Vulnerability Discovery in Multi-Version Software Systems

Jinyoo Kim, Yashwant K. Malaiya, Indrakshi Ray
Computer Science Department
Colorado State University, Fort Collins, CO 80523
[jyk6457, malaiya, iray]@cs.colostate.edu

Abstract — The vulnerability discovery process for a program describes the rate at which the security vulnerabilities are discovered. Being able to predict the vulnerability discovery process allows developers to adequately plan for resource allocation needed to develop patches for them. It also enables the users to assess the security risks. Thus there is a need to develop a model of the discovery process that can predict the number of vulnerabilities that are likely to be discovered in a given time frame. Recent studies have produced vulnerability discovery process models that are suitable for a specific version of a software. However, these models may not accurately estimate the vulnerability discovery rates for a software when we consider successive versions. In this paper, we propose a new approach for quantitatively modeling the vulnerability discovery process, based on shared source code measurements among multi-version software systems. Such a modeling approach can be used for assessing security risk both before and after the release of a version. The applicability of the approach is examined using two open source software systems, viz., Apache HTTP Web server and Mysql DataBase Management System (DBMS). We have examined the relationship between shared code size and shared vulnerabilities between two successive versions. We observe that vulnerabilities continue to be discovered for an older version because part of its code is shared by the newer and more popular later version. Thus, even when the installed base of an older version has declined, vulnerabilities applicable to it are still discovered. Our results are validated using the source code and vulnerability data for two major versions of Apache HTTP Web server and two major versions of Mysql DBMS.

I. INTRODUCTION

Security vulnerabilities are of great concern because an unpatched vulnerability can potentially permit a security breach. Vulnerability is a software defect that can be exploited to cause a security breach. In 2006 alone National Vulnerability Database (NVD) [1] recorded 6600 new vulnerabilities, a 35% increase over the previous year. Predicting the number of vulnerabilities in a software system that will be discovered in a given time frame is important for several reasons. It will allow the developers to plan for allocation of resources needed to develop patches to address the vulnerability. A quick patch development process will reduce the exposure to zero-day exploits which exploit the time window between the discovery of a vulnerability and release of a patch to remedy it. It also gives a measure of the trustworthiness of the software. We need a quantitative model that describes the rate of vulnerability discovery in a given software. The vulnerability discovery is an important component of the security risk. Quantitative models can allow a developer to evaluate a version for suitability of release. They will allow users to assess the risk presented by the discovery of new vulnerabilities.

In software reliability engineering, a related discipline, a number of software reliability growth models (SRGMs) have been proposed that model the defect finding rate during testing. An SRGM is a mathematical expression that can be fitted to experimental data to project software reliability [2,3]. For example, Musa and Okumoto have suggested a Logarithmic Poisson SRGM [4]. Recently, researchers have started investigating how the vulnerability discovery process can be described using some Vulnerability Discovery Model (VDM). Examples of VDMs include the work by Anderson, Rescorla, Alhazmi and Malaiya [5,6,7,8,9]. Each model uses a different approach and has several parameters. However, these works have emerged only recently and many of their limitations have not yet been investigated. The models generally assume that each piece of software is an independent and well-defined product. This does not account for software evolution or maintenance. Because software typically does change with time, and generally inherits code from a previous version, the VDMs can exhibit some departure from real data.

Eick et al. [10] have extensively studied software evolution. Specifically, they showed how source code decay caused by evolution is dependent on time. Izurieta and Bieman [11] showed how source code evolution is dependent on time and user requirements. Ozment [12] considered software code sharing between major versions of software and showed how initial source code affected entire software reliability. However, he did not propose any software vulnerability discovery model that can be used for predicting software vulnerability. This research aims to fill this gap.

We chose major versions of Apache HTTP Web server and Mysql DBMS because Apache HTTP Web server (58% in [13]) and Mysql DBMS (29% in [14]) market shares are the highest among several Web servers and

1530-2059/07 $25.00 © 2007 IEEE
DOI 10.1109/HASE.2007.55

DBMSs. The source code for the versions of Apache HTTP Web server and Mysql DBMS are also freely available. We have compared source codes of 26 versions of Apache HTTP Web server (1.3.x) [15] and all the versions (75 versions) of Mysql DBMS (4.x and 5.x) [14] and collected their vulnerability data from NVD to find out relationship between evolution of the software and vulnerability detection.

The model proposed for the single version [7,8,9] uses the hypothesis that the market share influences the vulnerability discovery rates. As the market share increases, so does the rate of vulnerability discovery. When the software loses its market share, the rate of vulnerability discovery also decreases. There is greater motivation for finding vulnerabilities for products with a larger market share. The market share data has been used as a predictor for vulnerability discovery in [10].

We have investigated this model in the context of multi-version software. Our first observation is that when market share of a previous version is decreasing, we sometimes see an increase in the rate of vulnerability discovery. Our hypothesis is that the shared code between two versions causes this anomaly. The increase in the rate of vulnerability discovery in the previous version when it has lost its market share is due to vulnerabilities being discovered in the new version that can be attributed to the shared code. We evaluated our hypothesis on two open source systems, namely, Apache HTTP Web servers and Mysql DBMS. The results confirm our hypotheses. This leads us to a new model for predicting the vulnerabilities of multi-version software presented here.

The rest of the paper is organized as follows. Section 2 examines the software evolution and code sharing trends in specific software systems and illustrates the impact of software evolution on vulnerability discovery. Section 3 describes the software vulnerability discovery models that focus on single version software and presents a new multi-version vulnerability discovery model. Section 4 evaluates the new model using real software vulnerability data. Section 5 concludes the paper with pointers to directions for further research.

II. SOFTWARE EVOLUTION

Software evolution is the entire process that deals with gradually changing software. These changes can be for maintenance or modifications to incorporate functional enhancements. Ideally, software evolution should improve reliability and functionality. Realistically, that does not always happen. New vulnerabilities may get introduced along with new code in the process of evolution. The goal of this paper is to understand the relationship between evolution and vulnerability discovery.

A. Software Evolution and Code Decay

Software evolution trend refers to the change in software code size with time. As expected, this trend depends on the project team and whether the project is open-source or commercial. Mockus et al. [16] have identified the environmental factors leading to software evolution and its development. Godfry and Tu [17] suggest that software evolution trend depends on software development participants and the project requirements. In this section, we will examine stable development projects that have gone through a number of versions, to see how software vulnerability discovery is impacted. Apache HTTP Web server and Mysql DBMS both have a several year history and are thus good examples for relating software evolution and vulnerability discovery.

	Ver 1.3.0	Ver 1.3.37	Ver 4.0.0	Ver 5.0.0
Release Date	6-5-1998	7-26-2006	10-12-2001	12-23-2003
ANSI C	92.87	92.09	62.86	42.78
Sh	5.66	6.19	4.27	2.89
Perl	1.42	1.39	6.04	2.61
CPP	0.11	0.07	20.41	42.78

Table 1. Apache and Mysql Source Code Pattern

Since they are both open-source projects, the source codes for successive versions are available. We analyzed the source code patterns of Apache HTTP Web server and DBMS using SLOCcount [18]. The results are shown in Table 1. The first two columns are for versions of HTTP Web server and the last two columns are for Mysql. The major fractions of the source code of Apache HTTP Web server and Mysql DBMS are .c and .sh files. We ignored the source code that was made for CGI scripts. We used a comment-stripping program to extract the original source code. This procedure was performed on all versions of Apache HTTP Web Server and Mysql DBMS. To get the shared code of Apache HTTP Web server and Mysql DBMS, we used Diff and Line counter tools that are installed in Unix and Linux. Comparisons were performed for 26 versions of Apache HTTP Web server (1.3.x) and 27 versions of Mysql DBMS (4.0.x).

The software evolution and code decay relative to the initial version are presented in Figures 1 and 2 for Apache and Mysql respectively. The major initial version of Apache was released in 1998 and that of Mysql was released in 2001. The software growth for both the systems now shows saturation. Apache HTTP Web server has had a larger percentage of software source code modification (43%) than Mysql DBMS (31%). One reason for this variation may be because of the different degree of changes in the requirements. Since the DBMS is a much more well-defined software, its development can be expected to be more stable than

a HTTP Web server. The code size for both systems has grown logarithmically. There were minor changes to the functional requirements but several patches of security were applied to the later versions in both cases. Thus, the evolution was determined more by reliability rather than functional requirements.

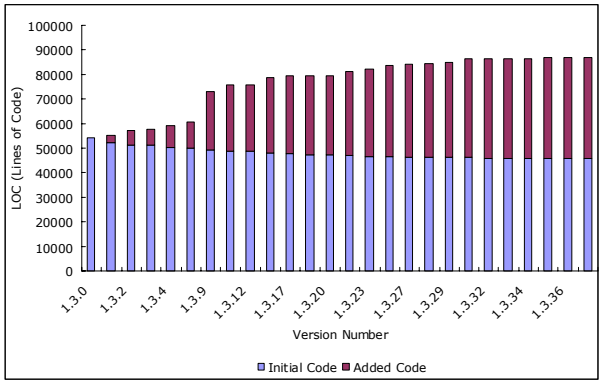

Figure 1. Apache HTTP Web server version 1.3.x evolution

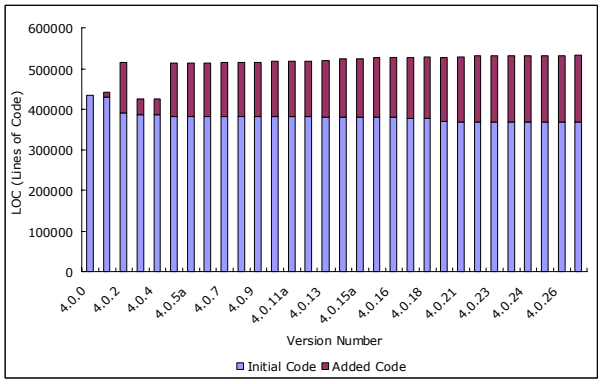

Figure 2. Mysql DBMS version 4.0.x evolution

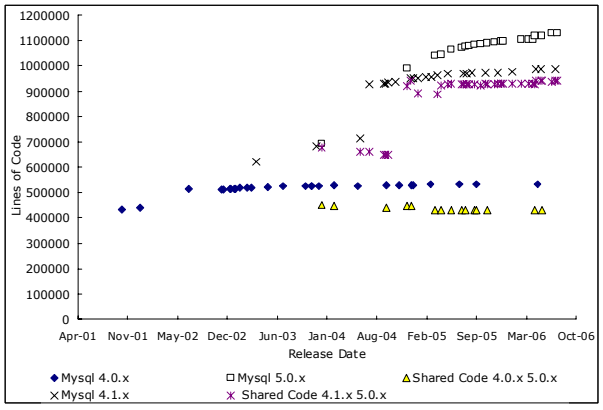

Figure 3. Mysql evolution and shared code for 4.0x, 4.1x and 5.x

Figure 3 shows code evolution of versions 4.0x, 4.1x and 5.0x of Mysql, including the code shared between successive versions. Each point corresponds to a specific subversion, indicated by specific values of x. Note that, the Mysql code evolution shows saturation for the three versions. It also shows that code shared between 4.0x and 4.1x as well as between 4.1x and 5.0x have been quite stable. There is no strong relation between software evolution and its version number. For obtaining the relationship between code sharing and vulnerability discovery, we need to compare successive versions of software instead of comparing between major versions. This is because evolution takes place with respect to the previous version. The code sharing between successive versions in Mysql also show saturation. In the next section, we describe the relationship between code sharing and vulnerability discovery.

B. Analysis of Vulnerability and Software Evolution

The software vulnerability trend is related to software evolution. Sometimes a vulnerability is found right after the release of the next version. Before we explain these trends, we discuss how to obtain vulnerability data. The apache HTTP Web server and Mysql DBMS vulnerability data was compiled using the following process:

1. For Apache HTTP Web server, we collected vulnerability discovery data for 8 years, from 5th June 1998 to 30th December 2006, by merging data from the NVD. For Mysql DBMS, we gathered vulnerability for six years after the release date, between 12th October 2001 and 30th November 2006. NVD is a public vulnerability database, which follows the CVE (Common Vulnerability and Exposure) vulnerability categorization developed by MITRE [19]. Using the CVE standard for vulnerability categorization ensures uniform treatment of vulnerabilities.
2. We collected details from the database resources linked with NVD to identify vulnerabilities associated with specific versions of Apache HTTP Web server and Mysql DBMS. NVD data itself usually does not indicate specific versions of the program.
3. The collected data was organized to compile vulnerabilities for specific versions of software. This process was needed to verify the vulnerability data set.

Ozment [20] has examined OpenBSD, an operating system software. Extremely few vulnerabilities found in Apache HTTP Web server and Mysql are operating system specific, thus we did not treat them separately.

We next investigated the relationship between vulnerabilities and software evolution. Figure 4 shows

the Apache HTTP Web server software evolution and its vulnerability discovery. The evolution and vulnerability discovery trends show a saturation phase. However, the plot for software vulnerability is growing slower than the software evolution model. Figure 5 shows the Mysql DBMS vulnerability discovery and software evolution trends.

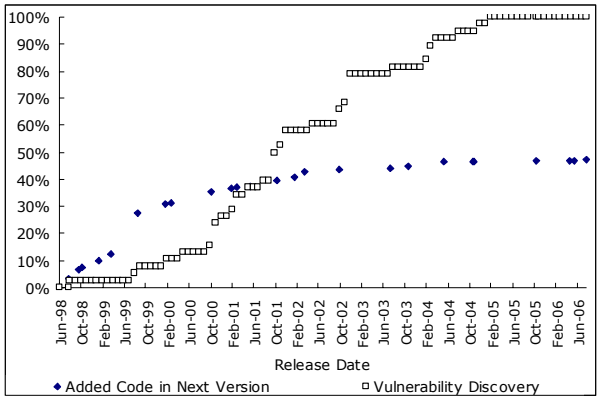

Figure 4. Vulnerabilities Discovered and Code Evolution in Apache HTTP Web server

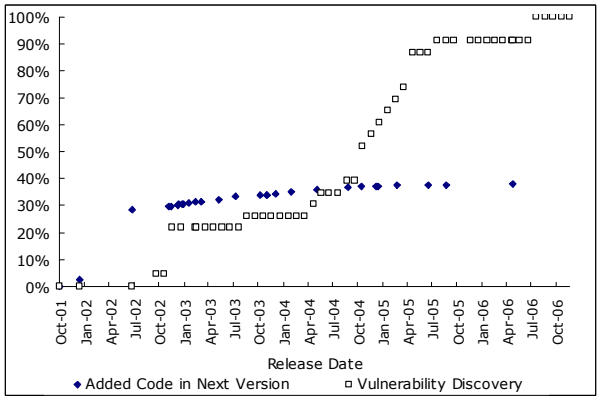

Figure 5. Vulnerabilities Discovered and Code Evolution in Mysql DBMS

In Figures 4 and 5, both vulnerability discovery and software evolution show saturation. However, there is a time gap between the onset of software evolution saturation and that of the vulnerability discovery. From these results, we see that the additional code in the later versions does not exhibit an immediate relationship with vulnerability discovery. However software evolution explains, why software vulnerabilities continue to be discovered. Many vulnerabilities linger for several versions until they are discovered. For a specific version, the vulnerabilities discovered include those introduced in that version plus some of the inherited vulnerabilities in the shared code. This makes modeling the vulnerability discovery in multi-version software more complex. In the next section we present a model to address this.

III. Software Vulnerability Discovery Model

Here we present a new model that accounts for the shared code and hence share vulnerabilities in successive versions of a software.

A. AML Vulnerability Discovery Model

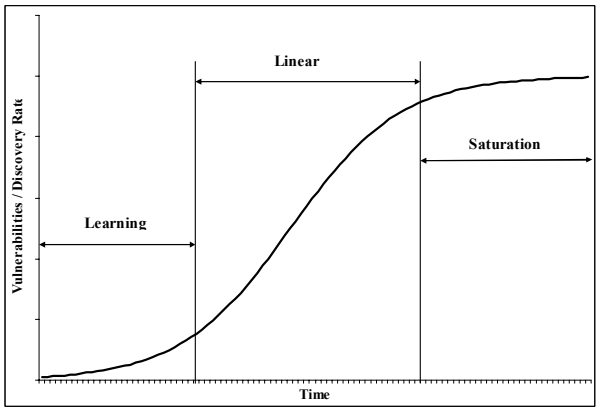

Figure 6. Alhazmi-Malaiya Logistic Model

The basic shape of the Alhazmi-Malaiya Logistic (AML) model is illustrated in Figure 6. At the release of software, the vulnerability discovery rate increases gradually. During this phase, called the learning phase, the software is gaining market share and installed bases is small. In the next phase the trend is linear. The slope here gives the maximum vulnerability discovery rate. The final phase is the saturation phase, where the vulnerability discovery rate slows down, and the cumulative number of vulnerabilities asymptotically approaches its highest value. This three phase logistic behavior is represented by the expression for the cumulative number of vulnerabilities Ω(t) in Equation 1.

$$\Omega(t) = \frac{B}{BCe^{-ABt} + 1} \qquad (1)$$

where B represents the estimated total number of vulnerabilities and the parameters A and C determine the shape of the curve [7]. The model is based on the assumption that the vulnerability discovery process is controlled by the market share of the software and the number of vulnerabilities remaining undiscovered [8]. This model has been found to yield a significant goodness-of-fit for many widely used software systems [7,8,9,21]. However the plots of actual data sometimes show a departure from the model following the release of a new version [8].

It should be noted that this model does not require the knowledge of the market-share data, since the market share variation is implicit in the model itself. Alhazmi and Malaiya have proposed an effort-based model

[21,22] that uses the market share data, however that model is not considered here. When the vulnerability finding effort is irregular, which may be the case for systems with limited deployment, an effort-based model may be needed to project the vulnerability discovery rate.

B. Multi-version Software Vulnerability Discovery Model

The AML model assumes that the software represents an independent and stable implementation. While the model shows a good goodness-of-fit for many systems, it does not explain a frequently observed sudden increase in vulnerability discovery rate system when the next software version is released. This anomaly led us to investigate a multiple version software vulnerability discovery model (MVDM), which takes into account the impact of a new version. Nowadays practically all common programs have several upgraded versions because of growing user and vendor requirements. As a result, multiple versions of some software are under use simultaneously.

The later versions of software are expected to have better software reliability and functionality than the previous ones. This gives rise to a vulnerability discovery trend different from single version VDM such as the AML model, since the software design is changed or new code is added from time to time. A new version typically inherits a significant fraction of the code or implementation from the previous version. Even when the installed base for a specific version may have shrunk significantly, a section of its code may be embedded in the newer and more popular version. A vulnerability found in the shared code of a new version, will also be applicable to the older versions containing the shared code. Here we propose an advanced software vulnerability discovery model which incorporates the impact of vulnerabilities discovered in the code inherited by the later versions.

We assume that shared functionality and shared code inherited from a previous version of software is tested for vulnerabilities during usage, even if the previous version is not in use any more. This is illustrated in Figures 7 and 8.

The first peak in Figure 7 represents the peak vulnerability discovery rate of the initial version of software. The second peak indicates the peak vulnerability discovery rate in the second version. The small peak within the second peak represents the vulnerability discovery in the shared code in the second version. Figure 7 assumes that when the second version is released, its vulnerability discovery rate starts rising while the installed base and hence the vulnerability discovery rate in the first version declines.

The cumulative number of vulnerabilities for earlier version software is presented in Figure 8. The first version software vulnerability discovery model shows onset of a saturation phase, however due to the shared vulnerabilities discovered in the second version of software, the vulnerability discovery rate rises again resulting in a distorted logistic graph. The cumulative number of vulnerabilities Ω(t) for some given software with multiple versions is given by an addition of two terms.

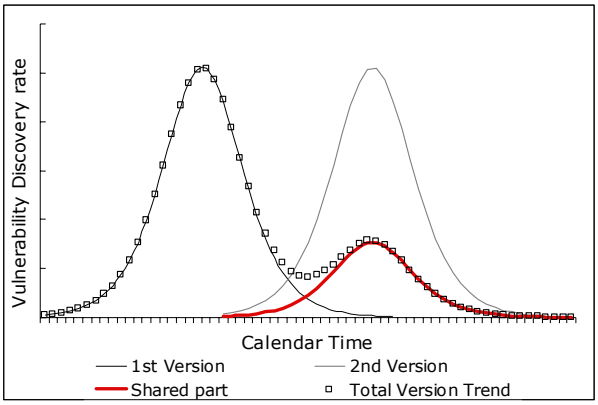

Figure 7. Multi-version Software Vulnerability Discovery Model

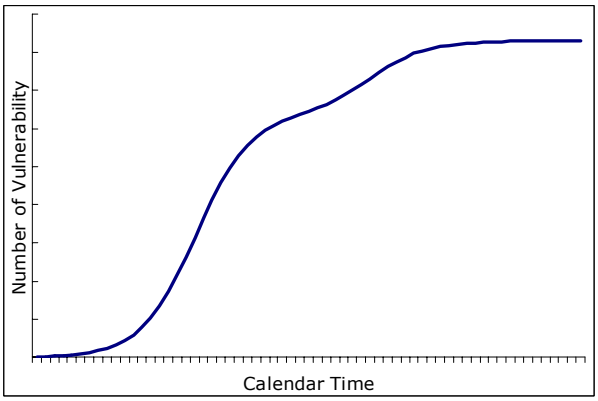

Figure 8. Cumulative vulnerabilities applicable to the earlier version in a multi-version Software System

$$\Omega(t) = \frac{B}{BCe^{-ABt}+1} + \alpha \frac{B'}{B'C'e^{-A'B'(t-\varepsilon)}+1} \quad (2)$$

In Equation 2, the parameter α indicates shared components such as shared code and shared functionality, and the parameter ε is the time lag between the release dates of the two versions. Equation 2 is referred to as the multiple version vulnerability discovery model (MVDM). The two version modeling concept can be generalized to multi-version software

modeling as given in Equation 3.

$$\Omega(t) = \sum_{i=1}^{n} \alpha_i \frac{B_i}{B_i C_i e^{-A_i B_i (t-\varepsilon i)} + 1} \qquad (3)$$

When successive releases are close to each other, the summation will result in a plot that will show delayed onset of saturation, in effect prolonging the linear phase of the logistic curve. In the next section we estimate α by actually measuring the amount of code shared to validate the approach. Further research is needed to develop more convenient empirical methods for estimating α.

IV. Evaluation of the Multi-version Model

Computationally the process of applying the MVDM as given in Equation (2), is an extension of the approach used for using the AML VDM. First we identify the vulnerabilities limited to the earlier version, which we refer to as pure vulnerabilities. We separate the vulnerabilities shared between the earlier and the later version. The fraction of code shared, represented by the parameter α is evaluated by examination of the code for the two versions. AML modeling is done to find the pure vulnerability discovery data for the first version of software. The parameters A, B and C are estimated using statistical model fitting.

We next examine the data for the shared vulnerabilities. Using the value of α, the parameters A', B' and C' are estimated. The plots for the MVDM are then obtained by using addition of the two parts of the model – pure vulnerabilities in the earlier version and the shared vulnerabilities. Goodness-of -fit is then evaluated using the Chi-square tests and P-value evaluation. For comparison, the simple AML VDM is fitted for the entire data for all the vulnerabilities, pure and shared, in the earlier version.

	Previous Version	Next Version	Shared Code Ratio α
Apache	1.3.24 (3-21-2002)	2.0.35 (4-6-2002)	20.16%
Mysql	4.1.1 (12-1-2003)	5.0.0 (12-22-2003)	83.52%

Table 2. Shared Source Code Ratio α

Here we apply the approach to two successive versions of Apache – 1.3.24 and 2.0.35 and two successive versions of Mysql 4.1.1 and 5.0.0. Table 2 gives the values of α, in the last column, which is a measure of the shared code.

Figure 9 shows cumulative vulnerabilities in Apache versions 1.3.x and 2.0.x. The pure vulnerabilities in 1.3.x exhibits a saturation phase in the vulnerability life-cycle. However, when the shared vulnerabilities from the second version are added, the overall plot for 1.3.x shows continuous vulnerability discovery. The fitted plots for pure vulnerabilities of 1.3.x, vulnerabilities in 2.0.x and the shared vulnerabilities are given in addition to overall MVDM model (given by a thick line). This is an example of the superposition effect [10], ignoring which can lead to inaccuracy in the estimates for the vulnerability discovery trend.

Since in open source software, we can analyze the structure to evaluate the shared code, we can estimate one of the major parameters of the MVDM, and thus do a more detailed modeling. The fitted parameter values and the goodness-of-fit results for the Apache HTTP web server are presented in Table 3.

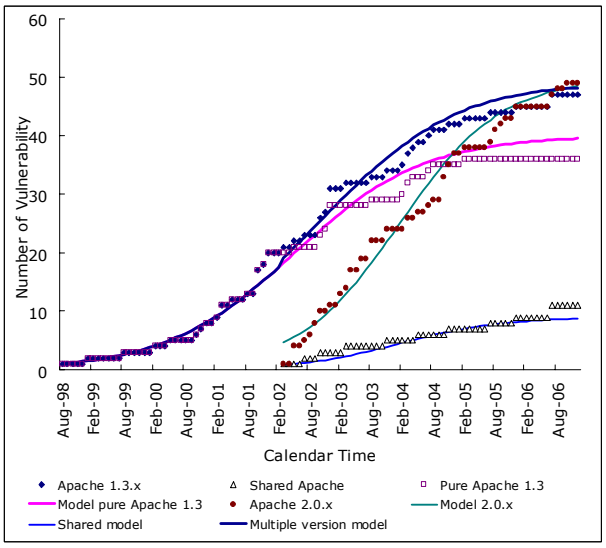

Figure 9. Apache Multi-version software vulnerability discovery modeling

	A	B	C	P value	χ^2	χ^2-critical
Single AML Result	0.0012	54.939	0.701	1	27.79	125.46
MVDM 1st Step	0.0024	36	1			
MVDM 2nd Step	0.0015	54.207	0.171			
MVDM Overall Result				1	9.294	125.46

Table 3. AML and MVDM Fitting result for Apache HTTP Web server

Since in open source software, we can analyze the

structure to evaluate the shared code, we can estimate one of the major parameters of the MVDM, and thus do a more detailed modeling. The fitted parameter values and the goodness-of-fit results for the Apache HTTP web server are presented in Table 3.

In Table 3, the top row gives the results for an application of the simple AML VDM. The next two rows show the fitted parameters for the two steps for the MVDM. The last row gives the goodness-of-fit for the overall MVDM. Both models show significant goodness-of-fit through chi-square test results. The Chi-square values suggest that the MVDM gives a better fit than the existing AML VDM. It should be noted that the shared code can be evaluated at the very release of the later version, and thus α can be estimated before a significant number of shared vulnerabilities have been found.

To verify the general applicability of the multiple software vulnerability discovery modeling presented in the previous subsection, we applied it to the Mysql data using the same methodology. We used Mysql version 4.1.x and 5.0.x, because the previous version of Mysql is 3.2x, and its original source was coded only for Linux. From versions 3.22 onwards, it was available for Windows version software also. Since OS conversion affects the number of users, the comparison between 3.2x version and 4.x version would not be meaningful. The results of the application of the MVDM for 4.x version and 5.x version are presented in Figure 10. The computation method and the plots obtained are similar as the two Apache versions.

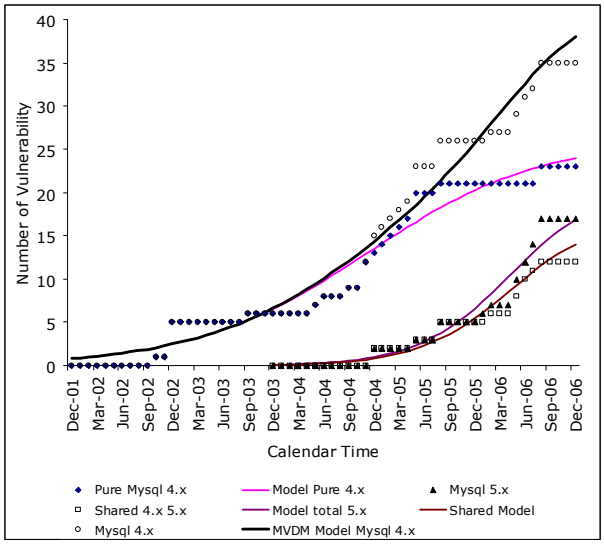

Figure 10. Mysql Multi-version software vulnerability discovery modeling

The results for Mysql show the same pattern as for Apache versions that we considered. The pure vulnerabilities of Mysql version 4 show saturation from mid 2005, however the vulnerabilities shared with the later version have continued to be discovered, again showing how the vulnerability discovery in the initial version software is influenced by the later version. The fitting results are shown in Table 4. It shows that the MVDM results in a lower χ^2 value and thus it provides a better fit compared with using the single AML VDM.

The proposed MVDM explicitly models the shared code and thus permits more accurate modeling. This can potentially be used to develop methods with high predictive capability with further investigations. The limitation of this approach is that it uses more parameters compared with a single vulnerability discovery model. However, the parameters arise because of the use of shared code, and thus this modeling approach is meaningful for generalized software vulnerability discovery modeling.

	A	B	C	P value	χ^2	χ^2-critical
Single AML Result	0.0012	60	0.8	0.99	37.12	80.232
MVDM 1st Step	0.0036	26.207	1.27			
MVDM 2nd Step	0.0088	20.818	10.19			
MVDM Overall Result				1	35.35	80.232

Table 4. AML and MVDM Fitting result for Mysql DBMS

V. CONCLUSIONS

Predicting the vulnerability discovery rates in major software packages is important for both developers and users. A few vulnerability discovery models have recently been proposed. However these do not take some of the characteristics of multi-version software into account. We have examined several versions of two open source software: the Apache HTTP Web Server and Mysql DBMS to identify the relationship between software evolution and vulnerability discovery. We also proposed a new model for estimating the vulnerabilities by taking into account the shared code among successive versions. The MVDM was validated with data obtained from the NVD. The proposed model considers the impact of the life-cycles of the individual versions on the vulnerability discovery trend for overcoming the limitations of existing simple VDMs.

For the commercial systems that are not open source, the successive versions of the source code will not be available outside of the developing organization.

However the approach examined in this paper can be used internally within the organization.

Further research is needed to enhance the accuracy of the proposed approach. The analysis needs to be repeated to other types of applications to establish its general applicability. Source code base analysis may not be enough to identify all shared vulnerabilities, because some vulnerabilities may be inherent in a procedure or approach coded differently in different versions of the software. Further research is needed to compare the higher- level behavior of the multiple versions of some software. The need to evaluate the degree of code sharing may be eliminated if methods can be developed to estimate it empirically.

References

[1] National Vulnerabilities Database, http://nvd.nist.gov, 2006

[2] Y. K. Malaiya, J. Denton, "What Do the Software Reliability Growth Model Parameters Represent?" Int. Symp. on Software Reliability Engineering, 1997. pp. 124-135.

[3] ReliaSoft Publishing, "Reliability Growth and Repairable System Data Analysis Reference", http://www.weibull.com/relgrowthwebcontents.htm

[4] J. D. Musa, A. Ianino, K. Okumuto, "Software Reliability Measurement Prediction Application", McGraw-Hill, 1987.

[5] R. Anderson, "Why information security is hard – An Economical Prospective", Proceedings of the 17th annual Computer Security applications conference, 2001, pp. 358-365.

[6] E. Rescorla, "Is finding security holes a good idea?", In Workshop on Economics and Information Security, 2004. Minneapolis, Minnesota, pp. 14-19.

[7] O. H. Alhazmi and Y. K. Malaiya, "Modeling the Vulnerability Discovery Process," Proc. Int. Symp. Software Reliability Eng, Nov. 2005, pp. 129-138.

[8] O. H. Alhazmi, Y. K. Malaiya, I. Ray, "Measuring, Analyzing and Predicting Security Vulnerabilities in Software Systems," Computers and Security Journal, Volume 26, Issue 3, May 2007, pp. 219-228

[9] O. H. Alhazmi, Y. K. Malaiya, "Application of Vulnerability Discovery Models to Major Operating Systems," to appear in IEEE Trans. Reliability, Dec 2007.

[10] S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, A. Mockus, "Does Code Decay? Assessing the Evidence from Change Management Data," IEEE Transactions on Software Engineering, 2001, 27(1), pp. 1-12.

[11] C. Izurieta, J. Bieman, "The Evolution of FreeBSD and Linux", ISESE'06, September 21–22, 2006, Rio de Janeiro, Brazil, pp. 204-211.

[12] A. Ozment, "Software security growth modeling: Examining vulnerabilities with reliability growth models." In Proceedings of the First Workshop on Quality of Protection, September 2005, Milan, Italy, pp. 223-233.

[13] Netcraft, "April 2007 Web Server Survey", http://news.netcraft.com/archives/Web_server_survey.html

[14] The Mysql Product achieve, http://downloads.mysql.com/archives.php (2006)

[15] The Apache Software Foundation achieve, http://archive.apache.org/dist/httpd (2006)

[16] A. Mockus, R. T. Fielding, J. D. Herbsleb, "A case study of open source software development: the apache server", In *ICSE* (2000), pp. 263–272.

[17] M. W. Godfrey, Q. Tu, "Evolution in open source software: A case study", In *ICSM*(2000), pp. 131–142.

[18] Counting Source Lines of Code (SLOC), http://www.dwheeler.com/sloc/

[19] MITRE, http://www.mitre.org

[20] A. Ozment, S. E. Schechter. "Milk or Wine: Does Software Security Improve with Age?", Proceedings of The Fifteenth Usenix Security Symposium. July 31 - August 4 2006, Vancouver, BC, Canada, pp. 31-39.

[21] S.W. Woo, O. H. Alhazmi, Y. K. Malaiya, "Assessing Vulnerabilities in Apache and IIS HTTP Servers", (DASC'06), September 29-October, 2006, Indianapolis, USA, pp.103-110.

[22] S-W. Woo, "An Analysis of Vulnerabilities in Web-servers and Browsers Using Time based and Effort-based Models," Thesis, CS Dept, Colorado State University, 2006.