International Journal of Mathematical, Engineering and Management Sciences
Vol. 2, No. 4, 288–299, 2017
ISSN: 2455-7749

Modeling and Characterizing Software Vulnerabilities

Navneet Bhatt[#], Adarsh Anand[*]
Department of Operational Research,
University of Delhi, Delhi-110007 India
[#]navneetbhatt@live.com, [*]adarsh.anand86@gmail.com
[*]*Corresponding author*

V. S. S. Yadavalli
Department of Industrial and Systems Engineering,
University of Pretoria, Republic of South Africa
sarma.yadavalli@up.ac.za

Vijay Kumar
Amity University, Noida, Uttar Pradesh
vijay_parashar@yahoo.com

(Received December 13, 2016; Accepted March 24, 2017)

Abstract
With the association of software security assurance in the development of code based systems; software developers are relying on the Vulnerability discovery models to mitigate the breaches by estimating the total number of vulnerabilities, before they're exploited by the intruders. Vulnerability Discovery Models (VDMs) provide the quantitative classification of the flaws that exists in a software that will be discovered after a software is released. In this paper, we develop a vulnerability discovery model that accumulate the vulnerabilities due to the influence of previously discovered vulnerabilities. We further evaluate the proportion of previously discovered vulnerabilities along with the fraction additional vulnerabilities detected. The quantification methodology presented in this article has been accompanied with an empirical illustration on popular operating systems' vulnerability data.

Keywords: Vulnerability discovery modeling, Software security, Vulnerability categorization.

1. Introduction

Despite the progress made in computer programming and the respective software engineering practices, almost all the software program we often use in our day to day life still contain numerous bugs. However, post release of a software, some of the defects encountered are clearly more hazardous than the others. These flaws may affect the safety of the software system, henceforth termed as software vulnerabilities. A software vulnerability can be defined as "an instance of a mistake in the requirement, development, or implementation of a software such that its execution may violate the security policy" (Krsul, 1998). It has always been top most priority for a software engineer to discover the flaws, and also mitigate the risk by quickly distributing the patches. During the development of a software system, developer unintentionally inject some vulnerabilities in the source code repository, which are later noticed and resolved. All the potential vulnerabilities in a software are not discovered at the same time. Consequently, on the basis of the degree to which an individual vulnerability is discovered in the software, the developer can categorize the individual vulnerability based on a Common Vulnerability Scoring System (CVSS). The categorization procedure is suggested by the FIRST (www.first.org) as an effort to offer a vendor independent score system and reports a CVSS based vulnerability distribution to catalog various vulnerabilities based on their types. The National Vulnerability Database (NVD) maintained by National Institute of Standard Technology (NIST) provides the

score report and distribution of each vulnerability. A common vulnerability scoring system is an open framework for assessing the characteristics and severity of software vulnerabilities. Development of scoring system is important because they can assist in, investigating the intrinsic qualities of a vulnerability, and the penetration capabilities for breaching a soft spot. A CVSS suggests an approach to capture both quantitative and qualitative characteristic of a vulnerability to the software developer. The numeral score allows a developer to rank a vulnerability based on its severity and further helps the organization to assess the risk and prioritize the patching process.

The extent of an impact to the confidentiality, integrity and availability due to the exploitation of a vulnerability affects the security of the whole system. When a vulnerability is discovered, various metrics such as: base, temporal and environmental are calculated that captures several properties based on the intrinsic characteristic, change in time and process environment of a vulnerability. The vulnerability discovery refers to examine and locating the possible bug, flaw or weakness of the software system using various statistical tools and techniques. Post-release, both testers and users attempt to discover the vulnerable points in the software, and a certain proportion of users are attackers trying to breach the software. In this regard, software testers have to effectively monitor the vulnerability discovery process and evaluate the threat level corresponding to each vulnerability. Further, quantifying the vulnerabilities in a software system is similar to the detection of underlying faults in a software. Like the categorization of software faults help software engineers to check for reliability; in a similar fashion categorization of vulnerabilities helps the developer to counter-measure the threat due to any potential breach. These adequate measures are like, assigning resources for security testing, development and scheduling the safety patches.

In the context of vulnerability discovery, a flaw present in the software is a type of defect that can imply a high degree of risk to a software system. Due to its analogous behavior, various researchers have incorporated the concept of software reliability growth modeling in order to quantify the trends in the vulnerability discovery process. With proper modeling of software vulnerability process, the developer might be aware of the dormant flaws present in the software and can apply the adequate resources to inhibit the threats. As a brief review of related research, a substantial number of Vulnerability Discovery Models (VDMs) have been developed recently. These vulnerability models consider various aspects of vulnerability scenario ranging from exponential to s-shaped vulnerability discovery curves. The taxonomy of major VDMs, can be divided into two groups: time-based and effort based models. The time-based VDMs are parametric functions that can predict the total number of vulnerabilities discovered at a given time point. Most of the VDMs developed in the literature considers time as the governing factor. Since, the vulnerability repository uses calendar time intervals for the vulnerability disclosure. It was Anderson (2002) who first introduced the VDM, and the model developed was explicitly based on the SRGM outline. Anderson (2002) applied the Brady et al. (1999) model to capture the trend of vulnerability discovery. Yet, the empirical results suggest worst fitting of data. Needham (2002), Alhazmi and Malaiya (2005) argued that the difference in fitting the data for the Anderson Thermodynamic (AT) model is due to sociological factors like: decrease in vulnerability discovery rate can be described due to the losing attractiveness of software version over time rather than the difficulty in discovering vulnerabilities (Massacci and Nguyen, 2014). Later, in 2005, Rescorla (2005) attempted to classify the trends in the vulnerability discovery data by considering the linear and exponential model to predict the number of vulnerabilities. Alhazmi and Malaiya (2005) proposed a logistic, s-shaped model to capture the phenomenon considering

the impact of vulnerability detection rate during the three phases. The model also advocated that the security defect discovery differs distinctly from the normal software bugs. At first, users need to understand the target system in order to infiltrate, so they discover few vulnerabilities. With the increased attractiveness over the time, a significant number of users begin targeting the system resulting an amplified growth. Finally, the discovery process gets saturated due to a substantial switching of user to a newer technology. Furthermore, Alhazmi and Malaiya (2005) also focused on effort based modeling, where the discovery of vulnerabilities is based on efforts applied rather than time alone. In other work, Joh et al. (2008) considered that, in some situations the discovery growth curve could be asymmetric in nature and suggested to use Weibull distribution for vulnerability discovery due to the skewness present in its pdf. On the similar lines, Younis et al. (2011) inspected the applicability of Folded VDM. Moreover, Kapur et al. (2015) examined the logistic detection rate while discovering vulnerabilities. Recently, Anand and Bhatt (2016) proposed a hump-shaped model to capture the vulnerability exposure pattern due to the attractiveness of a software product in the market. They used a weighted criteria based ranking approach to judge the performance of proposed model with the existing VDMs. Besides the single version VDMs, multi-version VDMs have received less attention. A few authors have considered the discovery pattern in a multi-version software. Kim et al. (2007) examined the influence of shared source code in multi-versions vulnerability discovery process of a software. Lately, Anand et al. (2017) have formulated a multi-version VDM to quantify the number of vulnerabilities discovered. The model was based on the feature enhancement and shared code phenomenon that considered the vulnerability discovery rate attributed for the latest offering that is also accountable in previous version of the software due to code sharing.

The purpose of our article is that most of the past studies considered the discovery of a loop holes as a single vulnerability, while estimating the potential security defects. They did not elucidate exactly what number of additional flaws are being discovered, when a conceptual flaw is discovered as one vulnerability. Several researchers showed the existence of many vulnerabilities reported for the latest offering were also present in its preceding releases due to the existence of shared code. Moreover, a software can be a part of software product family and the vulnerabilities discovered in any version may indeed present in the other versions of the same product family. For example, a CVE entry, CVE-2011-5046, discovered in the Microsoft Windows family had affected different products namely Microsoft Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, and Windows 7. Hence, a single vulnerability discovered in any version may affect the discovery process of other versions and consequently might escalate the flaw count. Additionally, it has been observed that, one vulnerability count could exploit different types of vulnerabilities that exists in a software due to one conceptual flaw. As in the case of a CVE entry, CVE-2016-3375, a single flaw in the OLE Automation mechanism and VBScript scripting engine causes four different vulnerabilities, viz. Denial of Service, Execute Code, Overflow, and Memory corruption. Therefore, a single vulnerability can also be attributed for the discovery of additional vulnerabilities during the vulnerability discovery process (Windows 7, 2016).

In this paper, we first describe how vulnerability discovery model can be used to capture the above-mentioned phenomenon to model the vulnerability discovery process, then present an analytical derivations needed to compute the fraction of additional vulnerabilities discovered during the vulnerability discovery process. Next, in section 3, the predictability of the proposed model is illustrated and the segregation of the vulnerabilities lying dormant in the software is made into two classes. In section 4, the conclusion is given followed by the references.

International Journal of Mathematical, Engineering and Management Sciences
Vol. 2, No. 4, 288–299, 2017
ISSN: 2455-7749

2. Model Development

The vulnerability discovery model, suggested by the Alhazmi and Malaiya (2005) considered the impact of two factors that governed the rate of change of vulnerability discovered. The first factor deals with the number of installed base due to the rising popularity of a software and the second factor capture the decreasing phenomenon of the number of undetected vulnerabilities with time. The model advocated an s-shaped discovery curve due to its logistic behavior. The vulnerability discovery process modeled here considers the association of conceptual flaws being discovered with the flaws that triggers the detection of some supplementary flaws during the discovery process. In the model we assume that the discovery process is initiated by a certain number of vulnerabilities discovered during the testing process and the rate of change of vulnerability discovered at a given time comprises of two components that administrate the vulnerability discovery process; the first factor constitute the vulnerabilities discovered with a detection rate r and the second factor represents additional vulnerabilities which are detected due to the influence of the vulnerabilities discovered by time t. The differential equation describing the discovery process can be modeled as:

$$\frac{d\Omega(t)}{dt} = r\left(N - \Omega(t)\right) + s\frac{\Omega(t)}{N}\left(N - \Omega(t)\right) \tag{1}$$

where r represent the vulnerability detection rate and s resemble the rate that constitute the discovery of additional vulnerabilities influenced by the discovered vulnerabilities. $\left(N - \Omega(t)\right)$ denotes the untapped vulnerabilities remaining in the software and $\frac{\Omega(t)}{N}$ is the fraction of vulnerabilities discovered by the time t. We get a closed form solution after solving the equation (1) as:

$$\Omega(t) = N\left(\frac{1 - e^{-(r+s)t}}{1 + \frac{s}{r}e^{-(r+s)t}}\right) \tag{2}$$

It is interesting to note that, the behavior of our proposed model can be comparable to the model proposed by Kapur and Garg (1992) in the software reliability studies. Moreover, if we take $b = r + s$ and $\beta = \frac{s}{r}$, then, the proposed model reduces to the model given by Kapur et al. (2015). If we define, $f(t)$ as the probability of vulnerability discovered at time t and $F(t)$ as the fraction of vulnerabilities being discovered by time t, then the likelihood of vulnerability at a given time t or the equation (1) can be expressed as:

$$f(t) = \frac{dF(t)}{dt} = \left[r + sF(t)\right]\left[1 - F(t)\right] \tag{3}$$

$$f(t) = \frac{dF(t)}{dt} = r\left[1 - F(t)\right] + sF(t)\left[1 - F(t)\right] \tag{4}$$

International Journal of Mathematical, Engineering and Management Sciences
Vol. 2, No. 4, 288–299, 2017
ISSN: 2455-7749

The solution to the equation (4) yield the s-shaped cumulative vulnerability distribution and is given as:

$$F(t)=\left(\frac{1-e^{-(r+s)t}}{1+\frac{s}{r}e^{-(r+s)t}}\right) \tag{5}$$

Further, the differentiation of $F(t)$ gives the non-cumulative vulnerabilities distribution representing the stated discovery process as:

$$f(t)=\frac{r(r+s)^2e^{-(r+s)t}}{(r+se^{-(r+s)t})^2} \tag{6}$$

Hence, if N is the total number of vulnerabilities in the software, then the cumulative number of vulnerabilities discovered by time t, $\Omega(t)$ given in equation (2) can be rewritten as:

$$\Omega(t)=N\cdot F(t) \tag{7}$$

The noncumulative vulnerability distribution given in equation (6) can be illustrated in the Fig. 1. The curve achieve its peak, $f(T^*)$ or $F(T^*)$, at time T^* when

$$T^*=-\frac{1}{(r+s)}\ln\left(\frac{r}{s}\right) \tag{8}$$

$$F(T^*)=\frac{1}{2}-\frac{r}{2s} \tag{9}$$

and

$$f(T^*)=\frac{1}{4s}(r+s)^2 \tag{10}$$

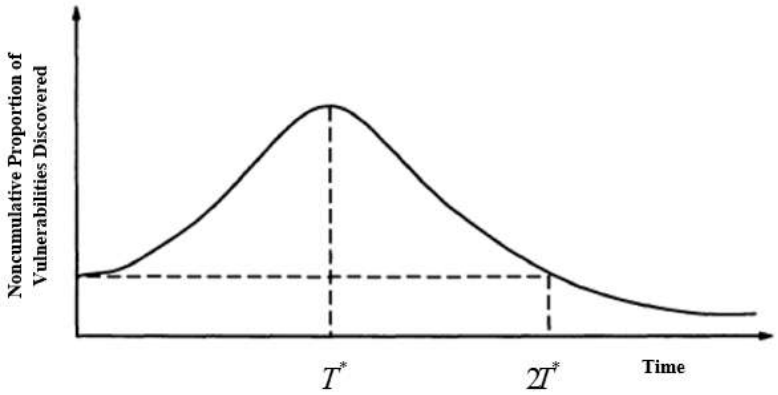

Fig. 1. Noncumulative vulnerabilities distribution

International Journal of Mathematical, Engineering and Management Sciences
Vol. 2, No. 4, 288–299, 2017
ISSN: 2455-7749

We can validate that the curve, as shown in the above Fig. 1, for the noncumulative vulnerability distribution is symmetric with respect to time. It can be shown that $f(t=0)=f(t=2T^*)=r$, that is, the proportion of noncumulative vulnerability discovered around the peak time T^* up to $2T^*$ confirming the symmetric behavior of the vulnerability discovery rate for the proposed s-shaped vulnerability discovery model. In fact, Younis et al. (2011) mentioned that, there is no assertive reasons advocating the rise and fall should be symmetric in the case of Alhazmi and Malayia Logistic model. However, Anand and Bhatt (2016) claimed that vulnerability discovery rate follows a hump-shaped curve showing the symmetric behavior of the discovery rate and hence can be backed using the equation (6).

It can be noted that, the term $r\left[1-F(t)\right]$ in equation (4) represents the proportion of vulnerabilities discovered by developer with a vulnerability discovery rate, r. Here, the vulnerability count resembled by this proportional are influenced by the advisory reports by software vendor. In contract, the second term $sF(t)\left[1-F(t)\right]$ represents the fraction of additional vulnerabilities discovered due to the influence of the previously disclosed vulnerabilities. The proportion of vulnerability discovery is depicted in Fig. 2, representing the vulnerabilities discovered due to the security bulletin by the software vendor and the additional vulnerabilities attributed because of the previously disclosed vulnerabilities.

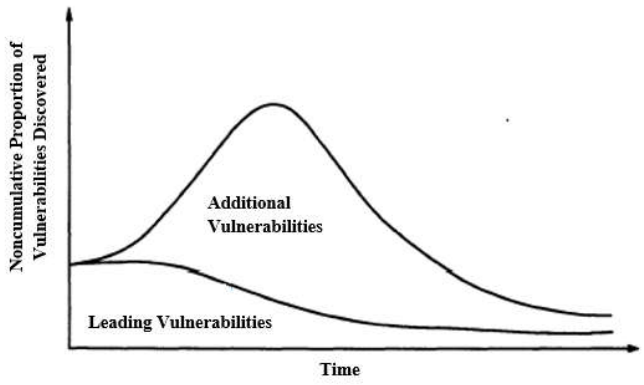

Fig. 2. Categorization of noncumulative vulnerabilities discovered

Moreover, in this work we try to capture the proportion of discovered vulnerabilities due to the influence of above stated two factors. As assumed, $F(t)$ is cumulative fraction of vulnerabilities discovered by time t. Therefore, as per the proposed model, we can anticipate that $F(t)$ inhibit two components. Viz., $F_1(t)$ the proportion of vulnerability discovered mentioned in the advisory reports by software vendor (leading vulnerability) and $F_2(t)$ corresponding to the additional vulnerabilities discovered. Because, $r\left[1-F(t)\right]$ gives the fraction of vulnerabilities discovered by developer at time t, then the total fraction of discovery, $F_1(t)$, between any two time periods, say t_0 and t_F, is given by:

International Journal of Mathematical, Engineering and Management Sciences
Vol. 2, No. 4, 288–299, 2017
ISSN: 2455-7749

$$F_1(t) = r\int_{t_0}^{t_F}\left[1 - F(t)\right]dt\,.$$

Because $F(t)$ is given by equation (5), Hence $F_1(t)$ can be inferred as

$$F_1(t) = r\int_{t_0}^{t_F}\left[1 - \frac{1-e^{-(r+s)t}}{1+\frac{s}{r}e^{-(r+s)t}}\right]dt\,.$$

This gives after integrating,

$$F_1(t) = \frac{r}{s}\ln\left[\frac{r+se^{-(r+s)t_0}}{r+se^{-(r+s)t_F}}\right].$$

Substitution of $t_0 = 0,\ t_F = t$ in the above equation yields:

$$F_1(t) = \frac{r}{s}\left[\frac{1+\frac{r}{s}}{1+\frac{r}{s}e^{-(r+s)t}}\right] \tag{11}$$

Therefore, the proportion of additional vulnerabilities discovered is given by $F_2(t) = 1 - F_1(t)$, that is

$$F_2(t) = \left[1 - \frac{r}{s}\left[\frac{1+\frac{r}{s}}{1+\frac{r}{s}e^{-(r+s)t}}\right]\right] \tag{12}$$

3. Data Analysis

This section addresses the practical relevance of the proposed VDM by predicting the future trends of the vulnerabilities. We assess the predictability of the proposed model by fitting the VDM to an observed sample and evaluate the goodness-of-fit criterion of the fitted model on the observed samples to predict the future behavior of the vulnerabilities. We apply the non-linear least square methodology to evaluate the estimation procedure on the security vulnerability data set of four different Operating Systems of two product family namely Microsoft Windows and Apple Macintosh (Mac Os X Server, 2016; Windows Xp, 2016; Windows Server, 2016; Mac Os X, 2016). The proposed VDM can only make sense if it closely fits the historical data and perfectly forecasts the future. Here, we compare the proposed model with the Alhazmi and Malaiya Logistic model. The parameter estimation, and comparison criteria of the two models are given in Table 1 and 2 respectively.

International Journal of Mathematical, Engineering and Management Sciences
Vol. 2, No. 4, 288–299, 2017
ISSN: 2455-7749

Parameter Estimates	Data Set							
	Windows XP		Windows Server 2008		Apple Mac OS		Apple Mac Server	
	Proposed VDM	AML	Proposed VDM	AML	Proposed VDM	AML	Proposed VDM	AML
N	942.15	876.51	566.13	550.26	1599.72	1293.52	654.85	649.36
r	0.0107	0.0004	0.0217	0.00139	0.0118	0.0002	0.0134	0.0008
s	0.3059	-	0.6612	-	0.2338	-	0.5254	-
c	-	0.0521	-	0.08554	-	0.03082	-	0.0806

Table 1. Model parameter estimation results

Goodness-of-fit Criterion	Data Set							
	Windows XP		Windows Server 2008		Apple Mac OS		Apple Mac Server	
	Proposed VDM	AML	Proposed VDM	AML	Proposed VDM	AML	Proposed VDM	AML
MSE	436.33	544.95	411.49	558.56	4247.90	5324.42	273.2647	304.82
Bias	-1.1448	-2.2445	-2.2445	-3.5331	-9.3646	-9.4827	-0.4087	-1.2427
Variation	19.3027	21.4873	21.4873	20.1217	59.8997	67.2204	15.2001	16.0082
RMSPE	19.3366	21.6042	21.6042	20.4295	60.6273	67.8860	15.2056	16.0564
R-Square	0.994	0.993	0.993	0.990	0.976	0.970	0.996	0.996

Table 2. Model comparison results

Table 2 reports the comparison criterions for the proposed VDM in each data set. Here, we observe that the proposed VDM fits the observed sample perfectly. All the comparison criterions are comparatively lesser than the AML model. Further, R-square shows a close fit which can be exhibited by the Fig. 3 to 6.

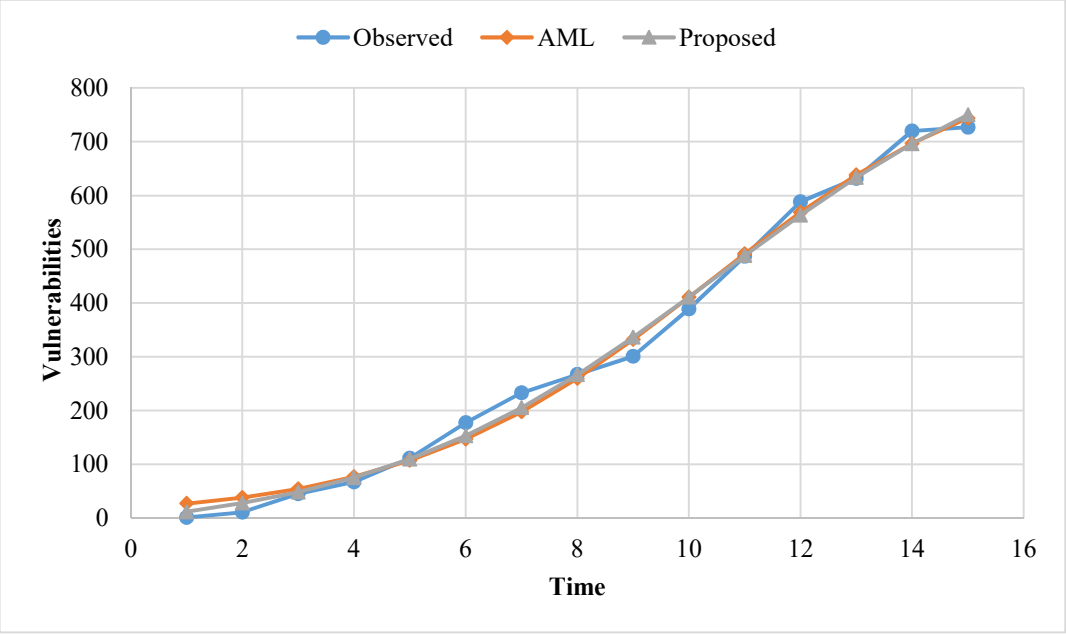

Fig. 3. Goodness of fit curve (Windows XP)

International Journal of Mathematical, Engineering and Management Sciences
Vol. 2, No. 4, 288–299, 2017
ISSN: 2455-7749

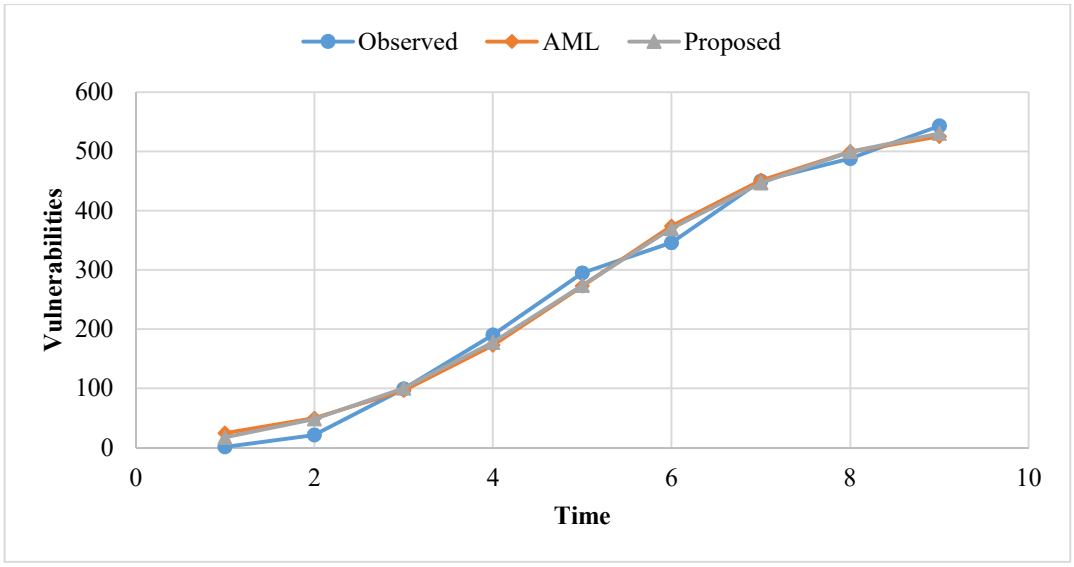

Fig. 4. Goodness of fit curve (Windows Server 2008)

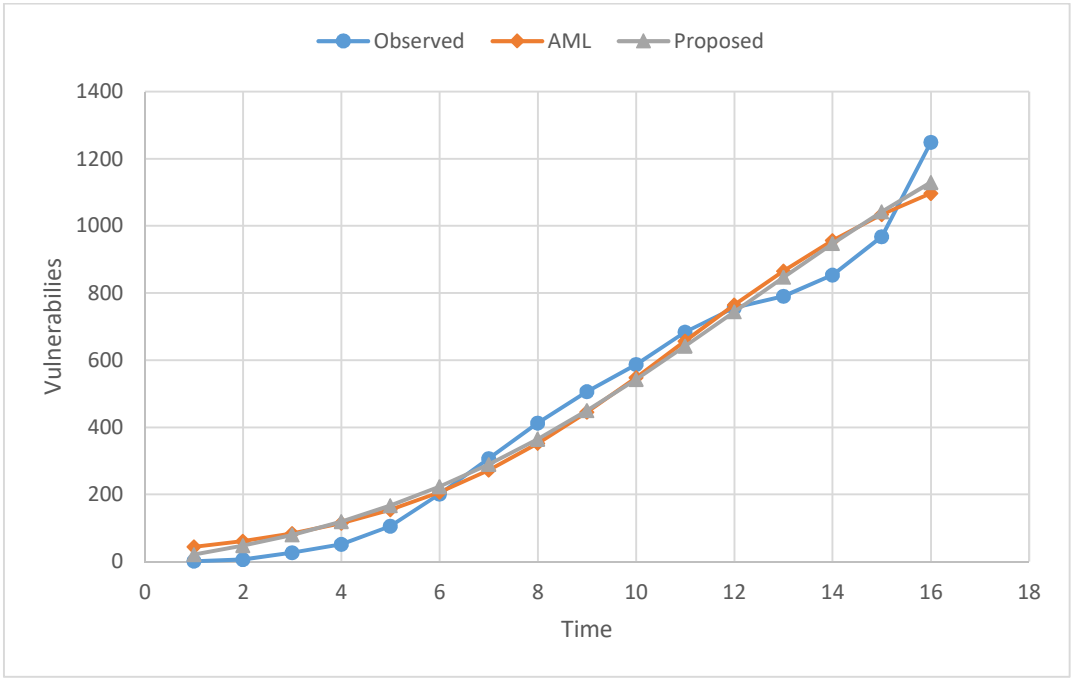

Fig. 5. Goodness of fit curve (Mac OS X)

International Journal of Mathematical, Engineering and Management Sciences
Vol. 2, No. 4, 288–299, 2017
ISSN: 2455-7749

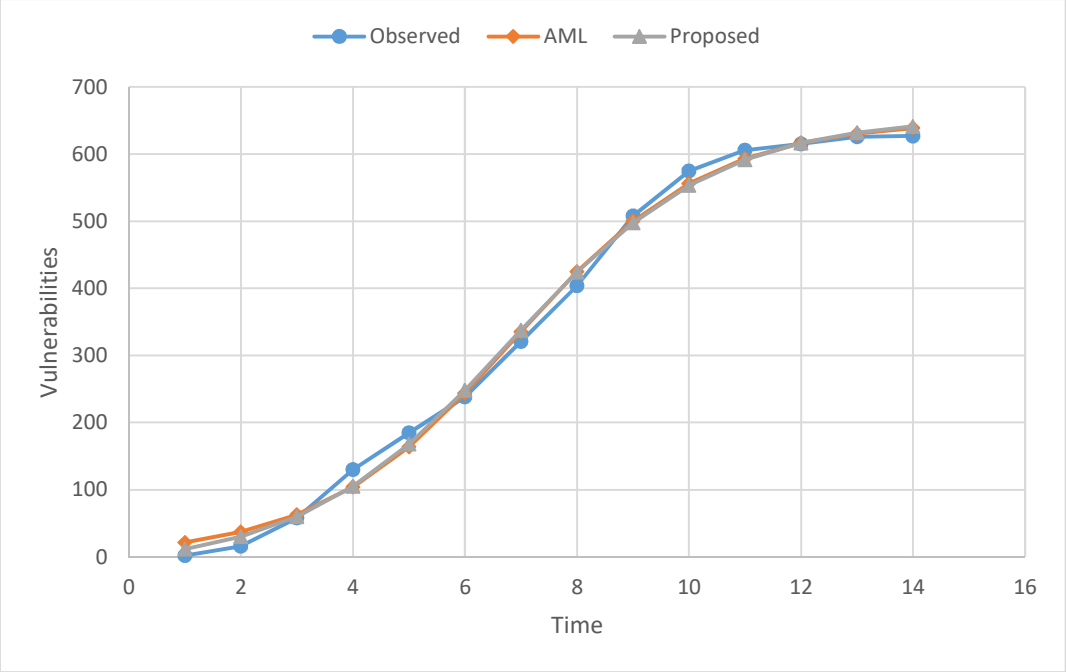

Fig. 6. Goodness of fit curve (Mac OS X Server)

Table 3 and 4 shows the observed and predicted values of vulnerabilities along with the proportion of vulnerabilities discovered due to the software vendor and additional vulnerabilities detected due to the previously discovered vulnerabilities.

	Windows XP				Windows Server 2008				
Time	Observed	Predicted	Proportion of Vulnerability Discovered	Additional Vulnerability Discovered	Time	Observed	Predicted	Proportion of Vulnerability Discovered	Additional Vulnerability Discovered
1	1	11.78544099	10.08419514	1.701245849	1	1	17.09748	12.12293163	4.974552103
2	11	27.48970665	20.0212921	7.468414558	2	21	48.07273	23.73655335	24.33617752
3	45	48.20109453	29.76338719	18.43770734	3	99	100.1238	34.46158183	65.66220353
4	67	75.14755573	39.25004731	35.89750841	4	190	177.4409	43.78291931	133.6579589
5	111	109.5932501	48.40729309	61.18595699	5	295	273.5476	51.1972757	222.3503454
6	177	152.6460128	57.14815769	95.49785515	6	346	369.7033	56.48520881	313.218085
7	233	204.9690998	65.37584336	139.5932565	7	450	447.1308	59.86431499	387.2665319
8	267	266.4350634	72.99035493	193.4447085	8	488	499.292	61.83724404	437.454768
9	301	335.8252583	79.89873734	255.926521	9	543	530.3464	62.9195045	467.4269449
10	389	410.7293112	86.02761696	324.7016943	-	-	-	-	-
11	487	487.77477	91.33512242	396.4396476	-	-	-	-	-
12	588	563.1864386	95.81847538	467.3679632	-	-	-	-	-
13	632	633.4978621	99.51451434	533.9833477	-	-	-	-	-
14	720	696.1454709	102.4929529	593.652518	-	-	-	-	-
15	727	749.7473076	104.844833	644.9024746	-	-	-	-	-

Table 3. Predicted and vulnerability proportions for Windows family

International Journal of Mathematical, Engineering and Management Sciences
Vol. 2, No. 4, 288–299, 2017
ISSN: 2455-7749

	Apple Mac OS				Apple Mac Server				
Time	Observed	Predicted	Proportion of Vulnerability Discovered	Additional Vulnerability Discovered	Time	Observed	Predicted	Proportion of Vulnerability Discovered	Additional Vulnerability Discovered
1	1	21.15213296	18.78959	2.362539	1	2	11.48807261	8.753016688	2.735055921
2	6	47.39395772	37.29999	10.09397	2	16	30.27034514	17.30569785	12.96464729
3	26	79.69979821	55.46534	24.23446	3	58	60.03688353	25.5359052	34.50097833
4	51	119.0950692	73.20792	45.88715	4	130	104.9601477	33.26740407	71.69274365
5	105	166.5830164	90.43794	76.14508	5	185	167.9904911	40.27323223	127.7172589
6	201	223.0343393	107.0544	115.98	6	238	247.9418513	46.31102659	201.6308247
7	307	289.0404272	122.9474	166.093	7	321	337.3250658	51.19611496	286.1289508
8	413	364.7427593	138.0028	226.7399	8	404	424.1818712	54.87976861	369.3021026
9	506	449.6670153	152.108	297.559	9	508	497.8158279	57.47291745	440.3429105
10	587	542.6055999	165.1604	377.4452	10	575	553.3546633	59.1939959	494.1606674
11	684	641.5977012	177.0761	464.5216	11	606	591.6630503	60.28602775	531.3770226
12	756	744.0425627	187.7985	556.2441	12	615	616.4856032	60.95750567	555.5280975
13	791	846.947189	197.304	649.6431	13	626	631.9245615	61.36199633	570.5625652
14	854	947.2648663	205.6056	741.6593	14	627	641.2838952	61.60254408	579.6813511
15	968	1042.246585	212.7504	829.4962	-	-	-	-	-
16	1249	1129.722358	218.815	910.9073	-	-	-	-	-

Table 4. Predicted and vulnerability proportions for Apple Macintosh family

4. Conclusion

VDMs have the potential to help the developer in allocating resources to predict the future trends of vulnerabilities, optimize the test effectiveness and scheduling the updates and patches for an exploitation free working of a software. The work presented here involved the empirical methodology to capture the involvement of additional vulnerabilities discovered during the discovery process. The quality and predictability of the proposed VDM are evaluated by the parametric function that evaluate the ability to forecast the future vulnerability as function of time. To validate the methodology, we assessed the proposed VDM on four major operating system of two distinguished product family. The results show a better insight about the vulnerability discovery process and revealed that it is better to use the proposed s-shaped model to estimate the vulnerabilities.

References

Alhazmi, O. H., & Malaiya, Y. K. (2005, November). Modeling the vulnerability discovery process. *In 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05) (pp. 1-10). IEEE.*

Anand, A., & Bhatt, N. (2016). Vulnerability discovery modeling and weighted criteria based ranking. *Journal of the Indian Society for Probability and Statistics*, *17*(1), 1-10.

Anand, A., Das, S., Aggrawal, D., & Klochkov, Y. (2017). Vulnerability discovery modelling for software with multi-versions. *In Advances in Reliability and System Engineering* (pp. 255-265). Springer International Publishing.

Anderson, R. (2002). *Security in open versus closed systems—the dance of Boltzmann, Coase and Moore. Technical report*, Cambridge University, England.

Brady, R. M., Anderson, R., & Ball, R. C. (1999). *Murphy's law, the fitness of evolving species, and the limits of software reliability* (No. 471). University of Cambridge, Computer Laboratory.

Joh, H., Kim, J., & Malaiya, Y. K. (2008, November). Vulnerability discovery modeling using Weibull distribution. *In 2008 19th International Symposium on Software Reliability Engineering (ISSRE) (pp. 299-300). IEEE*.

Kapur, P. K., & Garg, R. B. (1992). A software reliability growth model for an error-removal phenomenon. *Software Engineering Journal*, *7*(4), 291-294.

Kapur, P. K, Sachdeva, N, Khatri, S. K. (2015). Vulnerability discovery modeling. *International Conference on Quality, Reliability, Infocom Technology and Industrial Technology Management*, 34-54.

Kim, J., Malaiya, Y. K., & Ray, I. (2007, November). Vulnerability discovery in multi-version software systems. *In High Assurance Systems Engineering Symposium, 2007. HASE'07. 10th IEEE (pp. 141-148). IEEE*.

Krsul, I. V. (1998). *Software vulnerability analysis* (Doctoral dissertation, Purdue University).

Mac Os X Server. (2016). Vulnerability Statistics. http://www.cvedetails.com/product/2274/Apple-Mac-Os-X-Server.html?vendor_id=49. Accessed 6 February, 2016.

Mac Os X. (2016). Vulnerability statistics. http://www.cvedetails.com/product/156/Apple-Mac-Os-X.html?vendor_id=49. Accessed 6 February, 2016.

Massacci, F., & Nguyen, V. H. (2014). An empirical methodology to evaluate vulnerability discovery models. *IEEE Transactions on Software Engineering*, *40*(12), 1147-1162.

Needham, R. (2002). Security and open source. In open source software economics. Available at http://idei.fr/doc/conf/sic/papers 2002/needham.pdf.

Rescorla, E. (2005). Is finding security holes a good idea?. IEEE Security & Privacy, 3(1), 14-19.

Windows Xp. (2016). Vulnerability statistics. http://www.cvedetails.com/product/739/Microsoft-Windows-Xp.html?vendor_id=26. Accessed 6 February, 2016.

Windows 7 (2016). Vulnerability statistics. https://www.cvedetails.com/product/17153/Microsoft-Windows-7.html?vendor_id=26. Accessed 28 December, 2016.

Windows Server 2008. (2016). Vulnerability statistics. https://www.cvedetails.com/product/11366/Microsoft-Windows-Server-2008.html?vendor_id=26. Accessed 20 February, 2016.

Younis, A., Joh, H., & Malaiya, Y. (2011). Modeling learningless vulnerability discovery using a folded distribution. *In Proc. of SAM* (Vol. 11, pp. 617-623).