August 8, 2004, from <http://www.cert.org/security-improvement/modules/m07.html>
9. CNET News, Prepared by CNET, Retrieved December, 2004, from <http://www.news.com.com>
10. Diffie, W. Is Open-Source Software Better for Security Than Proprietary Software? ZDNet, January 16, 2003, Retrieved February 19, 2005 from [http://news.zdnet.com/2100-9595\\_22-980938.html](http://news.zdnet.com/2100-9595_22-980938.html).
11. Greene, W. H. and Greene, W. H. *Econometric Analysis*. 5th edition, Prentice Hall, Upper Saddle River, NJ, 2002.
12. Hamm, S. Linuxinc. *BusinessWeek*, Jan. 31, 2005, 60-68.
13. Hann, I.; Roberts, J.; and Slaughter, S. Why Developers Participate in Open Source Software Projects: An Empirical Investigation. In *Proceedings of the Twenty-Fifth International Conference on Information Systems*, 2004, pp. 821-830.

14. ICAT Metabase, 2004, <http://icat.nist.gov/icat.cfm>
15. Jelinski, Z. and Moranda, P. B. Software Reliability Research. In W. Freiberger, (ed.) *Statistical Computer Performance Evaluation*. Academic Press, New York, 1972, pp. 465-484.
16. Johnson, J. P. Economics of Open Source Software. Working paper, 2001.
17. Kannan, K. and Telang, R. Market for Software Vulnerabilities? Think Again. *Management Science*, Forthcoming, 2005.
18. Lerner, J. and Tirole, J. The open source movement: Key research questions. *European Economic Review*, 45, (2001), 819-826.
19. Lerner, J. and Tirole, J. Some Simple Economics of Open Source. *The Journal of Industrial Economics*, 50, 2, (2002), 197-234.
20. MITRE. Common Vulnerabilities and Exposure. 2004, <http://www.cve.mitre.org>
21. MIZI Research, Linux Vs Windows: Which is more secure? 2004, Retrieved January 20, 2005 from <http://www.mizi.com/en/index/news-app/story.32>
22. Myers, G. *Software Reliability: Principles and Practices*. John Wiley and Sons, New York, NY, 1976.
23. The Open Source Definition, Open Source Initiative, Retrieved February 14, 2005 from <http://www.opensource.org/docs/definition.php>
24. Pescatore, J. Management Update: Mount a Solid Defense Against Worms and Viruses. Gartner intraWeb Report ID Number: G00123950, 2004, Retrieved December, 2004 from <http://www.itap.purdue.edu/itresources/gartner/index.html>
25. Rescorla, E. Security holes... who cares? 2002, Retrieved January 20, 2005 from <http://www.rtfm.com/upgrade.pdf>
26. Shooman, M. Operational Testing and Software Reliability Estimation During Program Development. In *Record of the 1973 IEEE Symposium on Computer Software Reliability*, IEEE, New York, 1973, pp. 51-57.
27. Silver, M. and Pescatore, J. Security Holes Increase Windows Client TCO. Gartner intraWeb Report ID Number: G00123511, 2004, Retrieved December, 2004 from <http://www.itap.purdue.edu/itresources/gartner/index.html>
28. Sliwa, C. App Tests for Win XPSP2 Burden Users: Most Delay Installing XP Security Update. *Computerworld*, August 20, 2004.



29. Sumita, U. and Shantikumar, G. A Software Reliability Model with Multiple-Error Introduction & Removal. *IEEE Transactions on Reliability*, 35, 4, (1986), 459-462.
30. Vijayan, J. Microsoft Release Four Patches Under New Monthly Schedule. *Computerworld*, 37, 46, (2003) 6.

Operating System	Classification	N
Microsoft Windows XP, 2000, NT, ME, 98, 95	Proprietary	116
Sun Solaris	Proprietary	31
Mac OS X	Proprietary	32
Unix	Proprietary	46
Linux & Variants	Open Source	256
IBM AIX	Open Source	23
SGI Irix	Proprietary <sup>1</sup>	41
Others	Proprietary	15

**Table 1: Operating Systems and their Classification in the Sample**

---

<sup>1</sup> Open source modules were added to the IRIX operating system, but only in May 2004. For the period of our study, the software was proprietary.

Year	Open Source Software			Proprietary Software			Non-parametric test p-values
	N	Mean (days)	Std. Dev.	N	Mean (days)	Std. Dev.	
2000	37	57.76	57.52	17	35.19	23.00	0.1709
2001	94	30.64	41.77	83	37.59	47.96	0.1705
2002	81	34.12	37.78	105	42.56	47.49	<b>0.0587</b>
2003	67	32.73	50.55	76	44.68	56.24	0.2736
All	279	35.75	46.59	281	41.22	49.35	<b>0.0539</b>

**Table 2: Mean-time-to-patch values for open-source and proprietary operating system software. The Kruskal-Wallis test p-values at the 10% significance level are shown in bold.**

Year	Market Share for OSS (in percentage)	Market Share for PS (in percentage)
2000	<1	94
2001	1	95
2002	2.4	95
2003	2.8	97
2004 (Estimates)	1	98

**Table 3: Market share data for open source and proprietary operating system software (Client Software Only) [9].**

Type of OS affected	Inter-arrival times of vulnerabilities (Kruskal Wallis test p-value = 0.3625)		
	N	Mean (days)	Std. Deviation
OSS	278	4.59	7.20
PS	280	4.90	15.91

**Table 4: Mean inter-arrival times of vulnerabilities for open-source and proprietary operating system software.**

Type of Severity	Open Source Software			Proprietary Software			Non-parametric test p-values
	N	Mean (Days)	Std. Deviation	N	Mean (Days)	Std. Deviation	
High	136	36.49	52.73	154	44.72	53.89	<b>0.0281</b>
Medium	134	34.99	40.67	117	37.07	43.82	0.6782
Low	09	38.78	29.29	10	38.20	33.58	1.0000

**Table 5: Patch times for the different severity levels of the vulnerabilities. The Kruskal-Wallis test p-values at the 10% significance level are shown in bold.**

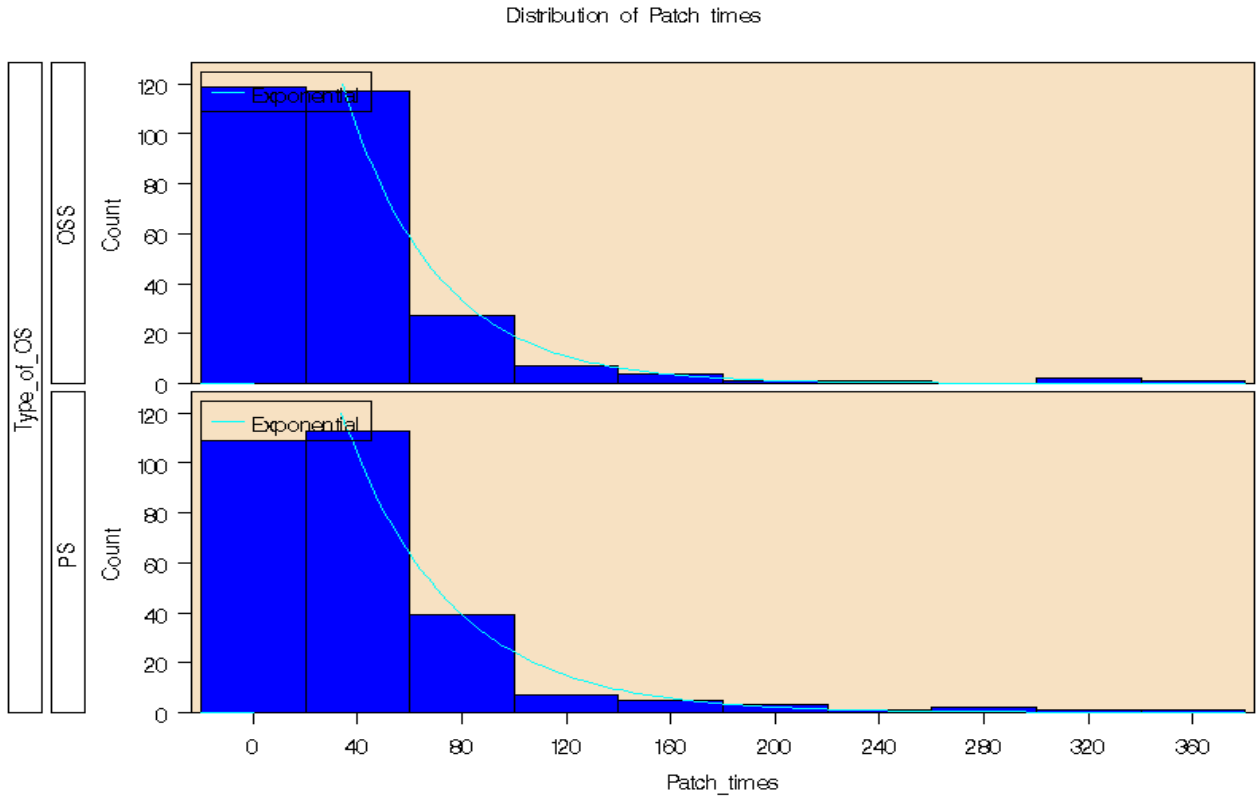
Type of Vulnerability	Open Source Software			Proprietary Software			Non-parametric test p-values
	N	Mean (Days)	Std. Deviation	N	Mean (Days)	Std. Deviation	
Confidentiality	38	40.39	45.83	26	34.00	36.34	0.6274
Integrity	56	29.16	28.70	37	29.76	29.51	0.8628
Availability	77	39.83	46.22	79	38.89	48.93	0.9958
Security Protection	165	33.27	48.80	189	43.37	52.42	<b>0.0079</b>

**Table 6: Patch times of the different types of vulnerabilities for the two operating system software. The Kruskal-Wallis test p-values at the 10% significance level are shown in bold.**

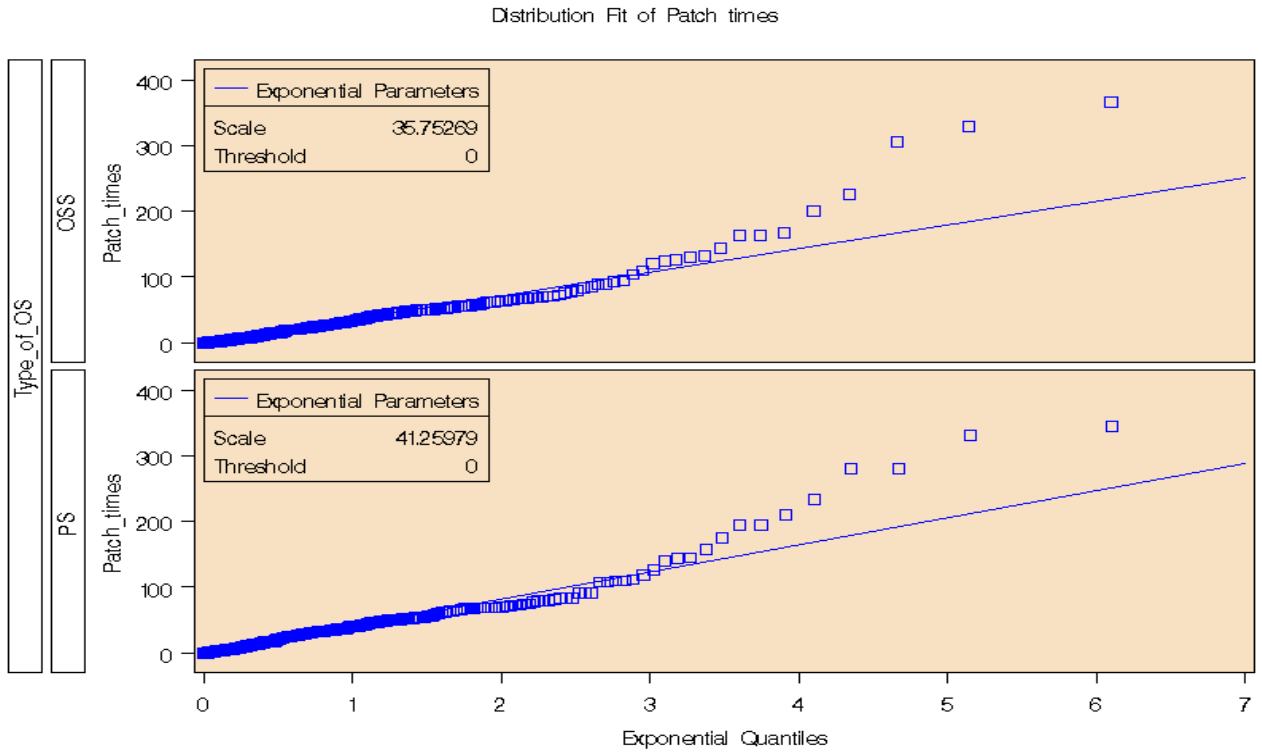
Exploit Range	Open Source Software			Proprietary Software			Non-parametric test p-values
	N	Mean (Days)	Std. Deviation	N	Mean (Days)	Std. Deviation	
Remote	114	37.11	40.77	148	41.90	45.84	<b>0.1478</b>
Local	175	35.67	49.77	134	39.81	51.14	<b>0.4166</b>

**Table 7: Patch times according to the exploit ranges for the two operating system software. The Kruskal-Wallis test p-values at the 10% significance level are shown in bold.**



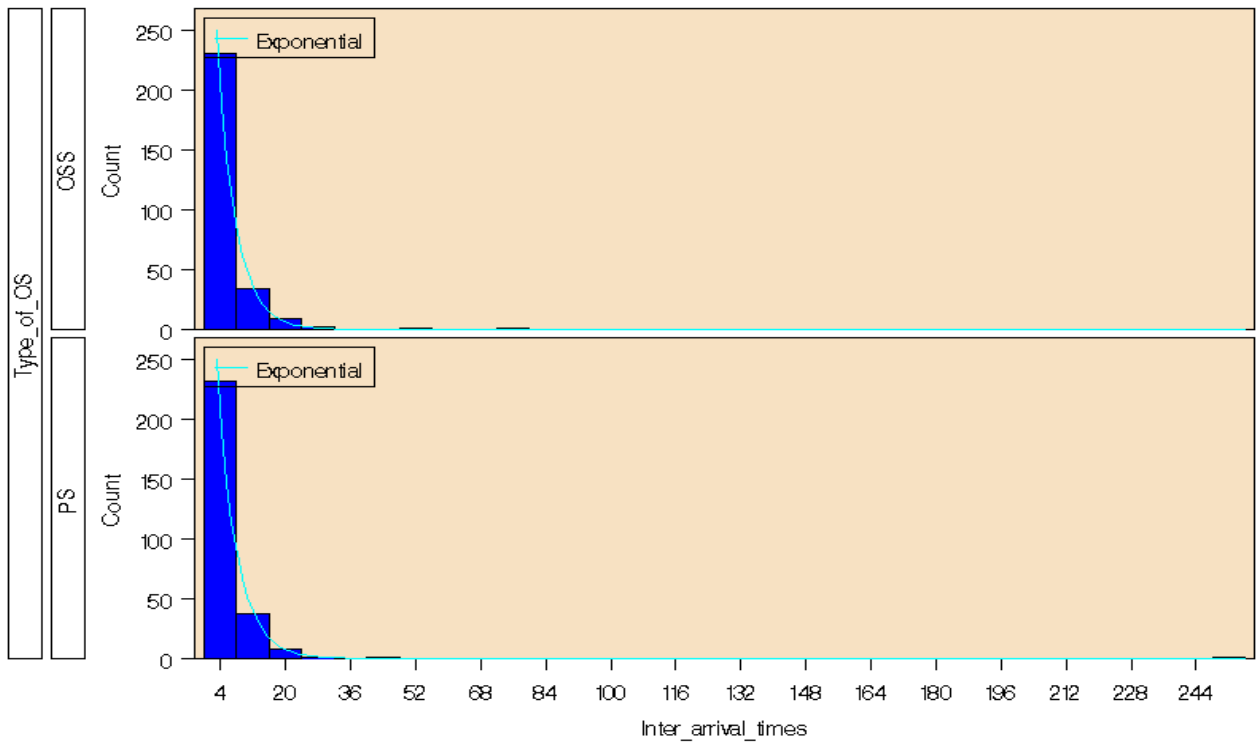


**Fig. 1: Overall mean time to patch for open-source and proprietary operating system software.**

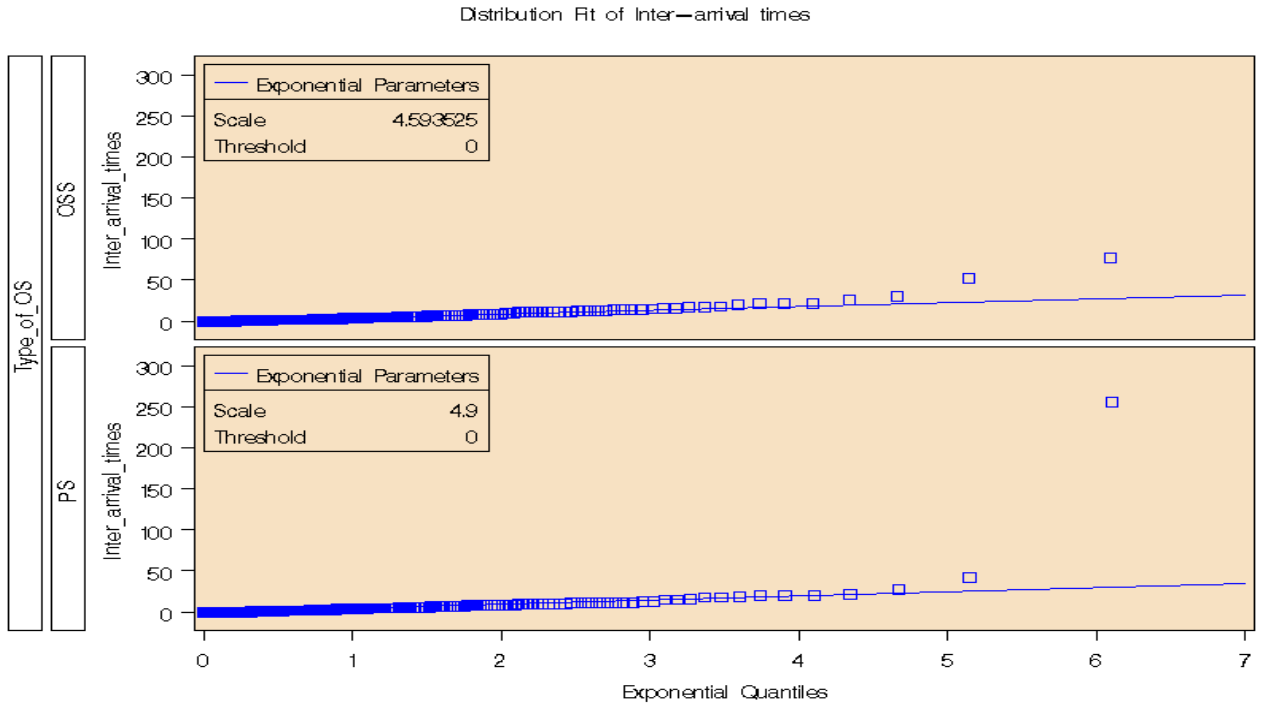


**Fig. 2: Fit of the exponential distribution for vulnerability patch-times.**

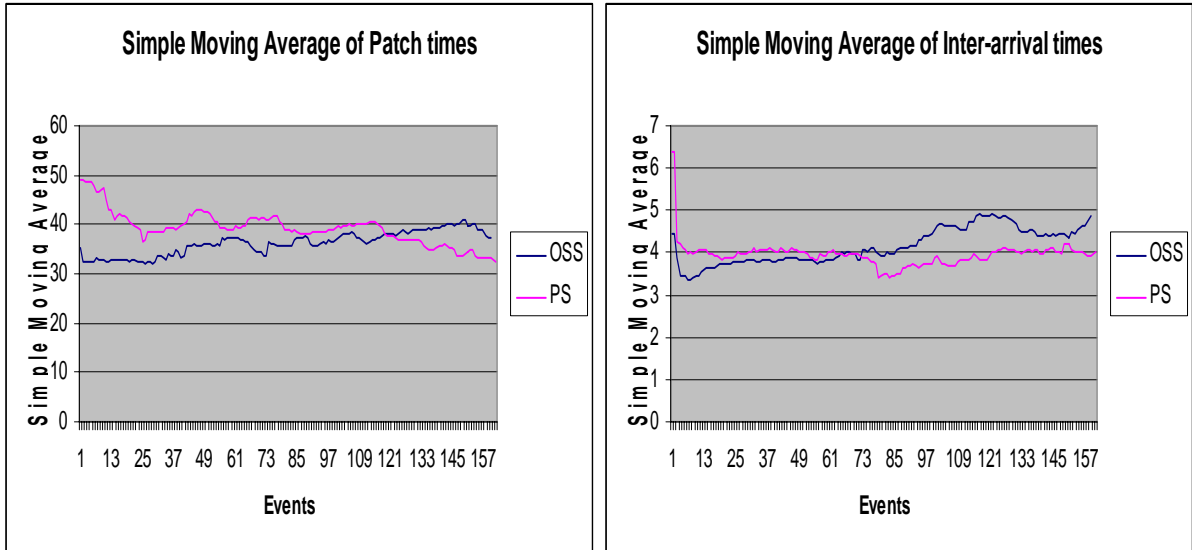
Distribution of Inter-arrival times



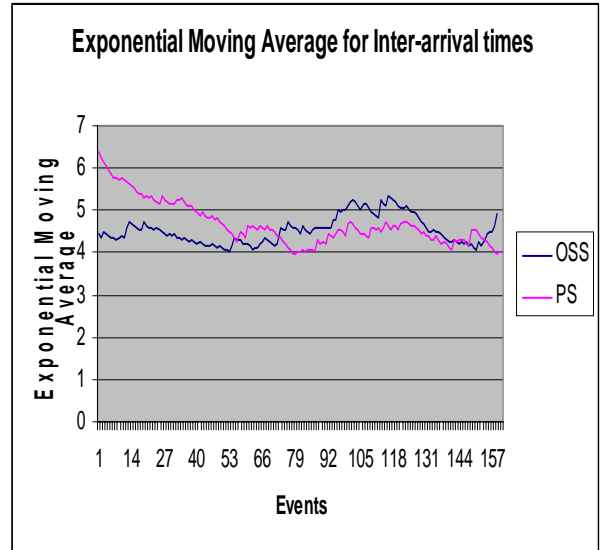
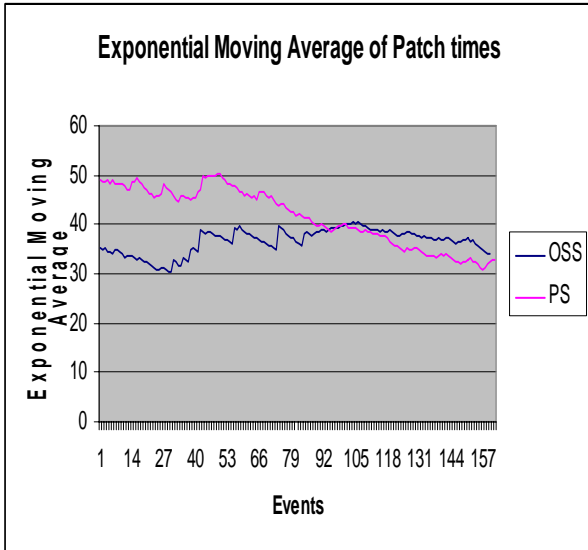
**Fig. 3: Inter-arrival times of vulnerabilities for open-source and proprietary operating system software.**



**Fig. 4: Fit of the exponential distribution for inter-arrival times of vulnerabilities.**



**Fig. 5: Simple Moving Average for patch-times and vulnerability inter-arrival times for 120-events interval.**



**Fig. 6: Exponential Moving Average for patch-times and vulnerability inter-arrival times for 120-events interval.**