December 2006

Comparative Investigation of Vulnerabilities in Open Source and Proprietary Software: An Exploratory Study

Nitin Walia
University of Wisconsin - Milwaukee

Balaji Rajagopalan
University of Wisconsin - Milwaukee

Hemant Jain
University of Wisconsin - Milwaukee

Comparative Investigation of Vulnerabilities in Open Source and Proprietary Software: An Exploratory Study

Nitin Walia
University of Wisconsin – Milwaukee
npwalia@uwm.edu

Balaji Rajagopalan
Oakland University
rajagopa@oakland.edu

Hemant K. Jain
University of Wisconsin – Milwaukee
jain@uwm.edu

ABSTRACT
The success of products like Apache and Linux has propelled increased awareness and adoption of open source software (OSS). Despite increased adoption of OSS products, questions about their security and reliability remain. Using four popular OSS and proprietary products as an initial sample, we examine the vulnerability patterns in OSS and proprietary products. Our analysis suggests that for both proprietary and open source products, in general, severe vulnerabilities are identified relatively late in the product's life and continue to emerge months after the software release. In particular, contrary to expectations, detection of vulnerabilities is no faster in open source (OS) than proprietary products. However, open source products had lower count of vulnerabilities at all levels of severity compared to proprietary products. We propose a conceptual framework to explain the variations in vulnerabilities between the OS and proprietary products. Our insights from the study have implications for research and practice.

KEYWORDS: Security, Reliability, Open Source Software, Vulnerability

INTRODUCTION

Open Source Software (OSS) products are emerging as preferred computing tools at least in some domains as evidenced by their usage in organizations. The wide adoption of some open source products by a number of high profile corporations (e.g., IBM, Apple, HP, Sun) and government agencies (e.g., Britain, Brazil, China, Germany, Japan, South Korea, South Africa, & Russia and the U.S. Department of Agriculture, the Federal Aviation Administration, the U.S. Department of Energy, the U.S. Air Force) indicates that open source products are in direct competition with their proprietary counterparts in at least some categories (Krishnamurthy 2002; von and von 2003; Scacchi 2001). The widespread adoption and acceptability of some open source products like Linux & Apache also suggests that these products are comparable to commercially available software in terms of product quality based on product features, reliability, and security.

Security, in particular, is a top concern for organizations as the exploitation of software defects by hackers and unauthorized intruders becomes increasingly common. Hence, firms concerned with increasing threats to data and application security give high consideration to security aspects in software adoption decisions. Vulnerability of software is defined as a defect that could lead to compromise of data and application resulting in loss of information, damage to the underlying system, and/or make system unavailable. A better understanding of how vulnerabilities manifest during the software product lifecycle can be instrumental in developing mechanisms to manage them. In addition, a comparative study of vulnerabilities in open source and proprietary products will also provide insights into the relative quality of products along the dimension of security and reliability. To this end, in this exploratory study, we address two related questions: (a) Are open source products more secure than proprietary software? (b) Do vulnerability patterns over the software product life cycle differ for open source and proprietary software?

Our analysis suggests that overall open source products have relatively less number of vulnerabilities than proprietary products but surprisingly not much difference exists in their vulnerability patterns. In addition, our findings indicate that for both OSS and proprietary products severe vulnerabilities are identified relatively late in product's life and continue to emerge months after a product's release. Although there is some evidence of a relationship between market share of the product and the rate and number of vulnerabilities further research is needed to confirm this hypothesis. The rest of the paper is organized as follows. In next section, we present a review of prior work, followed by details of data collection and research method.

Next, a discussion of results is presented. Finally, the paper concludes with a summary of findings and directions for further research.

THEORETICAL BACKGROUND

Both proprietary and open source software products have had their fair share of worms (Slammer, Blaster) and security vulnerabilities, causing financial loss for corporations, governments and even individual users. The underlying development model used to build a software product is one of the key factors in a product's ability to be more secure and reliable. Hence, the dissimilarities between the open source and proprietary software development model arguably lead to differences in security and reliability of software. The terms *cathedral* and *bazaar* are often associated with open source and proprietary development models respectively. Kuwabara (2000) discusses and distinguishes between these two models.

> *"The Cathedral and the Bazaar*, is a metaphorical reference to two fundamentally different styles of software engineering. On the one hand, common in commercial development, is the Cathedral model, characterized by centralized planning enforced from the top and implemented by specialized project teams around structured schedules. Efficiency is the motto of the Cathedral. It is a sober picture of rational organization under linear management, of a tireless watchmaker fitting gears and pins one by one as he has for years and years. On the other hand is the Bazaar model of the Linux project, with its decentralized development driven by the whims of volunteer hackers and little else. In contrast to the serene isolation of the cathedral from the outside, the bazaar is the clamor itself. Anyone is welcome - the more people, the louder the clamor, the better it is. It is a community by the people and for the people, a community for all to share and nurture. It also appears chaotic and unstructured, a community where no one alone is effectively in charge of the community. Not all are heard or noticed, and not all are bound to enjoy the excitement..."

Prior work has focused on how the two development models, open source and proprietary, influence design strategies that could potentially influence the level of reliability of the developed software. Several researchers have argued for the superiority of the open source development model which by making source code available promotes input and contribution from a large number of developers with a variety of perspectives and technical skills to spot and fix errors (Feller and Fitzgerald 2000 ; Hippel 2001; Scacchi 2004; Ulhoi 2004; Wu and Lin 2001). In a similar vein, Zhao and Elbaum (2003) found support for the Linus' law, "given enough eyeballs, all bugs are shallow" (Raymond 2000). Their analysis showed that in a majority of large open source projects, users found nearly 80% of bugs because of decentralized coordination between developers and users enabled by the availability of source code. Besides the availability of free source code, the OSS development model differentiates itself from proprietary model by two key principles (1) "Release early, release often, and listen to your customers" and (2) "Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix will be obvious to someone" (Raymond 2000).

In a study undertaken by Paulson et al. (2004), open source products (Linux, Apache, GNU complier) were compared with three proprietary wireless telecommunication products. Their results indicated that open source products had fewer defects than proprietary products. However, the validity of their findings has been questioned on the ground that the proprietary products selected in their study had a very different domain/architecture (wireless telecommunication) than OS products. Studies have also shown that open source vendors patch vulnerabilities faster than proprietary vendors (Arora et al. 2004; Ying et al. 2004). In a study conducted to examine the development model of Apache Web Server (Mockus et al. 2000), it was found that open source model exhibited very spontaneous responses to vulnerabilities as the required patches were released right away to users compared to proprietary products where patches were generally bundled with new releases. Another interesting point noted by Anderson (2001) is that users are more willing to provide feedback to the developers of a open source software (ex: Linux) than to proprietary firms like Microsoft.

Although there is some evidence touting the superiority of open source model, it is not without critics. For example, Lemos (2004) supporting the *cathedral* (proprietary) style of development claimed that the "many eyes" theory doesn't work as well in reality and that the profit oriented software companies, like Microsoft, can institute better security initiatives in response to vulnerabilities. Others have argued (e.g. Viega 2004) that Microsoft is more thorough when it comes to security because commercial firms are directly answerable to paying customers and averse to negative publicity. Moore (2004) noted that in 2004, Microsoft spent $6.8 billion on windows research and development and in comparison OSS products solely relied on volunteer developers for further development. Boulanger (2005) argued that having source code available for open source products is a multi edge sword, as it not only helps users and developers to respond against vulnerabilities expeditiously but also makes it easier for hackers to discover and exploit vulnerabilities.

To summarize the two perspectives, proponents of bazaar model argue that availability of source code, utilization of Internet for code sharing and development and parallel debugging provides a better route to manage and minimize security vulnerabilities, while its critics argue that the OSS development process/model is unorganized and leads to more bugs and

security vulnerabilities. Supporters of cathedral model believe that commercial interests and customer pressure (market forces) could only result in improved quality of software products. As the long-standing debate continues, empirical evidence comparing the vulnerability levels of software developed by the two models can help in understanding how the differences in the two models of development impact product quality.

In our study, we associate the OSS development model with the bazaar model and the proprietary software model with the cathedral model, implying the homogeneity of software development model for all products under each model. But we do acknowledge the fact that software products in each category (OSS and proprietary) make use of varying levels of bazaar and cathedral models. OSS development model could differ on multiple dimensions like paid and unpaid developers (Floss Final Report[1]; Golden 2004), number of developers (anywhere from a single developer to hundreds), bug reporting tools employed, version control mechanism, etc Similarly proprietary products could also adopt some practices of the bazaar model.. For the purposes of this study, given the differences in the core dimensions of these two development models (availability of source code, free software distribution or presence of volunteer developers), we classify a product's development model as either bazaar or cathedral.

In this exploratory study, we empirically compare vulnerability patterns of the products based on bazaar and cathedral software development models. By selecting four popular and visible software products from OSS and proprietary groups, we analyze in depth three types of vulnerabilities (High, Medium or Low) over several years. As a follow up, we extend this analysis to a larger sample of products.

DATA & MEASURES

Our data collection effort was divided into two parts: first for our pilot study (four products listed in Table 1) and then expanded to a larger data sample of sixty OSS and proprietary products. There are multiple data sources (CERT[2], NVD[3], OSVDB, securityfocus) available for capturing vulnerabilities data for software products. The motive behind our selection of securityfocus and OSVDB (Open Source Vulnerability Database) database for our study was to use a vendor-neutral, independent, and publicly available data source.

Securityfocus, with over 18 million page views a month and 2.5 million unique users annually, is one of the most comprehensive and trusted source of security information on the Internet. Symantec Corporation acquired SecurityFocus in the fall of 2002, but part of the purchase agreement was to keep SecurityFocus as an independent entity and not to be influenced by Symantec policies or corporate products. OSVDB, founded in August 2002 at the Black Hat and Defcon conferences, also aims to provide an independent and open source vulnerability database. For each product, vulnerability data was collected for all available years up to August 2004.

Development Model → Product Category ↓	**Proprietary**	**Open Source**
Web server	IIS 3.0 & 5.0	Apache 1.3.11 & 2.0.35
Server Operating System	Windows 2000	Linux2.4.1

Table 1: Sample distribution for software products

We initially examined two categories of software products: web server and server operating systems. The selection of the categories was based on the fact that breach of security in these categories of software products could result in severe consequences. The selection of these four products for our pilot study was also motivated by the fact that these products are leaders in market share in their category and hence, a prime target for attackers.

For classifying vulnerabilities into severity levels, we examined the standards followed by NVD, CERT, and Secunia & Symantec. NVD and CERT follow a three level scale (High, Medium, and Low) whereas Secunia and Symantec follow five-level scale (Very severe, severe, Moderate, Low, Very Low) for vulnerability levels. To reduce the complexity of the scale

[1] Free/Libre and Open Source Software: Survey and Study, Source: http://www.infonomics.nl/FLOSS/
[2] CERT: is a center of Internet security expertise a federally funded research and development center operated by Carnegie Mellon University
[3] NVD : National Vulnerability Database is a product of the NIST Computer-Security-Division and is sponsored by the Department of Homeland Security's National Cyber Security Division

without losing much information, we classified the vulnerabilities into three severity levels (see Table 2 for definitions) adopting the same standard followed by NVD.

Severity	Definition	Example (Source: symantec.com)	Threat Level
High	Typically used for remotely exploitable vulnerabilities, and can lead to system compromise. Key Points: 1. Remotely executable 2. Underlying system Compromised	"Microsoft Windows Shell is prone to remote code-execution vulnerability. This issue is due to a flaw in its handling of remote COM objects. Remote attackers may exploit this issue to execute arbitrary machine code in the context of the targeted user. This may facilitate the remote compromise of affected computers." *Bugtraq ID: 13132*	Wild: High Damage: High Distribution: High
Medium	Typically used for remotely exploitable Denial of Service vulnerabilities against services like FTP, HTTP, and SMTP, and for vulnerabilities, which allows system compromises but require user interaction. Key Points: 1. Needs user participation (clicking on a web site or opening a email or running an exe) 2. Compromises the Services running on the system	"Microsoft Exchange Server 2000 is vulnerable to a denial of service (DOS) attack in which a remote attacker can temporarily disrupt mail service by sending mail with malformed attributes directly to the server. As a result, the Store Service in Exchange 2000 uses all available CPU resources while processing the message." *Bugtraq ID: 4881*	Wild: Moderate Damage: Moderate Distribution: High
Low	The vulnerability does not typically yield valuable information or control over a system but instead gives the attacker knowledge that may help the attacker find and exploit other vulnerabilities. Key points: 1. Need local access to the machine to execute the worm/virus 2. Could be used to aid further severe attacks	"The Orinoco drivers for Linux kernels are susceptible to remote information-disclosure vulnerability. This issue is due to the driver sending un-initialized kernel memory in small network packets. Remote attackers may exploit this issue to access potentially sensitive kernel memory, aiding them in further attacks." *Bugtraq ID: 15085*	Wild: Low Damage: Low Distribution: Low
Wild: Measures the extent to which a virus is spreading among computer users. **Damage:** Measures the amount of damage that a given infection could inflict. **Distribution:** Measures how quickly a program spreads itself.			

Table 2. A Classification System for Vulnerabilities
(Source: NVD, CERT and www.symantec.com & www.secunia.com)

Two graduate students were briefed on the criteria for classifying vulnerabilities into the three categories. High inter-rater reliability (> 80%) of their classification indicated a high degree of consensus. We calculated rate of vulnerabilities for each product as a measure of total number of (detected) vulnerabilities divided by respective product's life in months (as on Aug-2004). The rate of vulnerability was calculated for all three level of vulnerability severity (Table 3).

Product	Age (in Months)	Rate of Vulnerability		
		High	Medium	Low
Apache	55	0.07	0.27	0.20
IIS	73	0.08	0.29	0.26
Linux2.4	43	0	0.37	0.97
Windows2000	53	0.43	1.72	1.85

Table 3. Rate of Vulnerability

ANALYSIS & RESULTS

In this section, we present results from our comparative investigation of the vulnerability patterns for the four OSS and proprietary products. Our insights are clustered into three categories: Vulnerability Distribution, Vulnerability Detection & Slowdown, and Vulnerability patterns. In addition, we include results from our expanded (on-going) study.

Vulnerability Distribution

Looking at vulnerability distribution for the four products examined (Figure 1), we can infer that open source products have less number of vulnerabilities than proprietary products and this pattern is consistent for all the three levels of vulnerabilities: high, medium, and low. This suggests that open source products examined in this study are more secure than their proprietary counterparts.

Looking at the rate of vulnerability (Table 3.), in the server category, Windows2000 had extremely elevated rate of vulnerabilities in all three categories high, medium, and low during its lifecycle compared to Linux 2.4. The difference is quite small for the two products (Apache & IIS) in web server category.

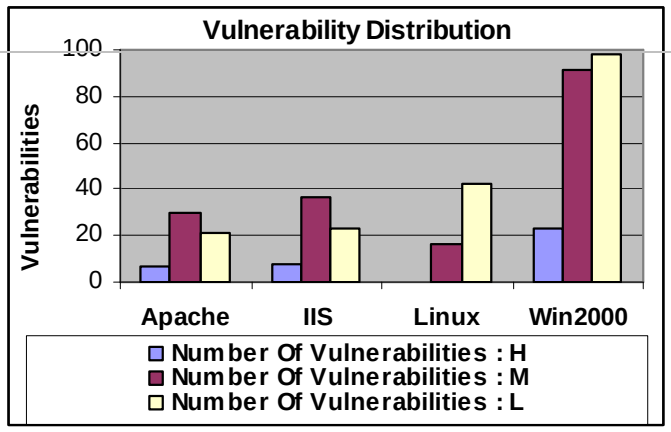

Figure 1: Vulnerability distribution for software products

When proprietary products are evaluated (in terms of vulnerabilities) against OS products, commercial organizations argue that their larger market share and high visibility make them a convenient target for hackers thereby causing their products to have more number of vulnerabilities than OSS products. But our study, by comparing equally visible and popular OSS products (Apache & Linux) with their proprietary counterpart, exposes the inadequacy of this rationale of being at an unfavorable position. It is highly doubtful if the differences in market and visibility can fully explain the relatively large differences in the vulnerability distribution and rate of vulnerabilities between the products.

Vulnerability Detection and Slowdown

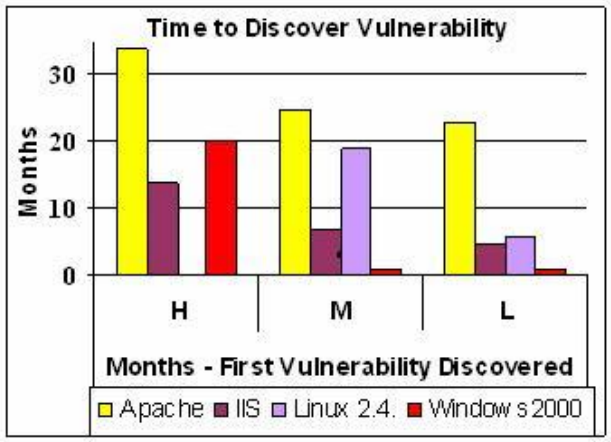

Figure 2a. Time to Discover (first) Vulnerability

Vulnerability Detection (Figure 2a): This metric is used to measure number of months it took for the first high (severe) vulnerability to be discovered for a product. The rationale behind the metric is to analyze the affect of availability of source code and eyeballs (Linus Law) on the detection rate of vulnerabilities.

For Linux, no high vulnerability has been discovered (as of Aug-2004) but medium and low vulnerability surfaced quite late in the product's lifecycle whereas for Windows2000 it took 20 months before the first high (severe) vulnerability was discovered whereas medium and low vulnerability surfaced within a month of its release. Performing a comparative analysis in web server category - Apache, even with a dominant market share, took more than twice the time for vulnerabilities to emerge compared to IIS. This is surprising and supports the skeptics of the 'Linus law' that question the value of large number of eyeballs contributing significantly to making the product bug-free.

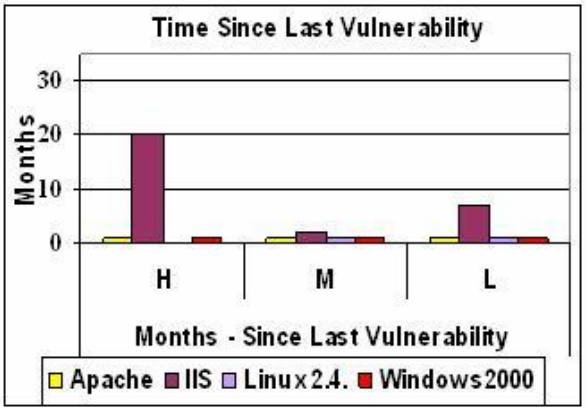

Figure 2b.Time Since Last Vulnerability

Vulnerability Slowdown (Figure 2b): This metric measures (in months) as to how long it takes for a product to be free of severe vulnerabilities. The rationale behind this metric is to analyze whether a software product has stabilized in terms of major security issues.

As of Aug-2004 (when data collection was done) vulnerabilities were still emerging in all the three severity categories for both windows2000 and Linux. In web server category, data suggests that vulnerabilities have stabilized for IIS, as it has been 20 months since last high (severe) vulnerability was discovered. In fact, even on medium and low vulnerabilities IIS performs much better than Apache.

To summarize, for Apache and Linux the vulnerabilities across all categories took more than twice as much time to surface compared to proprietary products. Again, interestingly, this pattern directly contradicts the notion that the open source model, by its nature should make security vulnerabilities easier to detect and fix. As to vulnerability slowdown, by and large the products under both models continue to have vulnerabilities even relatively late in the product life cycle.

Vulnerability Pattern

Category	Age - months	Takeoff Month		
		H	M	L
Apache	55	40	42	26
IIS	73	32	30	18
Linux 2.4.	43	0	38	6
Windows2000	53	48	48	24

Table 4. Vulnerability takeoff for software products

Vulnerability patterns of all products in this study seem to exhibit the well-known *takeoff* phenomenon (Golder and Tellis 1997; Agarwal and Bayus 2002), i.e., vulnerabilities are very low (if not zero) for several months during the early stage of a product's life and tend to show a sharp increase after time *t* as shown in Table 4. The take off month denotes the time t (in this case month) when highest numbers of vulnerabilities (difference of two consecutive months) were recorded for a product/category and the vulnerabilities stabilized or declined after the time t.

As evident from Table 4, the market leader in both categories: Windows2000 and Apache depict similar patterns. Windows2000 has late takeoff month for all three severity categories compared to Linux. Apache also depicts late takeoff month for all three-severity categories compared to IIS.

This late identification of severe vulnerabilities for both Windows2000 and Apache, even with high market share, can be problematic. It reveals that even in widely used products it takes long time to discover vulnerabilities and even longer to fix them. Thus, the vulnerable products may be running for long time before it is fixed.

Overall, our pilot study suggests that irrespective of the development model, severe vulnerabilities are identified relatively late in product's life cycle and continue to emerge months after the software release. The positive aspect for OSS products is that they had relatively low number of vulnerabilities at all levels of severity compared to proprietary products. However, the troubling aspect for proponents of the OSS model is the inability of the large number of eyes to translate into quicker detection of vulnerabilities.

Category ↓	**Number of products**	
	Proprietary	**OSS**
Web server	2	2
Web browser	2	4
Desktop OS	7	4
Server OS	6	7
Database	2	4
Office Suite	2	2
Communication tools	3	5
Others	2	6
Total	26	34

Table 5: Sample distribution for software products

Expanded Study

In this section we report the initial findings of our expanded study, which is still in progress at the time of this publication. When comparing OSS products with proprietary products, we have to take into account that software development process has matured and streamlined in large open source projects like Apache, Linux and Send mail (Moon and Sproull 2000; Koch and Schneider 2002; Koru and Jeff 2004) compared to medium and smaller open source projects. On the other hand it has also been argued that the development methodology and reliability of OSS is still on the same lines regardless of the size or popularity of OSS product (Feller and Fitzgerald 2000; Scacchi 2001; Aziz et al. 2005; Koch 2005).

To test this, we performed a comparative study of sixty OSS and proprietary products (Table 5) ranging in size (small, medium, and large products) and market share. The objective was to determine whether the results of our pilot study holds across different categories, size and market share of products.

As evident from Table 6, open source products (compared to proprietary products) in most categories have lower number of high and medium vulnerabilities. It should be noted that large sample size of open source products in 'Server OS' and 'Others' category of products may have contributed to higher vulnerability count in these categories. Another interesting observation that can be made from data in the table 6 is that products in popular categories (Desktop OS and server OS) have higher number of vulnerabilities compared to products in less popular categories (Database or Office suite). This may be due to nature of product e.g. operating systems are generally a target of attacker since it can provide access to system resources. It may also be due to lower volume of use.

Category ↓	**Rate of Vulnerability**					
	Proprietary			**OSS**		
	H	**M**	**L**	**H**	**M**	**L**
Web server	0.09	0.34	0.28	0.07	0.28	0.20
Web browser	0.54	0.89	1.19	0.13	0.31	0.59
Desktop OS	0.69	0.64	1.64	0.25	0.34	1.16
Server OS	0.24	0.43	0.51	1.42	0.23	0.78
Database	0.42	0.14	0.52	0.09	0.14	0.16
Office Suite	0.13	0.03	0.12	0.05	0.05	0.07
Communication	0.09	0.22	0.16	0.72	0.05	0.64
Others	0.10	0.47	0.78	0.68	0.32	1.30

Table 6. Rate of Vulnerability for software products

To forecast pattern of vulnerabilities for both OSS and proprietary products based on our data (60 products), we employed Holt's exponential smoothing method. Holt's two-parameter linear exponential smoothing is used to estimate separately the smoothed value of the time series as well as the average trend gain at each point in time. Two equations describing Holt's method are:

$S_t = A.y_t + (1 - A).(S_{t-1} + B_{t-1})$ (1)

$B_t = B.(S_t - S_{t-1}) + (1 - B)\, b_{t-1}$ (2)

S_t = smoothed value for time period t

B_t = smoothing the trend estimate for time period, t

Where A and B are smoothing constants between 0 and 1and t = 2, 3, 4

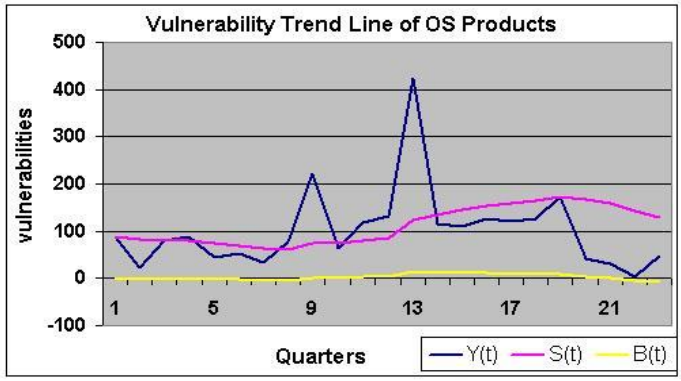

Figure 3a. Vulnerability Trend line of OSS Products

Even though open source products have relatively less number of vulnerabilities but in terms of vulnerability trend line, based on exponential smoothing method (Figure 3a, 3b), there is little difference between the open source and proprietary products. The trends (Figure 3a, 3b) validate our notion that for both OSS and proprietary products vulnerabilities start emerging in later part of product's life. It can be argued that the reason open source products exhibit this trend is because OSS products take longer time to be accepted by users and therefore the discovery of vulnerabilities is also delayed.

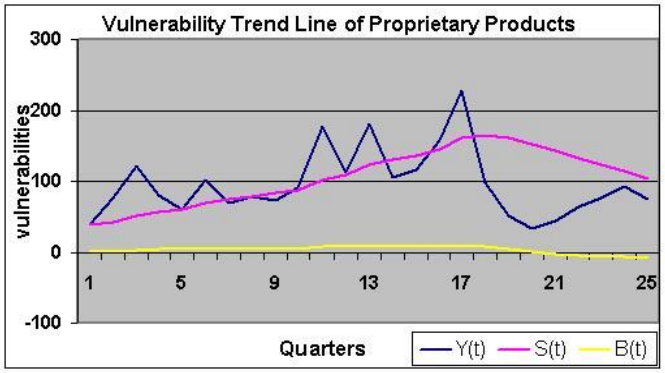

Figure 3b.Vulnerability Trend line of Proprietary Products

The preliminary analysis of data from our expanded study suggests that higher number of users and rising market share of a product does influence rate of detection of vulnerabilities. Two propositions that had some empirical support but need further investigation are:

1. Popularity of a software product has an affect on speed of vulnerability detection.
2. The development model of the product influences the rate and number of vulnerabilities.

The exact nature of impact of these factors needs additional analyses. Interestingly, trend line analysis for proprietary and open source software vulnerabilities shows that in both cases there is an increasing propensity of vulnerability with time.

CONCLUSION

This study examined the vulnerability patterns in software products based on bazaar and cathedral software development models. Our research provides empirical evidence that, in terms of severe vulnerabilities, open source products are more secure than proprietary products. On the other hand, contrary to our expectations, their general pattern of vulnerability detection was found to be quite similar. We also found evidence that higher market share and user base could lead to rapid vulnerability detection.

Our findings provide interesting implications for practice and research. Our results revealed that just being an open source product doesn't give an edge or guarantee early detection of vulnerabilities. Based on current scope of our work it is not clear whether our findings indicate failure of Linus law or it simply means that the growing number of OSS projects (118,620 projects on sourceforge.net[4]) has resulted in reduced ratio of beta-testers and co-developers available for each OSS project. Organizations and information technology project mangers may find these implications useful when adopting an OSS product or a developing strategy to initiate a new OSS project. For research, our study provides evidence that the software development models do affect the level of software quality but the affect varies on different dimensions used to measure software reliability and security. Another interesting research contribution that our study provides is the preliminary evidence

[4] Sourceforge.net -the largest repository of open source projects, http://sourceforge.net/ April30th, 2005

of a relationship between product's market share and vulnerability detection rate. This opens up possibilities for further extension of our work.

Overall, our study suggests that irrespective of OSS or proprietary model, products from both sides show similar propensity in terms of late and continued emergence of severe vulnerabilities. The next step of our study would be to integrate affect of various social and technological factors (influence of market share, motivation of developers and bias of intruders/Hackers, product type) with software development methodologies and practices to study vulnerability patterns. Our follow up study aims to build on this effort by incorporating additional metrics, updating the vulnerability data and conducting an in-depth empirical analysis.

REFERENCES

1. Agarwal, R. and Bayus, B. (2002) The market evolution and sales takeoff of product innovations, *Management Science*, 48, 8, 1024-1041.
2. Anderson, R. (2001) Why information security is hard - An economic perspective, *Proceedings of the 17th Annual Computer Security Applications Conference,* December 10 - 14, Washington, DC, 358-364.
3. Arora, A., Nandkumar, A., Krishnan, R., Telang, R. and Yang,Y. (2004), Impact of vulnerability disclosure and patch availability : An empirical analysis, *Proceedings of the 3rd Workshop on Economics and Information Security*, May 13-15,Minneapolis, MN.
4. Aziz, H. J., Gao, J., Maropoulos, P. and Cheung, W. M. (2005) Open standard, open source and peer-to-peer tools and methods for collaborative product development, *Computers in Industry Amsterdam* 56,3, 260-271.
5. Boulanger, A. (2005) Open-source versus proprietary software: Is one more reliable and secure than the other? , *IBM Systems Journal* 44, 2, 239-248.
6. d'Aspremont, C., Gabszewicz, J. J. and Thisse, J.F. (1979) On Hotelling's stability in competition, *Econometrica,* 47,5, 1145-1150
7. Feller, J. and Fitzgerald, B. (2000) A framework analysis of the open source software development paradigm, *Proceedings of the 21st International Conference in Information Systems,* April 16-19, Brisbane, Queensland, Australia, 58-69.
8. Feller, J. and Fitzgerald, B. (2002) A further investigation of open source software: Community, co-ordination, code quality and security issues, *Information Systems Journal,* 12,1 , 3-5
9. Ghosh, R. A. (1998) Interview with Linus Torvalds: What motivates free software developers? ,*First Monday: Peer-Reviewed J. Internet,* 3, 3.
10. Golder, P. and Tellis, G. (1997) Will It ever fly? Modeling the takeoff of really new consumer durables, *Marketing Science,* 16, 3, 256-270.
11. Golden,B (2004) *The source of open source*, Addison Wesley Professional, Boston, MA 02116
12. Hippel, E. V. (2001) Innovation by user communities: Learning from open-source software, *MIT Sloan Management Review* 42, 4, 82–86.
13. Koch, S. (2005) Evolution of open source software systems: A large-scale investigation, *Proceedings of the first International Conference on Open Source Systems,* July 11-15,Genova, (Eds.), 148-153
14. Koch, S. and Schneider, G. (2002) Effort, cooperation and coordination in an open source software project: Gnome, *Information Systems Journal* 12, 1, 27-42.
15. Koru, A. G. and Jeff, T. (2004) Defect handling in medium and large open source projects, *IEEE Software,* 21, 4,54-61
16. Krishnamurthy, S. (2002) Cave or community? An empirical examination of 100 mature open source projects, *First Monday: Peer-Reviewed J. Internet* ,7,6
17. Kuwabara,K (2000) Linux: A bazaar at the edge of chaos, *First Monday: Peer-Reviewed J. Internet* ,5,3.
18. Laat, P. (2004) Evolution of open source networks in industry, *Information Society New York,* 20, 4, 291-299.
19. Lemos, R. (2004) Too much trust in open source, *CNET News.* (http://news.zdnet.com/2100-3513_22-864256.html ; last accessed Dec 23 2005).

20. Mockus, A., Fielding, R. T. and Herbsleb, J. D. (2000) A case study of open source software development: The Apache server, *Proceedings of the 22nd International Conference on Software Engineering*, June 4-11, Limerick Ireland, 262-372
21. Moon, J. Y. and Sproull, L. (2000) Essence of distributed work: The case of Linux kernel, *First Monday: Peer-Reviewed J. Internet*,.5,11
22. Moore, P. (2004) The reason why you hear more about windows vulnerabilities is because we feel responsible to communicate to our users about it, *Indiatimes.com, (* http://infotech.indiatimes.com /articleshow/15168374.cms; last accessed Aug 12 2005)
23. Obasanjo, D. (2002) The myth of open source security revisited v2.0, *developer.com, (*http://www.developer.com/open/article.php/990711; last accessed Aug 12 2005)
24. Open source vulnerability database (OSVDB), (http://www.osvdb.org/; last accessed December 20, 2004)
25. Paulson, J. W., Succi, G. and Eberlein, A. (2004) An empirical study of open-source and closed-source software products, *IEEE Transactions on Software Engineering* 30, 4, 246-256
26. Rajagopalan, B. and Bayus, B.L. (2004) Exploring the open source software bazaar, working paper, Oakland University, MI
27. Raymond, E. S. (2000) The cathedral and the bazaar, (*http://www.catb.org/~esr/writings/cathedral-bazaar/* (version-3); last accessed December 25, 2005)
28. Scacchi, W. (2001) Software development practices in open software development communities: A comparative case study, working paper, University of California, Irvine
29. Scacchi, W. (2004) Free/Open source software development practices in the computer game community, working paper for *IEEE Software,* University of California, Irvine
30. SecurityFocus (http://www.securityfocus.com; last accessed Sep 8, 2005).
31. Secunia. "About secunia advisories" (http://secunia.com; last accessed Jan 30, 2006).
32. Symantec (http://securityresponse.symantec.com/; last accessed Jan 30, 2006).
33. Ulhoi, J. P. (2004) Open source development: A hybrid in innovation and management theory, *Decision London* 42, 9, 1095-1114.
34. Viega, J. (2004) Open source security: Still a myth, *O'Reilly* (http://www.oreilly.com/; last accessed Sep 18, 2005).
35. Von, H. E. and Krogh, G. V. (2003) Open source software and the private collective innovation model: Issues for organization science, *Organization Science* 14, 2, 209-223.
36. Von, K. G. and Hippel, E. V. (2003) Special issue on open source software, *Research Policy* 32, 7, 1149-1157.
37. Wu, M. and Lin, Y.D (2001) Open Source software development: An overview, *Computer* 34, 6, 3-38.
38. Ying, A. T., Ng, R. and Chu-Carroll, M. C. (2004) Predicting source code changes by mining change history, *IEEE Transactions on Software Engineering* 30, 9, 574-586
39. Zhao, L. and S. Elbaum (2003). Quality assurance under the open source development model, *The Journal of Systems and Software*, 66, 1, 65-75.