THESIS

VULNERABILITY DISCOVERY IN MUTIPLE VERSION SOFTWARE SYSTEMS: OPEN SOURCE AND COMMERCIAL SOFTWARE SYSTEMS

Submitted by

JIN YOO KIM

Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2007

COLORADO STATE UNIVERSITY

June 20, 2007

WE HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER OUR SUPERVISION BY JIN YOO KIM ENTITLED VULNERABILITY DISCOVERY IN MULTIPLE VERSION SOFTWARE: OPEN SOURCE AND COMMERCIAL SOFTWARE SYSTEMS BE ACCEPTED AS FULFILLING IN PART THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE.

Committee on Graduate Work

[redacted]

Dr. Anura Jayasumana

[redacted]

Dr. Indrakshi Ray

[redacted]

Adviser Dr. Yashwant K. Malaiya

[redacted]

Department Head Dr. Darrell Whitley

ABSTRACT OF THESIS

VULNERABILITY DISCOVERY IN MUTIPLE VERSION SOFTWARE SYSTEMS: OPEN SOURCE AND COMMERCIAL SOFTWARE SYSTEMS

The vulnerability discovery process for a program describes the rate at which the vulnerabilities are discovered. A model of the discovery process can be used to estimate the number of vulnerabilities likely to be discovered in the near future. Past studies have considered vulnerability discovery only for individual software versions, without considering the impact of shared code among successive versions and the evolution of source code. These affecting factors in vulnerability discovery process need to be taken into account estimate the future software vulnerability discovery trend more accurately.

This thsis examines possible approaches for taking these factors into account in the previous works. We implemented these factors on vulnerability discovery process. We examine a new approach for quantitatively vulnerability discovery process, based on shared source code measurements among multiple version software system. The applicability of the approach is examined using Apache HTTP Web server and Mysql DataBase Management System (DBMS). The result of this approach shows better goodness of fit than fitting result in the previous researches. Using this revised software vulnerability discovery process, the superposition effect which is an unexpected vulnerability discovery in the previous researches could be determined by software discovery model.

The multiple software vulnerability discovery model (MVDM) shows that vulnerability discovery rate is different with single vulnerability discovery model's (SVDM) discovery rate because of newly considered factors. From these result, we create and applied new SVDM foropen source and commercial software. This single vulnerability process is examined, and the model testing result shows that SVDM can be an alternative modeling. The modified vulnerability discovery model will be presented for supporting previous researches' weakness, and the theoretical modeling will be discuss for more accurate explanation.

JIN YOO KIM
Computer Science
Colorado State University
Fort Collins, Colorado 80523
Summer 2007

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Software Development and Vulnerability

Software system development is a complex process. Additional or modified software reliability and functionality requirements force developers to add or change source code and its design [15, 25, 12]. As software evolves, developers are using software reliability engineering methodolgy to create reliable software during the development process [35]. However, software defects, which are produced during the development process, may not be found and removed from software during coding and testing, and the remaining software defects are discovered after release.

A software vulnerability is a special type of software defect that has the potential to violate software security. Although the number of vulnerabilities are few compared with overall defects [2], the vulnerabilities are the main cause of information security incidents. The number of incidents using software vulnerability has been increasing rapidly [32, 45, 10]. Software vulnerabilities cause security problems such as information leaks and system maintainence issues, as well as possibility of financial loss [54]. The cost of software vulnerabilities is hard to quantify, since vulnerabilities can be exploited through various methods, which may result in a decreased market share for the affected software product. Because of the need to find and mitigate the effects of software vulnerabilities, government, organization and security consulting companies have created software vulnerability databases such as the National Vulnerability Database [41], CERT [14], Secunia [49], Symantec Internet Security Report [55], and securityfocus [48]. Vulnerability descriptions, and the severity of software vulnerabilities are recorded in the vulnerability database [47]. A severity standard for software vulnerability has been in discussion for a long time. MITRE [30] provides a quantitative severity standard called CVSS (Common Vulnerability Scoring System), which is widely used for software vulnerability severity evaluation and for proper prevention of the vulnerabilities.

As the importance of software vulnerability prevention increases, vulnerability prediction has stated to get attention [37, 2, 3, 46, 56]. Quantitative vulnerability discovery modeling can be used for estimaing vulnerability trends in the near future. Previous studies on vulnerability discovery have focused on a single software version. However, most software projects involove multiple versions to continue to improve software reliability and functionality. Therefore, we need advanced vulnerability discovery modeling methods to estimate the number of software vulnerabilities.

Software evolution is an important consideration for quantitative analysis of software vulnerability. Previous quantitative analysis for the vulnerabilities had considered a few factors for the estimation of software vulnerabilities. These factors are calendar time, estimated software age, and others. The vulnerability discovery models using these limited factors cannot describe the total software vulnerability discovery process because of the different characteristics of each successive release. Therefore, additional factors need to be reconsidered for accurate estimation of the number of software vulnerabilities. In the next chapter, we will examine software vulnerability discovery modeling approaches to determine which are fit better.

1.2 Related Work

Software reliability modeling has been investigated using several approaches. Structural approaches, function tree based formal modeling [19] and reliability modeling using software environment effect [27, 16] have been examined for estimating software defect density. Structural analysis helps identify specific vulnerability components, however, this method requires a lot of software testing and time. The alternative to a structural approach is quantitative vulnerability modeling, which has seen increased interest recently as it provides a simple way to evaluate software security.

A few quantitative software vulnerability models have recently been proposed by a few researchers. Musa introduced software reliability engineering [36, 37, 35, 28, 34], and also proposed some of the software reliability growth models [37, 33]. For software quality measurement, he proposed a logarithmic poisson software reliability growth model [37]. Using the concept of SRGMs (Software Reliability Grwoth Models), software vulnerability discovery models (VDMs) have been developed by other researchers. Eric Rescorla [46] showed a software reliability growth model with quantitative analysis of the ICAT (now NVD) data [41], and suggested quadratic and exponential software vulnerability models. Anderson [8] proposed a thermodynamic vulnerability discovery model. Omar and Malaiya [3] proposed Logistic vulnerability discovery model (AML - model) that has flexible symmetric software vulnerability discovery, and the authors compared well-known software vulnerability discovery models using several software vulnerability datasets and showed goodness of fit for almost all software systems. Omar and Malaiya [2, 4] show goodness-of-fitness for their vulnerability discovery models. Woo [57] presented a software discovery trend fitting with AML modeling and showed that AML modeling is the most accurate modeling for a software system vulnerability prediction model across various software system.

Using a basic VDM can only go so far to explain software vulnerability trends. One element not examined previously is the effect of software evolution on vulnerability, and the connection between software evolution and its requirement. This relationship between software evolution and software requirements is well represented as a case study of Linux in [20]. Mockus et al.[17] have examined software source code decay introduced by its evolution. Clemente [23] has presented a linear shape evolution trend of source code in FreeBSD. Ozment [42] applied a software reliability growth model, and in [9], he focused on foundational source code share and foundational vulnerabilities in the OpenBSD operating system. Also he measured software age based on the software reliability growth model with discovered vulnerabilities. With a statistical approach, Paul [26] showed time-based and metrics-based defect forcasting with OpenBSD. These quantitative vulnerability modeling approaches, and the software evolution process will be considered in this work.

1.3 Contributions

This thesis presents several novel software vulnerability discovery models that have not been examined by other researchers.

Chaper 4 explores existing software vulnerability discovery models that focus on one version, or the entire vulnerabilities of specific software which does not separate several versions of the software. The limitation of previous software vulnerability discovery modeling is that the models do not consider software life time, or other factors that may cause an inaccurate modeling. This thesis aims to provide a software vulnerability discovery model for multiple version software systems which has not been explored in the past. A multiple version software vulnerability model will be presented to explain the superpostion effect, which comes from shared vulnerabilities among different versions of software. This multiple version software vulnerability discovery modeling work will include software evolution. This thesis also proposes an asymmetric vulnerability discovery model for predicting future vulnerability discovery which is caused by later version software release. The other vulnerability discovery model is suggested to test version software release effect. This new model also supports the reason of the superposition effect. Through examination of the fitting results, Weibull VDM (modified from a multiple VDM) will be introduced as a method for modeling single software.

1.4 Organization of the Thesis

This thesis is organized as follows. Chapter 2 introduces the factors which can affect the vulnerability discovery trends. Chapter 3 discusses the software evolution effect on vulnerabilities. Chapter 4 presents the software vulnerability discovery models for both individual and multiple version software. Chapter 5 presents multiple version software vulnerability discovery model fitting results for open source software systems using Apache HyperText Transfer Protocol (HTTP) web server and Mysql Database Management System (DBMS). Chapter 6 examines the software vulnerability discovery modeling for closed source software using Windows operating systems (OS) and Internet Explorer. Chapter 7 analyzes each vulnerability discovery model using actual data, and in Chapter 8 concluding comments are presented on approaches for software vulnerability discovery modeling. In addition to these we compare alternative software vulnerability discovery modeling approaches for each factor.

Chapter 2

Vulnerability Discovery Factors

Software vulnerabilities represent a fraction of the residual defects originally produced during the development process. Software defects are influenced by several factors such as program complexity, work standards, development team capabilities, testing time, and other factors during software development. These factors have been used for predicting the total number of software defects, or estimating software reliability [18, 34, 50]. Using a similar approach, we can define a metric *vulnerability density*. Using data from several case studies, using estimates for defect densities and the percentage of defects that are vulnerabilities [2], the total number of vulnerabilities can be estimated. The vulnerability discovery rates are influenced by various internal or external factors such as software usage and its life time, among others. Therefore, prediction of the software vulnerability discovery rate is not straight forward. Since the existing static approaches do not model the discovery rate of vulnerabilities, we cannot estimate the expected discovery rate.

In this chapter we will examine the significant factors which can affect software vulnerability discovery process, such as the size and amount of shared software code, software life time, software usage, and beta testing.

2.1 Software Code

Several studies have estimated prediction of software defects using software source code. These factors indicate source code size and modularity. However, for predicting vulnerability discovery in multiple version software we need to look at factors such as shared source code, shared modularity and common functionality among different versions.

2.1.1 Source Code Size

Akiyama [1] claims that software source code size is strongly related with software defects. On the other hand, Ozment expresses the view that [9], since vunerability rates coming from additional software code is lower than in the original code, the number of vulnerabilities is not directly related with the source code size. In other words, software defect density is related to software structure and development plan. These two views are comparable, however, since they both effectively explain software reliability growth modeling, we will follow both of them.

2.1.2 Shared Source Code

Software size is one potential prediction of the number of vulnerabilities. For just one version of software, source code defects are directly related with software source code size. However, software vulnerability in multiple version software comes from shared code. In [9], the vulnerability density of additional software source code is lower than legacy code, but its code size is larger than legacy code. Therefore, in the software evolution model concept, shared source code is one major contributer to software vulnerability. Modeling for vulnerability discovery process need to be considered the shared code effect with the code size.

2.2 Software Age and Usage

The aging of software is the result of software source code modification or decay as time passes. However, in this thesis, we will examine the software age factor as software discovery factors which are not used in software evolution. Software vulnerability can be revealed during testing or under usage. Thus, software aging and usage can affect the software vulnerability discovery rate. These factors are related to market share and maintenance policy. To show how software vulnerability discovery is affected by these, we will review Windows OS share and the maintenance policy of Microsoft as a case study.

2.2.1 Software Life Time and Share of Installed Base

Since a specific installed version software is typically used until the release of a new version, its life time depends on the plans of the software development team who desire for the customer to upgrade to the newer version. To determine typical software life time, we use software market share [13, 22] to find out the end of software life time. In Figure 2.1, the X axis is calendar time and the Y axis indicates the fractional installed base for each Windows OS version. In Figure 2.1, Windows 98 share decreased after Windows XP was released. Currently Windows 98 has not met the end of use, and vulnerabilities continue to be found in it. Therefore, the impact of vulnerability discovery rate from software life time is primarily dictated by the software development policy.

Software vulnerabilities are affected by installed base software share and usage. These factors have not been considered before. We may consider that install-based software share is one indicator of software usage. However, these are not synonymous. Software usage is based on time in use of the software, and is not affected by software install-based share. The relation between software install-based share and vulnerability discovery rate shows that the software usage effect is larger than install-based software share. The install-based software share of Windows OS is presented in Figure 2.1 [21, 22].

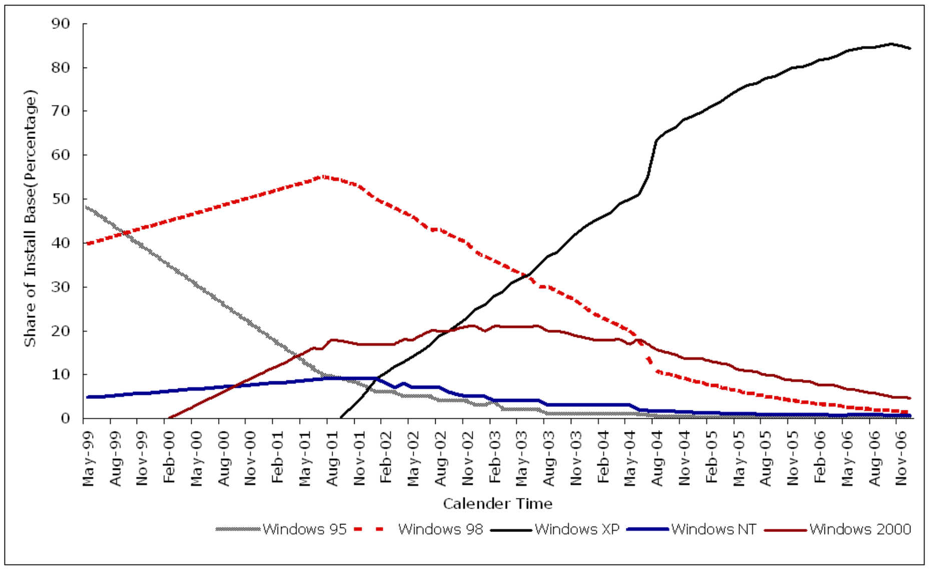

Figure 2.1: Windows OS Install-Based Usage

As shown in Figure 2.1, the Windows 98 share was reduced after the release of the next version of software. The vulnerability discovery trend of Windows 98 is presented in Figure 2.2.

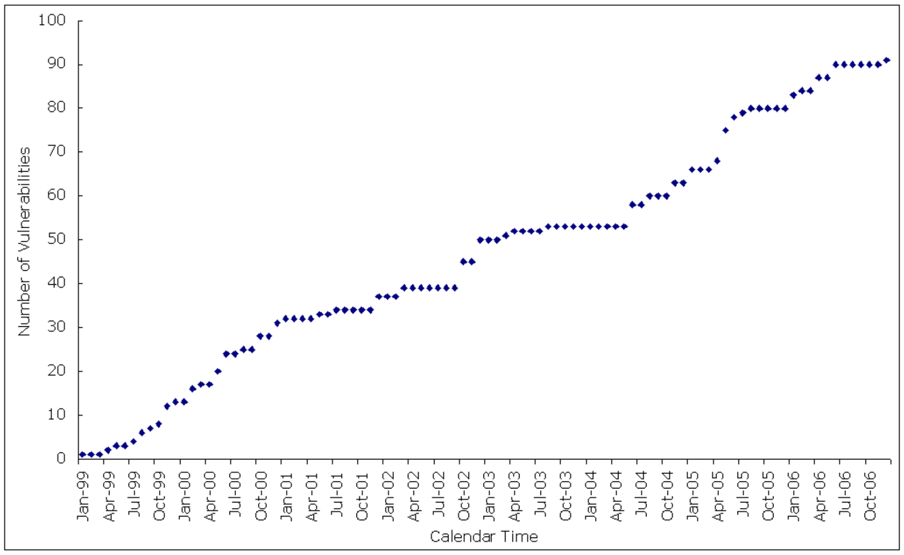

Figure 2.2: Windows 98 Vulnerability

In Figure 2.2, the X axis is calendar time, and the Y axis is the cumulative number of vulnerabilities. It shows that the vulnerability discovery trend is still increasing. From the Figure 2.1 and 2.2, there is no significant relation between software install-based share and vulnerability trend. Therefore, the install-based software sharing does not affect the software vulnerability discovery trend in any significant way. Although the install-based share is decreasing, the software vulnerability discovery trend is still increasing.

Using Weibull distribution, which fits the vulnerability discovery trend of Windows 98, examined in detail in Chapter 6, we can represent Windows98 vulnerability discovery trend as shown in Figure 2.3.

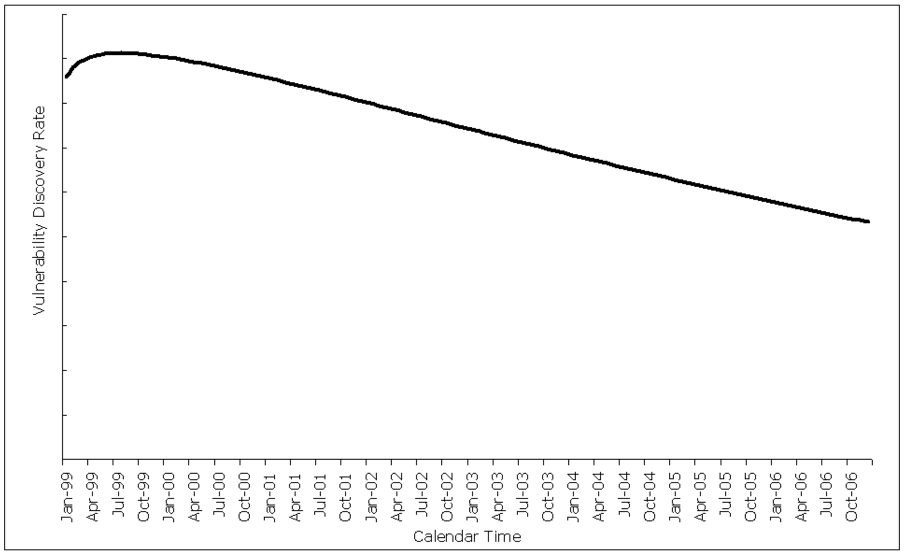

Figure 2.3: Windows 98 Vulnerability Discovery Trend

In Figure 2.3, the vulnerability discovery rate of Windows 98 shows slightly decreases. From the Figure 2.1 and 2.2, we find that the relationship between software share, and the software vulnerability discovery rate coming from its cumulative vulnerability discovery trend does not match up. Therefore, software usage factors are more important than market share on vulnerability discovery rate.

2.2.2 Maintenance Effect to Software Share and Usage

In the previous section, we examined how software vulnerability discovery trends and rates are affected by software usage. In this section, we will review the impact on software usage from the software maintenance policy of the project team. One example is presented in Table 2.1 [29].

Product Name	Release Date	General Support	Extention Support
Windows 98	06-30-1998	06-30-2002	07-11-2006
Windows 2000	03-31-2000	06-30-2005	07-13-2010
Windows XP	12-31-2001	04-14-2009	04-08-2014
Windows 2003	05-28-2003	2 years after next version release	5 years after

Table 2.1: Microsoft Windows OS maintenance policy

Software install-based share is related to software release dates, and this is represented in Figure 2.1. However, the software age problem is tied more to the software maintenance policy. If the maintenance is not supported, the vulnerabilities discovered through software usage decreases drastically because the software will not be used by users as before. This is one important factor impacting the software vulnerability discovery rate.

2.3 Software Testing

Source code accessibility for testing, and the level of software testing can determine software vulnerability discovery [47]. This section deals with the testing factor. Testing approaches can be categorized into several types, however, in this paper we will just examine testing methods in open and non-open source software.

2.3.1 WhiteBox and BlackBox Testing

Software testing can be categorized as whitebox or blackbox testing. Whitebox testing has an advantage in that it can perform structual software testing with source code and modules included operation testing. Blackbox testing only deals with binary code and operational testing. The discrepancy creates a difference in software reliability quality, and the software vulnerability discovery rate after software release. We assume that the software systems being handled in this paper have similar testing techniques implemented by the development teams, but not necessarily the users.

2.3.2 Open Source and Closed Source Program Comparison

Depending on the development process, a software system is categorized as either open or closed source software. There are many differences between open and closed source software development. For example, the development process of an open source program is based on volunteers who are interested in the project [20, 43]. Another difference between open and closed source software is testing accessibility. An open source program can be tested both by official testers and users using whitebox testing methodology, however, closed system testing methodoloy is only available in blackbox testing to users. Since closed systems do not provide their source code, its structual test may not be performed by users, and this creates a vulnerability discovery difference between open and closed source software systems. In chapter 6, we will examine different software vulnerability discovery modeling and show the best fitting model for closed source software system.

2.3.3 Beta Testing Version Software

One of the factor that makes vulnerability discovery between open and closed source software is the effect of beta test version software. A typical beta version uses the source code of previous successive version software and it also includes new structure and codes. Since this reused code, and the new code or structure includes vulnerabilities and the testing cases are focused more on the current vulnerabilities than inherited ones, the vulnerability discovery rate displays abnormalities. These factors should be considered for modeling in software vulnerability discovery modeling, since it can affect unexpected vulnerability discovery.

Chapter 3

Software Evolution and Vulnerability Discovery

3.1 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 thesis is to understand the relationship between evolution and vulnerability discovery.

3.1.1 Software Evolution Trend

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. [31] have identified the environmental factors leading to software evolution and its development. Godfrey and Tu [20] 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.

	Version 1.3	Version 1.3.37	Version 4.0.0	Version 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 3.1: 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 [51]. The results are shown in Table 3.1. 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).

In Figure 3.1, the software growth in Apache HTTP Web server shows saturation. To determine the similarity between different software systems, we examined Mysql DBMS source code, and the result is presented in Figure 3.2.

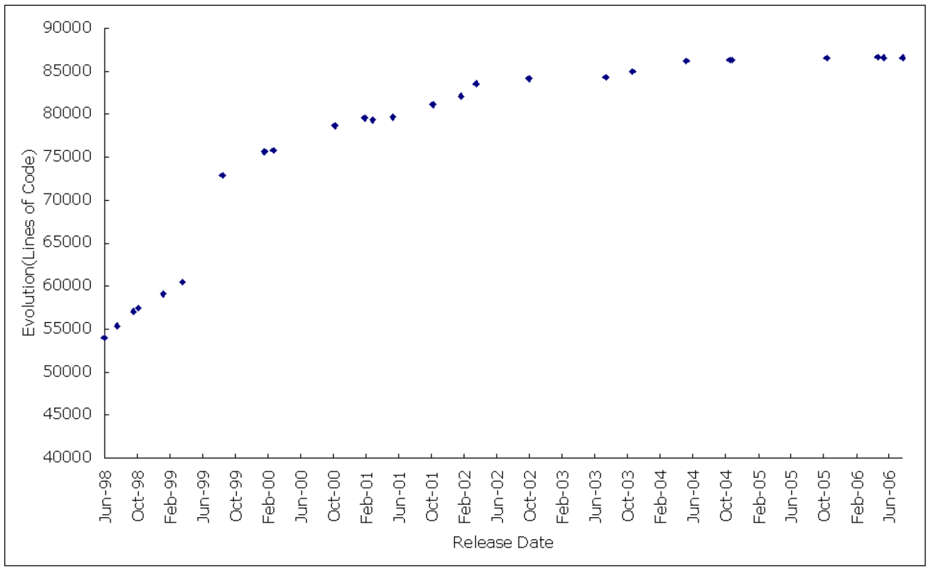

Figure 3.1: Apache HTTP Web Server Evolution Trend

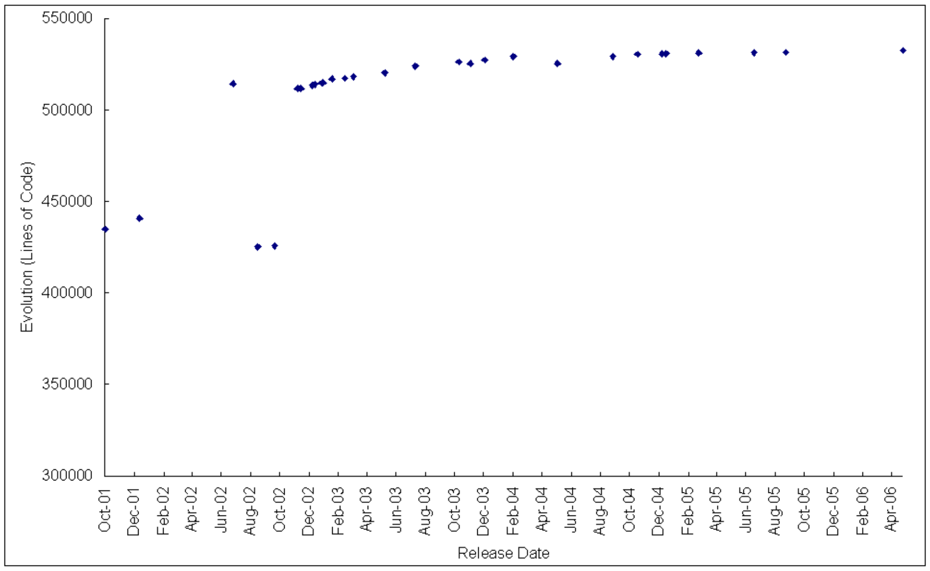

Figure 3.2: Mysql DBMS Evolution Trend

Apache HTTP Web server has larger software source code modification (43%) than Mysql DBMS (31%). The reason for this evolution may be a growing of software requirements. The major initial versions of systems were released in 1998(Apache) and 2001(Mysql), however, the DBMS was well-defined software, and its development has been more stable than a HTTP Web server.

We can see that the evolution of the software is intuitively logarithmic shape based on time. In Figure 3.1 and 3.2, the evolution of subversion software is determined by reliability than functional requirements because there were minor changes in their functionality but several patches for security was in their software system.

3.1.2 Software Code Decay

With software evolution, shared code is a major factor that impact vulnerability discovery. To measure the shared code of Apache HTTP server and Mysql DBMS in terms of lines of code, we used Diff and Line counter tools [44, 52]. These tools are basically installed in Unix and Linux. For more accurate counting, we consider only programming source code such as .c, .sh. .cpp and others. Through this methodolgies we could determine which legacy code was included in later version software. This result is shown in Figure 3.3 and 3.4.

Figure 3.3 shows the fraction of the code in later version of Apache that was inherited from the initiated version. Shared software code decay also shows saturation phase at the end of software life cycle.

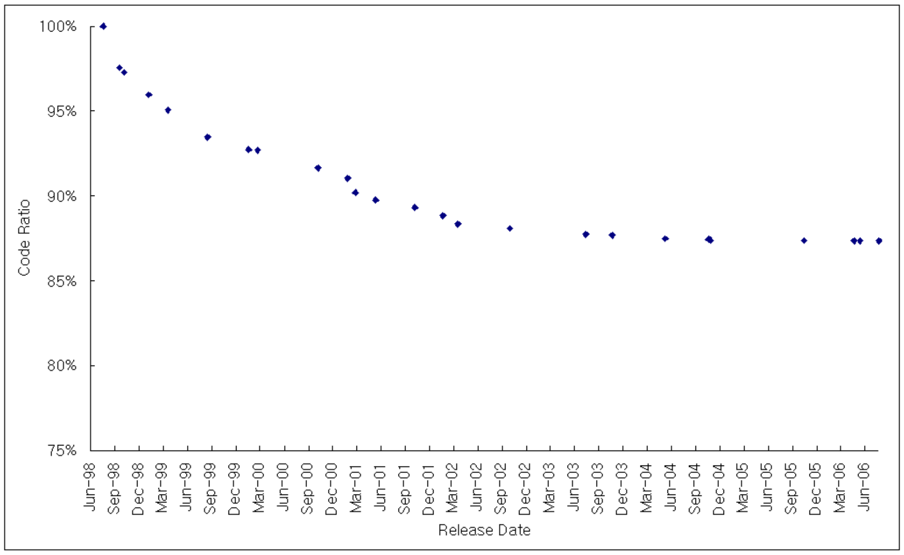

Figure 3.3: Apache HTTP Web Server Initial Code Decay

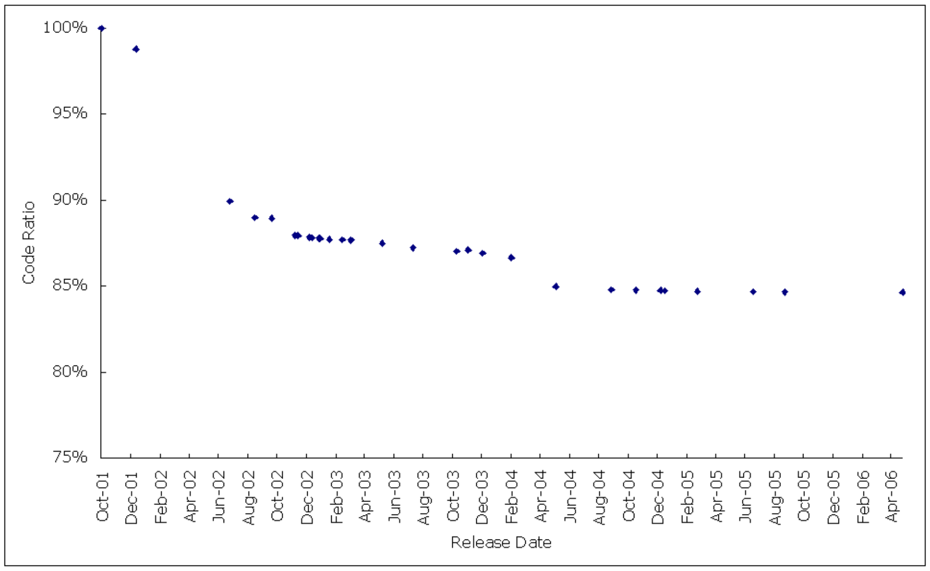

Figure 3.4: Mysql DBMS Initial Code Decay

Figure 3.4 shows a similar shape with Figure 3.3. We can determine the reuse ratio of initial code in Apache HTTP Web server. The first phase of the graph shows fast decline, however, the changed amount of initial code in later versions is lower than the initial phase.

To figure out the advanced proportion of the software, we compiled the legacy code ratio in later version. This work required a recursive file comparing process. Results of this work are presented in Figure 3.5 and 3.6.

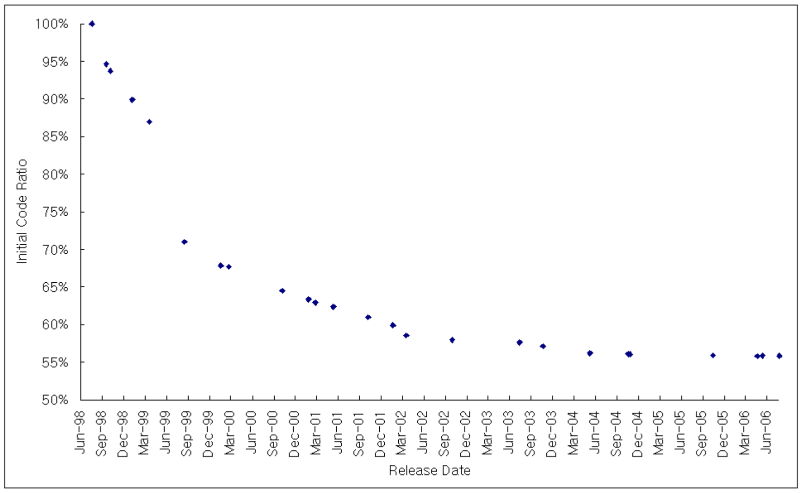

Figure 3.5: Remained Shared Source Code Trend in Apache

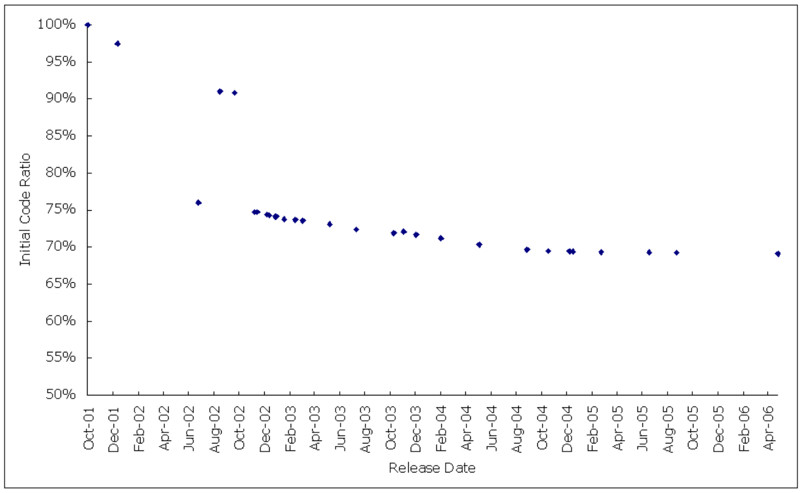

Figure 3.6: Remained Shared Source Code Trend in Mysql

In Figure 3.5 and 3.6, shows the evolution of the Apache HTTP Web server and Mysql DBMS source codes. In 1.3.x version of Apache, the legacy code from the software changed about 20%, the amount rose to 48% in the most recent version. If this source code change came from additional requirements, we could get a meaningful result that would illustrate the relationship between legacy code and security. With same method, in Figure 3.6, we get major Mysql DBMS shared source code lines. In the 4.0.x version of Mysql, we can gauge the evolution of software. The 16% initial code of Mysql converted to 31% new code in the latest version software. These results of Apache and Mysql show similar shapes to each other.

3.2 Code Evolution and Code Share Between Software Subversions

We have examined software evolution in Figure 3.1, 3.2, 3.3 and 3.4. However, these plots do not show the impact of service packs release time to time. In Figure 3.7, we examine Mysql 4.1.x source code share in 5.0, 4.1 and 4.0 versions for all the subversions. The shared code ratios for later version software in Mysql is also presented in Figure 3.7.

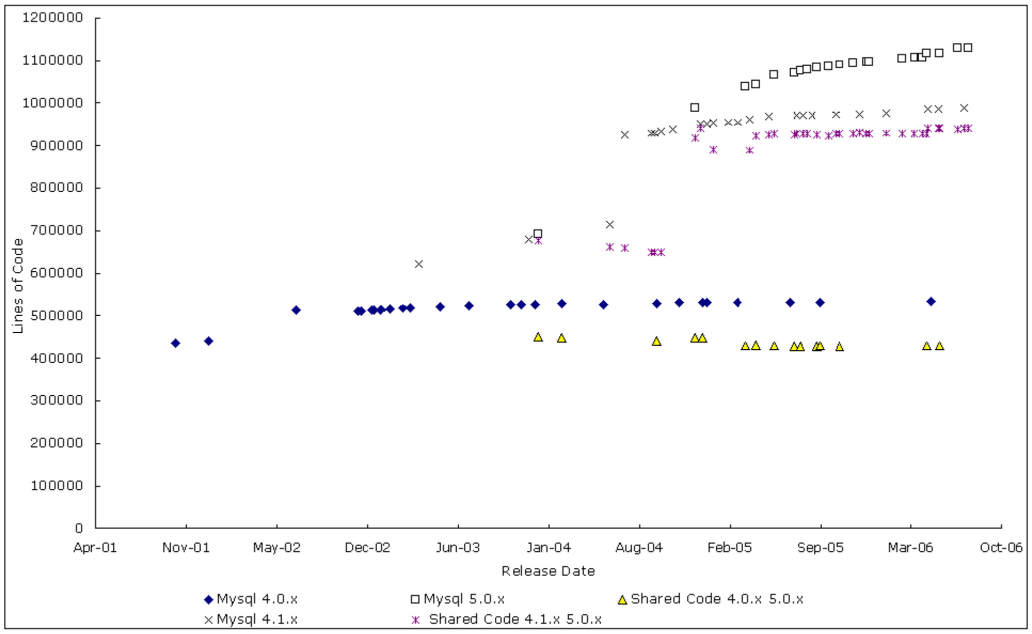

Figure 3.7: SubVersion Shared Source Code Trend in Mysql

Figure 3.7 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.

3.3 Software Evolution and Vulnerability Discovery

Since newer version software has more secured function and procedure in itself, vulnerability in previous version software can be found from changed source code. Usually, these vulnerabilities are anounced by vendors and representive case is CISCO software patch anouncement [53]. Using comparison two version of software can call old version software vulnerability. This procedure is one main cause of superposition effect of vulnerability discovery and increase of security accidents. However, upgrade security function and source code can be utilized by malicious users after latest version of software.

The software vulnerability discovery trend is related to software evolution. Sometimes vulnerability can be found right after the next version of software is released. To figure out the relation we will examine the software evolution and vulnerability discovery trend in Figure 3.8 and 3.9.

Apache HTTP web sever software evolution, and its vulnerability trends, are presented in Figure 3.9. The evolution and vulnerability discovery trend shows a saturation phase. However, the plots for software vulnerability are growing slower than the software evolution model. For more testing, Mysql vulnerability discovery trends and its software evolution trends are presented in Figure 3.9.

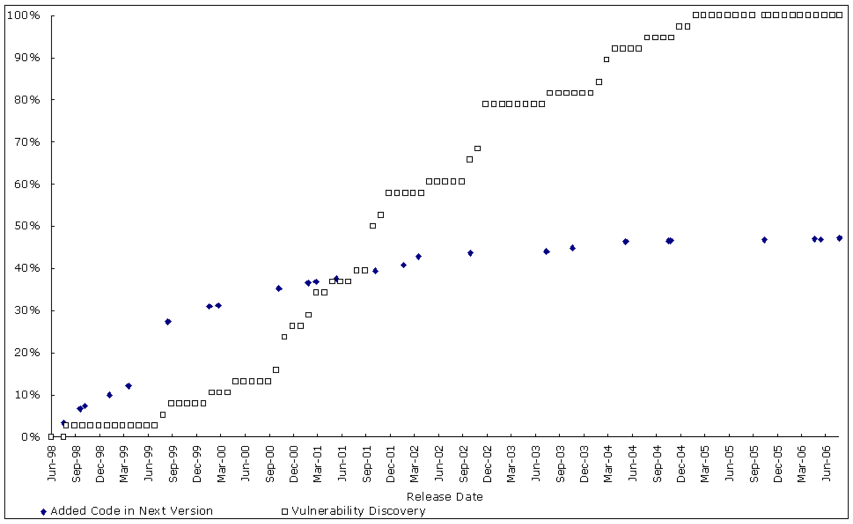

Figure 3.8: Apache Added Code and Vulnerability Discovery Trend

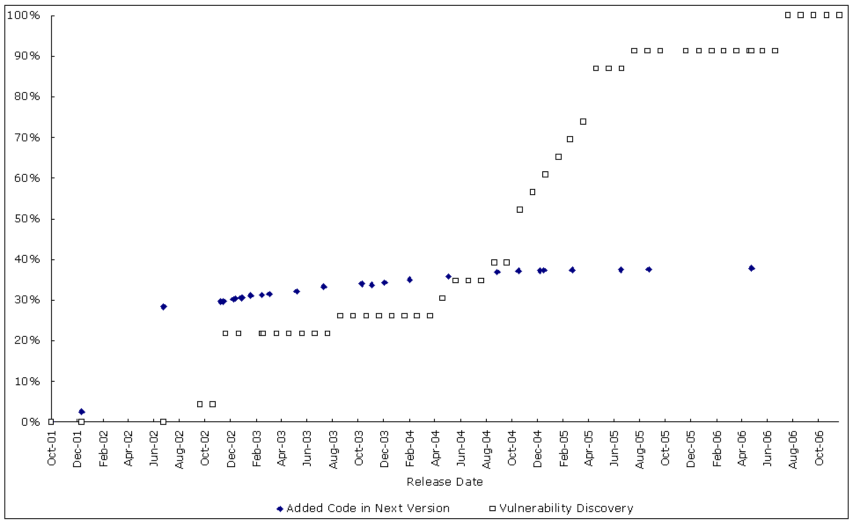

Figure 3.9: Mysql Added Code and Vulnerability Discovery Trend

In Figures 3.8 and 3.9, 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 vulnerabilites 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.

Chapter 4

Vulnerability Discovery Models

Software vulnerability discovery models are used to estimate software vulnerabilities in the testing phase of software reliability engineering [11, 36]. The software vulnerability discovery models have been proposed by a few researchers recently. All software VDMs focuses on single version software vulnerability discovery trends, or deal with the overall version software vulnerability discovery process, using a time based model. We review the models in the two chapter and we will suggests needed additions to the model to explain abnormalities observed in the trends.

4.1 Basic Software Vulnerability Discovery Models (VDM)

4.1.1 Musa-Okumoto Logarithmic VDM

The Musa-Okumoto reliability growth model [37] is a widely used software reliability growth model in software reliability engineering. It has also been applied to vulnerability discovery process. This model is represented in Equation 1.

$$\Omega(t) = \beta_0 \ln(1 + \beta_1 t) \tag{1}$$

In Equation 1, the parameter β_0 is the scale parameter of this model, which can decide software vulnerability number, the parameter β_1 is the shape parameter, which determines the vulnerability discovery model shape. This vulnerability discovery model is presented figure 4.1, the X axis indicates the time parameter and the Y axis is a cumulative number of software vulnerabilities.

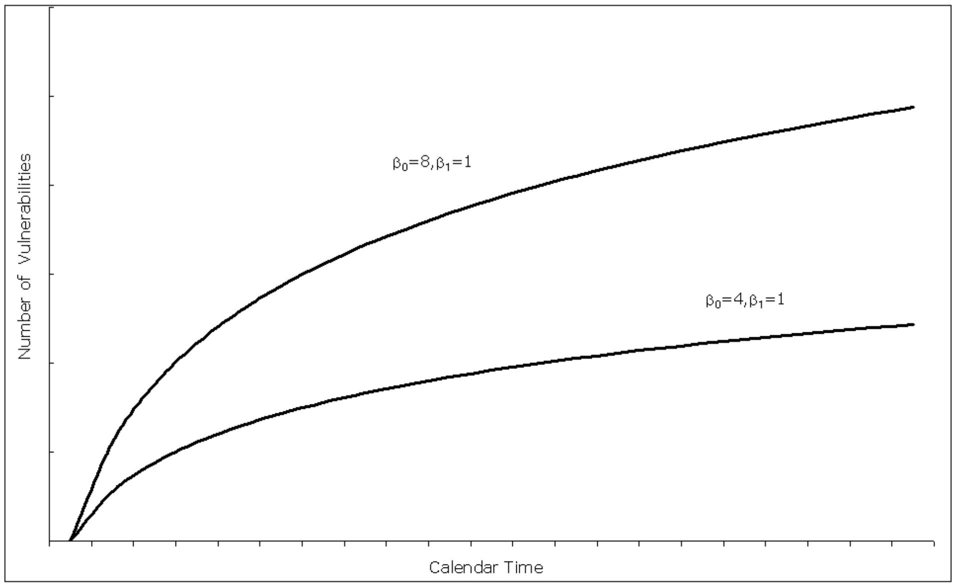

Figure 4.1: Musa-Okumoto Logarithmic VDM

In Figure 4.1, the X axis is calendar time and the Y axis indicates the cumulative number of discovered vulnerability. A characteristic of Musa's logarithmic software reliability growth model, presented in Figure 4.1, is that software defects are not found immediately after release of software, and its vulnerability discovery rate has a saturation phase as time passes.

4.1.2 Rescorla's Quadratic and Exponential VDM

Rescorla has suggested two software vulnerability discovery models [46]. These model have an opposing vulnerability discovery hypothesis. This subsection discusses his two software vulnerability discovery models.

4.1.2.1 Rescorla's Quadratic VDM

This vulnerability discovery model assumes that the software vulnerability discovery rate is constant. As a result, its cumulative vulnerability discovery model has an quadratic growth shape. The steps for the final model is represented in Equation 2.

$$\omega(t) = Bt + k$$

$$\Omega(t) = \int(Bt + K)dt = \frac{Bt^2}{2} + Kt \tag{2}$$

In Equation 2, the first step is an assumption of vulnerability discovery rate, and is a linear model. The cumulative vulnerability discovery in the second step shows time quadratic increasing rates. The characteristic of this model is that Ω grows, as given by the squared term in Equation 2. With the mathmatical stepping, we can get the modeling plot in Figure 4.2.

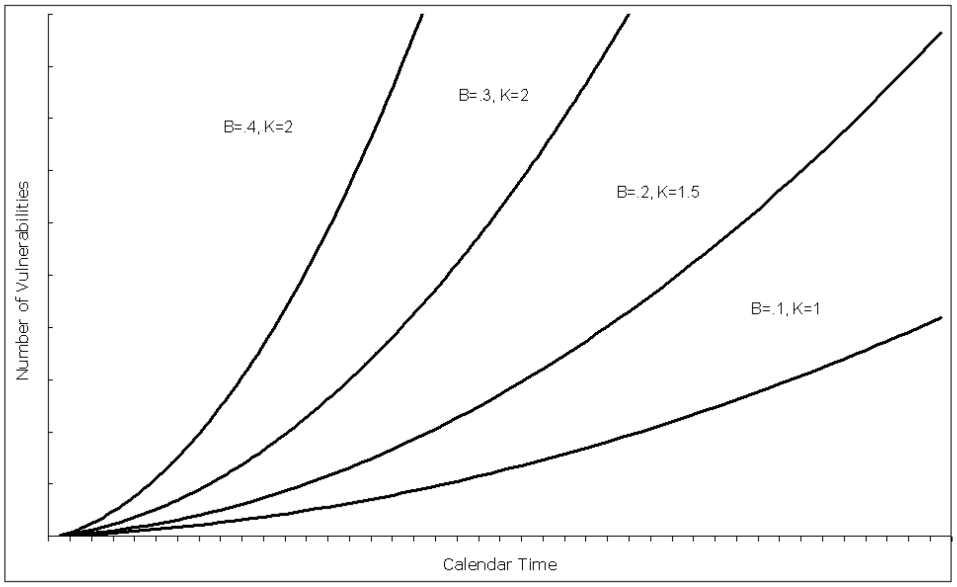

Figure 4.2: Rescorla Quadratic VDM

This vulnerability discovery modeling does not consider the end of the software life-cycle, however, it is valuable while the vulnerability discovery rate is linearly increasing. However, this is a good explanation for some open source software development processes which allow participation from any software project team or individuals. Caused by unplanned software evolution, the vulnerability discovery in Linux displays a fast growth rate. The limitation of this vulnerability discovery modeling is that it can only be applied to the running phase, which shows rapid increase in discovery rate. The example for the linux vulnerability discovery trend is represented in Figure 4.3.

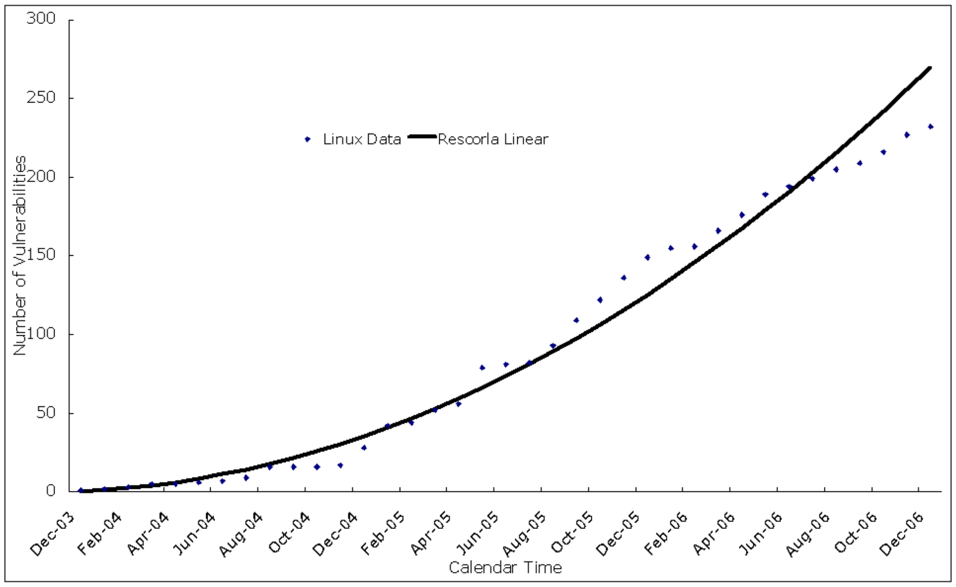

Figure 4.3: Quadratic VDM on Linux

In Figure 4.3, the vulnerability discovery trend seems that it is intuitively fit to model. However, these model fitting shows insignificant fit. Therefore, we can use this one to intuitive analysis.

	B	K	P-value	χ^2	χ^2critical	Result
Quadratic VDM	0.00014	0.0453	7.56E-06	149.601	105.267	Insignificant

Table 4.1: Rescorla Quadratic VDM Fitting Result

4.1.2.2 Rescorla's Exponential VDM

This model assumes that the discovery rate is initially grows after release, it eventually reaches a saturation phase. The modeling is presented in Equation 3.

$$\Omega(t) = \int N\lambda e^{-\lambda t} dt = N - Ne^{-\lambda t} = N(1 - e^{-\lambda t}) \qquad (3)$$

In the equation, the exponential model assumes that the vulnerability discovery rate is exponentially decreasing, the parameter λ is the shape parameter, and the N is the scale parameter in the equations. In the second step, the vulnerability discovery rate shows a exponential cumultative vulnerability number. This vulnerability discovery model is represented in Figure 4.4.

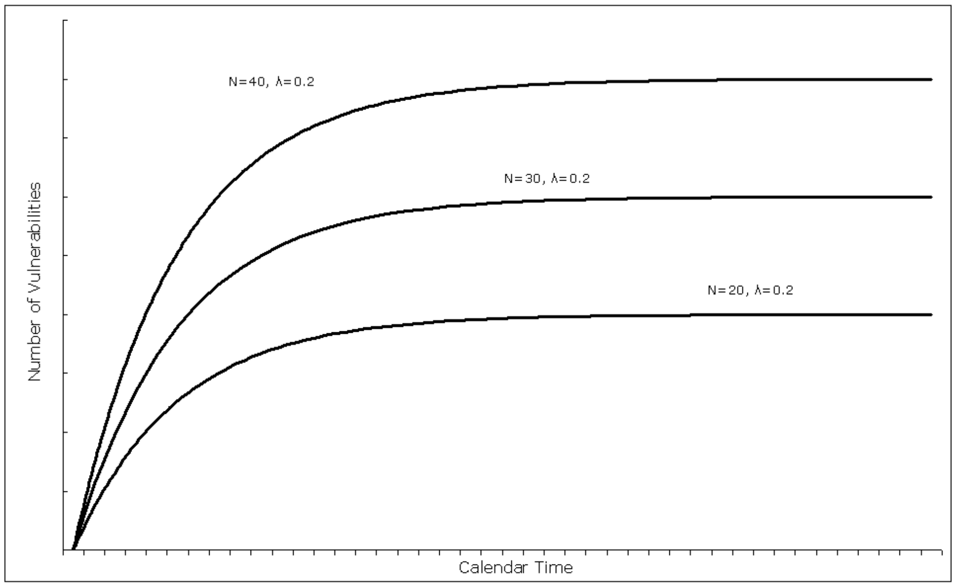

Figure 4.4: Rescorla Exponential VDM

The similarity between linear and exponential modeling is that the software vulnerability discovery rate is increasing, however, the models have different vulnerability discovery rates that depend on time.

4.1.3 Anderson's Thermodynamic VDM

Anderson's Thermodynamic Vulnerability Discovery Model [7] is given by Equation 4.

$$\omega(t) = \frac{k}{\gamma t}$$

$$\Omega(t) = \int(\tfrac{k}{\gamma t})dt = \tfrac{k}{\gamma}\ln(t) + \tfrac{k}{\gamma}\ln(C) \tag{4}$$

In Equation 4, the $\frac{k}{\gamma}\ln(C)$ represents the integration constant. This modeling equation has a shape similar to Musa's model. The difference, compared with Musa's model, is that the vulnerability discovery rate shows a rapid growth rate in the initial phase. Characteristic of this model is that discovered vulnerabilities do not come out right after release, and the software vulnerability discovery rate is highest post release. However, as time passes its vulnerability rate decreases. This discovery model is represented in Figure 4.5.

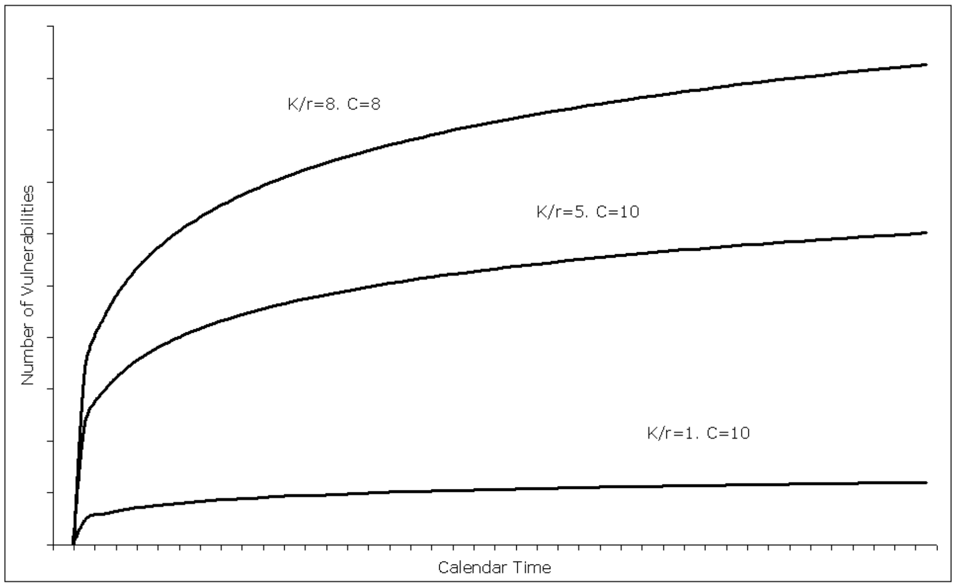

Figure 4.5: Anderson's Thermodynamic VDM

4.1.4 Alhazmi-Malaiya Logistic VDM

The basic shape of the Alhazmi-Malaiya Logistic (AML) model is illustrated in Figure 4.6. 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 $\Omega(t)$ in Equation 5.

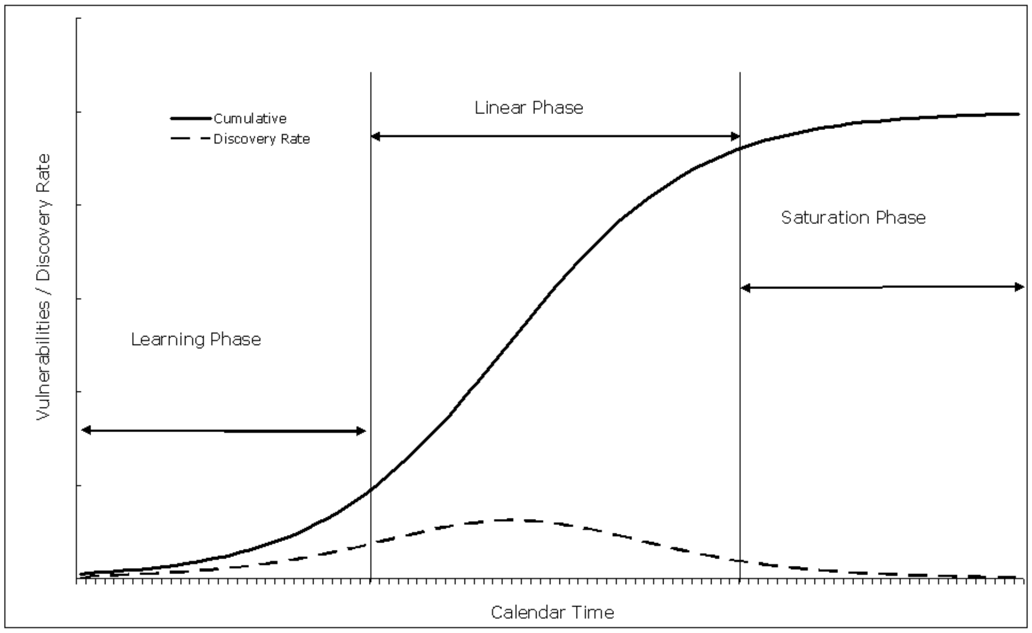

Figure 4.6: AML Vulnerability Discovery Model

where B represents the estimated total number of vulnerabilities and the parameters A and C determine the shape of the curve [3]. 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 [5]. This model has been found to yield a significant goodness-of-fit for many widely used software systems [3, 2, 5, 6]. However the plots of actual data sometimes show a departure from the model following the release of a new version [2].

$$\Omega(t) = \frac{B}{BCe^{-ABt}+1} \tag{5}$$

This model has been formed to fit almost every type of real data [3, 4, 6, 57]. However, it has a limitation. The data often shows a superposition effect due to an unexpected software vulnerability discovery rate in previous single vulnerability discovery modeling, and it is derived from several factors such as source code modification, user increase, environment changes, new software release and others. In figure 4.7, the irregular vulnerability discovery rate and its modeling is illustrated in one famous software system, Internet Explorer version 3.

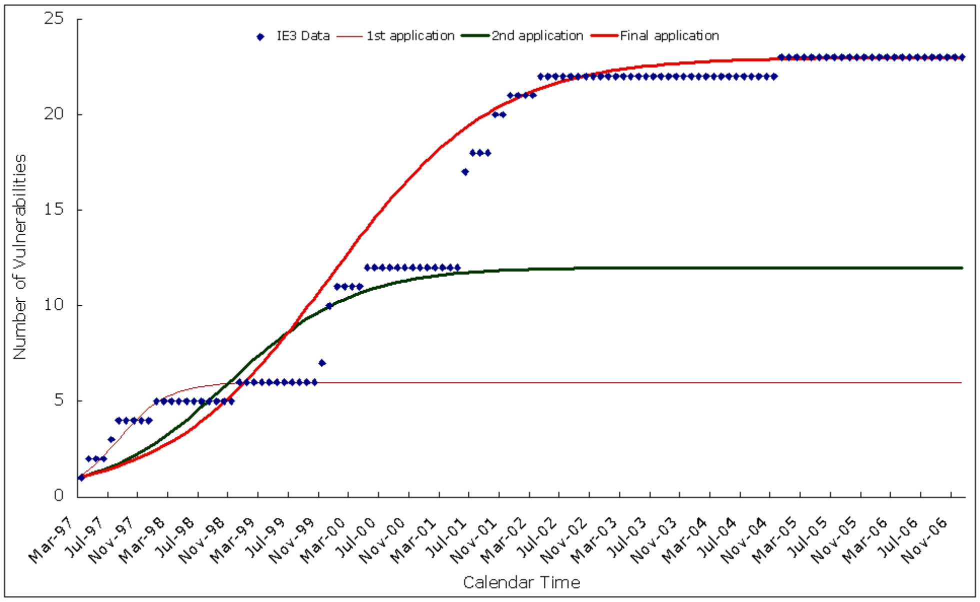

Figure 4.7: Internet Explorer 3 AML VDM Superposition Effect

To explain the superposition effect caused by shared source code and other functionality, we will propose a new multiple version software vulnerability discovery model in the next section. The theory behind the multiple version software vulnerability discovery model is that shared source code between two versions of software has a direct impact on the softwre vulnerability discovery rate.

4.2 Modeling Vulnerability Discovery in Multiple Version Software

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.

4.2.1 Need for a Multiple Version Software VDM

The discovery models mentioned in the previous section simply focus on independent or a single-version software. Some of these models show a responsible goodness-of-fit for almost all software systems [3, 4, 2]. However, the models do not explain an abnormal increases into vulnerability discovery rate in an initial version software system because of a new software version release. This anamoly led us to investigate a multiple version software vulnerability discovery model (MVDM). Nowadays almost all programs have several upgraded versions because of growing user and vendor requirements. As a result, multiple version software is being modified and used simultaneously. The later versions of software are expected to software reliability and functionality better than previous versions of software systems. This gives rise to a vulnerability discovery trend different from single version VDM, since the software design is changed or new code is added. We assume that shared functionality and shared code from previous versions of software may be tested during usage, though the previous version is not in heavy use anymore. Also, during later version software tests, undiscovered shared vulnerability in previous software is tested again. With this hypothesis, we made another advanced software vulnerability discovery model between two versions of software's vulnerability discovery rates. This assumption can explain the superposition effect, which is an atypical vulnerability discovery increasing rate between two versions of software vulnerability discovery rates.

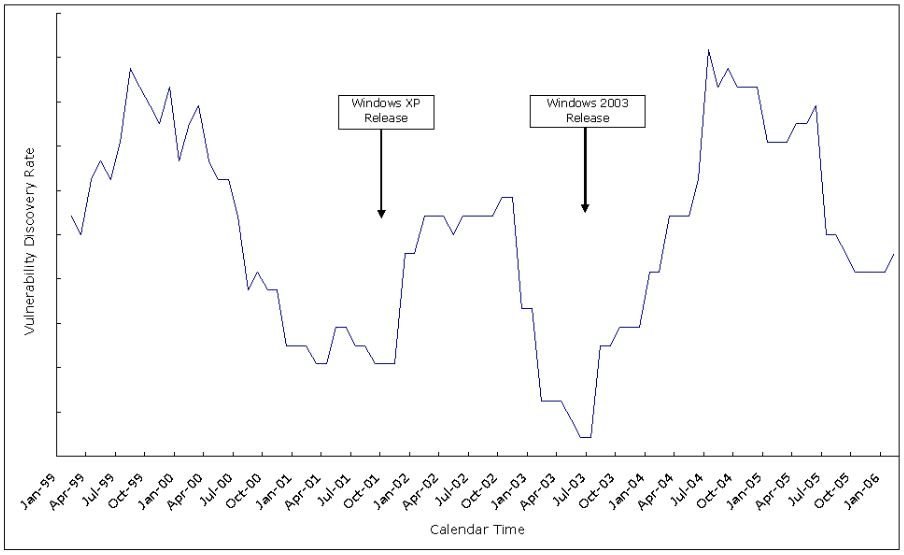

Figure 4.8: Windows 98 Vulnerability Discovery Rate using Moving Average

Figure 4.8 presents the real vulnerability discovery rate of Windows 98 using 12 months' moving average. In the result, we can find the vulnerability discovery rate is changed after next version software releases, and previous single vulnerability discovery models need to be considered. For advanced modeling, we change the AML model for multiple version software. This multiple version software vulnerability discovery modeling idea is presented in the next section.

4.2.2 Proposed Multiple Version Software VDM

The multiple software vulnerability discovery model in this section is based on the AML single software vulnerability discovery modeling, since it shows goodness-of-fit for almost all software systems. For multiple version software vulnerability discovery modeling, for estimating initial version vulnerability discovery trend accurately number, we will use the shared vulnerability in later version software. In addition, we assume that shared vulnerabilities arise because of shared source code and functions. This idea can be shown in several ways depending on the scenario, therefore, the next paragraph illustrates several scenarios of software development, release date and vulnerability discovery.

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 4.9, 4.10 and 4.11.

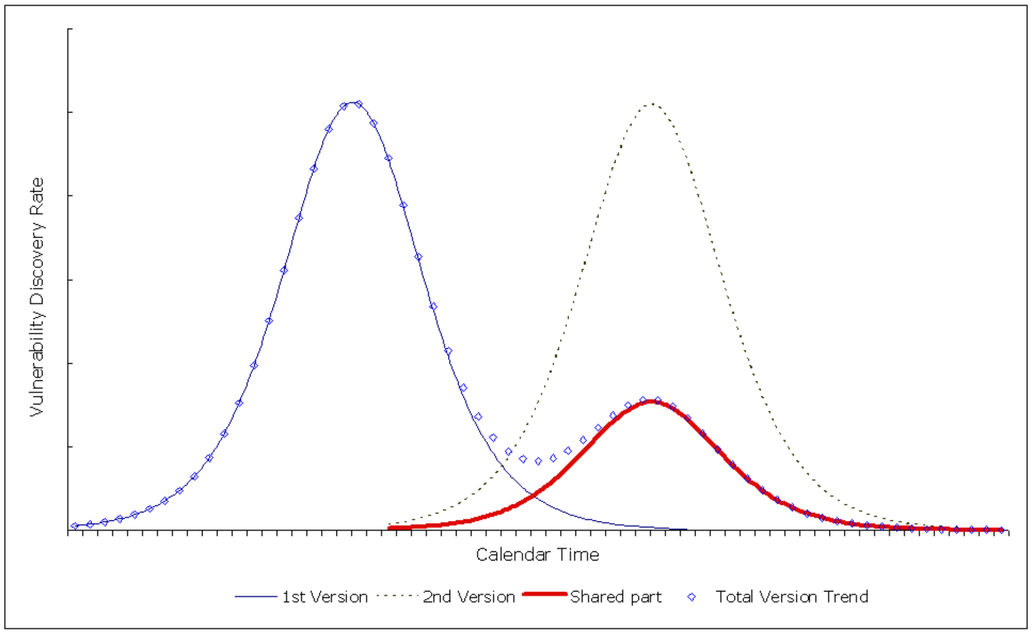

Figure 4.9: Basic MVDM Discovery Rate

In Figure 4.9, the X axis indicates calendar time and the Y axis is the vulnerability discovery rate. The first peak in Figure 4.9 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 4.9 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.

For determining the previous vulnerability discovery rate, the vulnerability discovery rate should be summated. The cumulative vulnerability discovery trend in previous version and later version software is also illustrated in Figure 4.10 and 4.11. In Figure 4.10, the first version software vulnerability discovery model shows a saturation phase, however, due to the shared vulnerabilities discovered in the second version of software, the vulnerability discovery rate is able to be shown as a distorted Logistic graph.

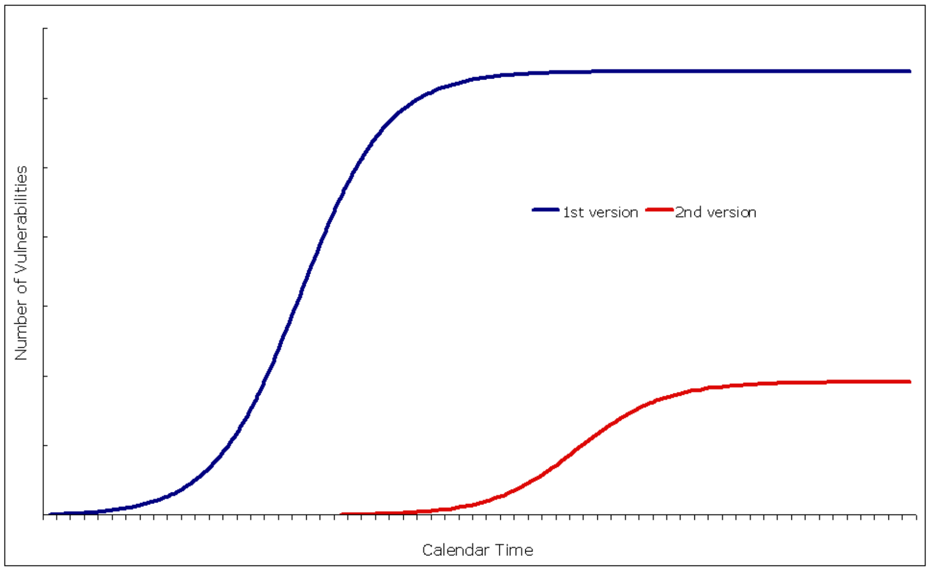

Figure 4.10: Separated Vulnerability Discovery Trend in Basic MVDM

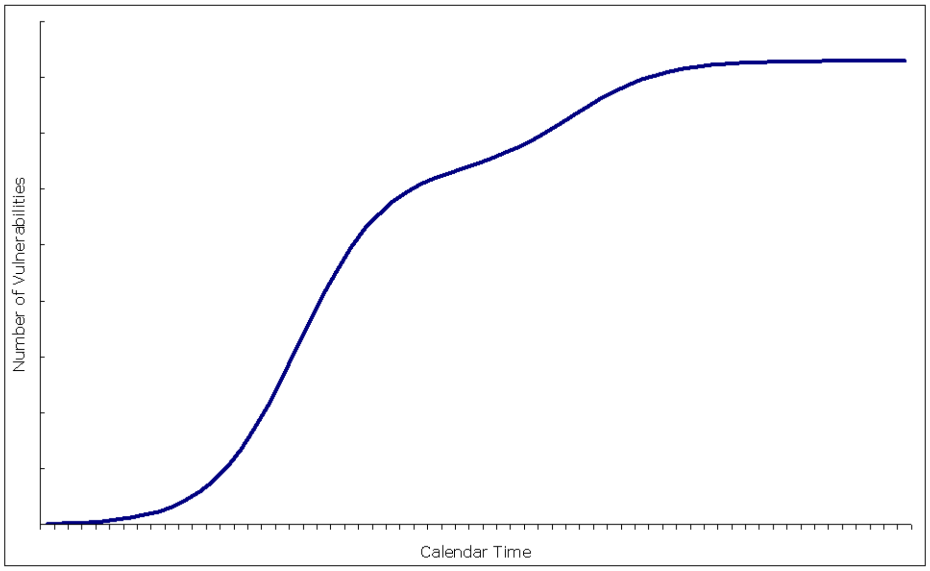

Figure 4.11: Basic MVDM

This result is illustrated in Figure 4.11, and the cumulative software vulnerability discovery model described shows how Logistic model still fits for multiple versions of software.

Using real data of vulnerability discovery, this shape might not be fitted significantly. However, the contribution of this modeling is that abnormal software vulnerability discovery rate can be explained. This modeling is one example, and the other example will be simulated in the following section.

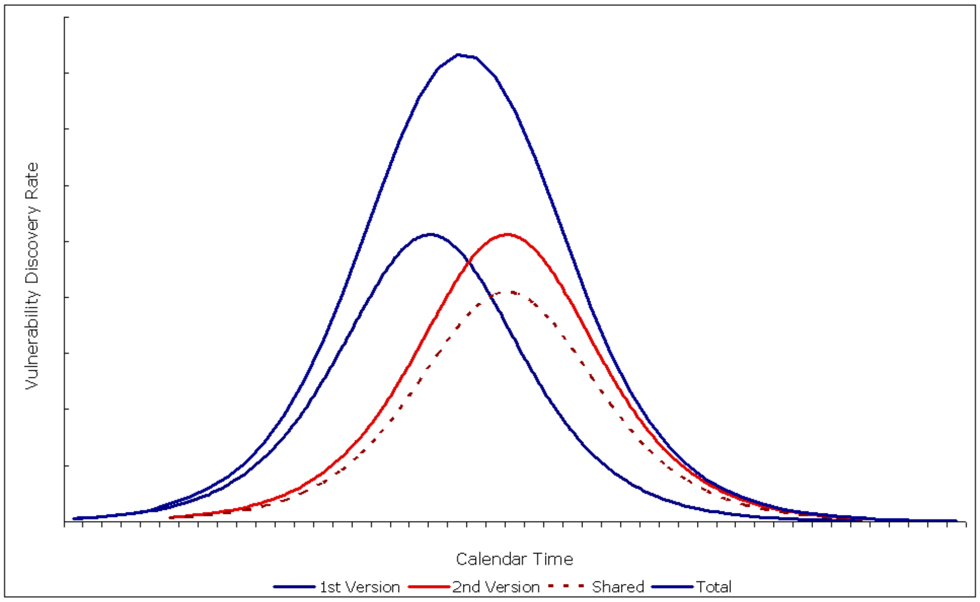

Figure 4.12: One-Hump MVDM Discovery Rate

Figure 4.12 presents the second scenario of the multiple version software vulnerability discovery model, where the release date on each version of software is very close, and its shared vulnerability discovery rate is large. We simulated a model which has 80% vulnerability sharing in later version software, and the time gap is not enough for the previous version's software vulnerability discovery rate to decrease significantly. Thus the following software version's vulnerability discovery rate is not far from the peak of the previous version.In figure 4.12, the largest curve indicates summated vulnerability discovery rate, and the two smaller curves represent two versions of software's vulnerability discovery rate, and the smallest line is the shared vulnerabilities in the second software system.

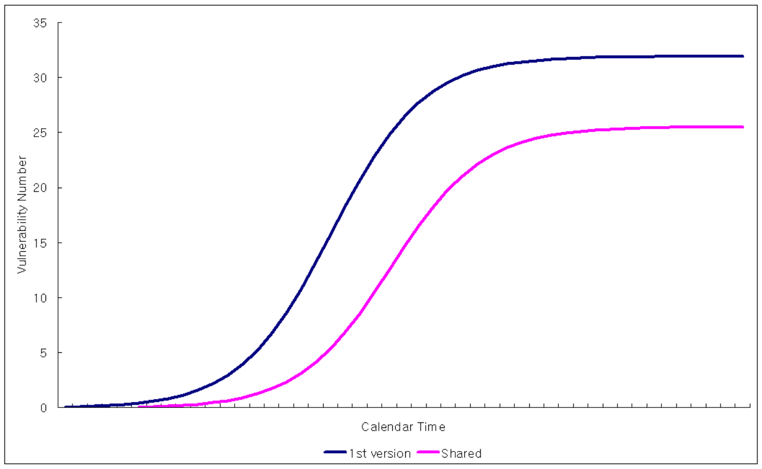

Figure 4.13: Separated Vulnerability Discovery Trend in One-Hump MVDM

The vulnerability discovery trend derived from Figure 4.12 is represented in Figure 4.13. These trend shows how the software which is released within samll time gap. The total vulnerability discovery rate causes a hump as shown in Figure 4.14. Usually, this trend comes from dependent versions which have service packs like those for windows and IE. These vulnerability discovery rates show gradual increasing and decreasing, and can be substituted with weibull vulnerability discovery modeling.

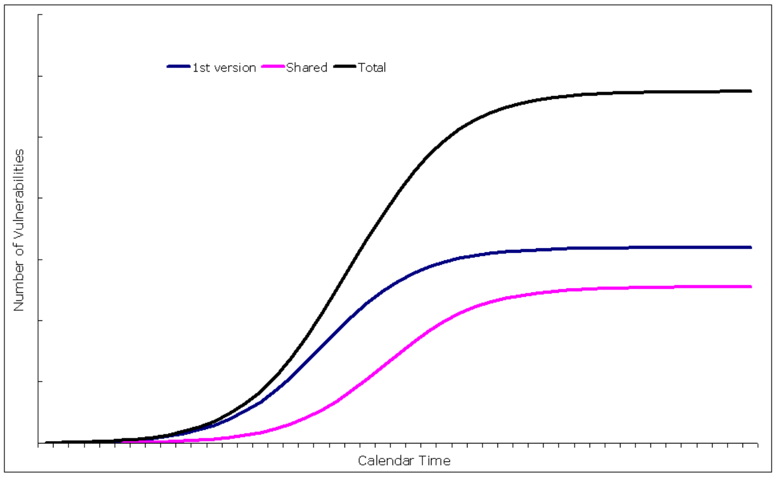

Figure 4.14: One-Hump MVDM Trend

The strong point of this modeling is that the software vulnerability discovery rate can be substituted with single software vulnerabiiltiy discovery modeling. However, expectation or prediction of a middle point can not be determined by multiple versions of software. Therefore, the other abnormal vulnerability discovery rate in one software system can be explained with this multiple modeling.

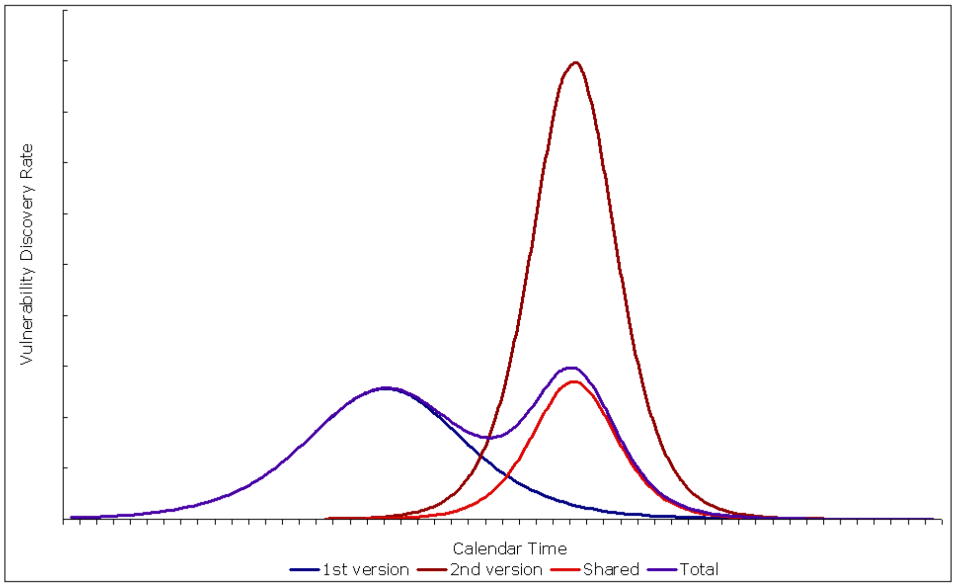

Figure 4.15: Two-Hump MVDM Discovery Rate

The last scenario of the multiple version software vulnerability discovery model, is presented in Figure 4.15. The hypothesis of Figure 4.15 is that the release date on each version of software is not close, but its shared vulnerability discovery rate in later version software is large. In Figure 4.15, the summated vulnerability discovery rate shows different discovery modeling.

This third scenario has two significant vulnerability discovery rate models. We call this modeling the Two-humped vulnerability discovery model, since it has two significant vulnerability discovery rates. The separated and combined vulnerability discovery trend is presented in Figure 4.16 and 4.17.

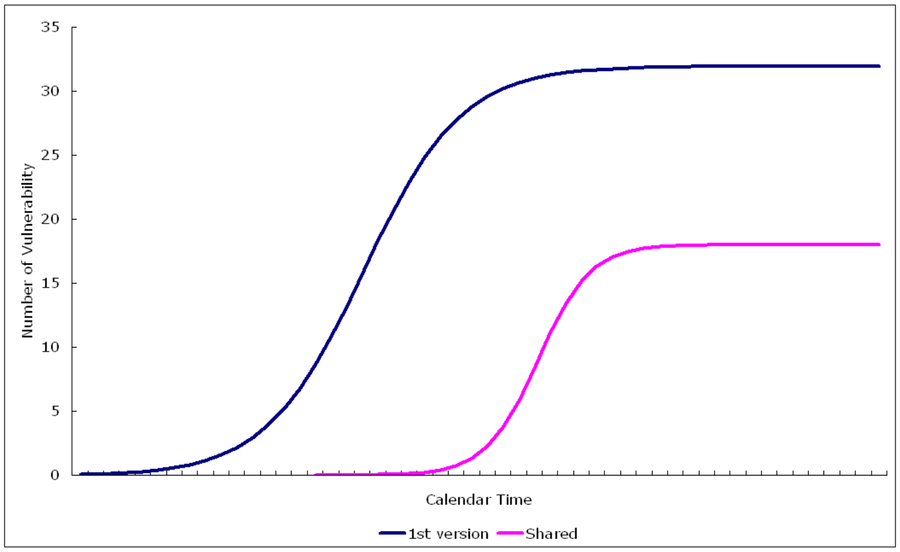

Figure 4.16: Separated Vulnerability Discovery Trend in Two-Hump MVDM

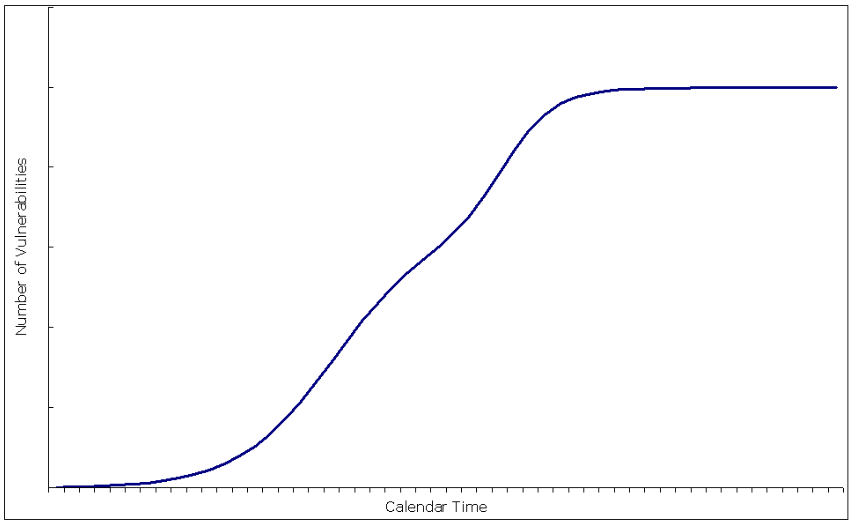

Figure 4.17: Two-Hump MVDM Trend

The third scenario of multiple version software vulnerability discovery modeling fits to a special case software system, which has a small number of users and usage time for the first release, and the system has an increased number of users and testing time with software source growing at the next release, such as the Mysql 3.2 version and 4.0 version vulnerability discovery trend. This is a very rare case, because the user environment cannot be changed drastically, and software developers tend not to change their software a lot. In the case of Mysql, porting the software, from Linux to Windows, creates a large change and can potentially absorb many new users. In this hypothesis, we have another problem. Since we did not consider a vulnerability growth model which has a decreasing vulnerability rate in one software, this vulnerability discovery rate should be reconsidered. However, this vulnerability discovery is one good explanation for the single version software vulnerability discovery trend. These model is presented in Equation 6.

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 $\Omega(t)$ for some given software with multiple versions is given by an addition of two terms.

$$\Omega(t) = \frac{B}{BCe^{-ABt}+1} + \alpha\frac{B'}{B'C'e^{-A'B'(t-\epsilon)}+1} \tag{6}$$

In Equation 6, the first formula indicated first version VDM and second formula is vulnerability discovery from the second version. 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 6 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 Figure 4.18 and Equation 7. These algorithms are very complex to represent each version of software. The hypothesis of Figure 4.18 is that the release date on each version of software is irregular and the vulnerability discovery rate is also different. However, each version reused initial version source code.

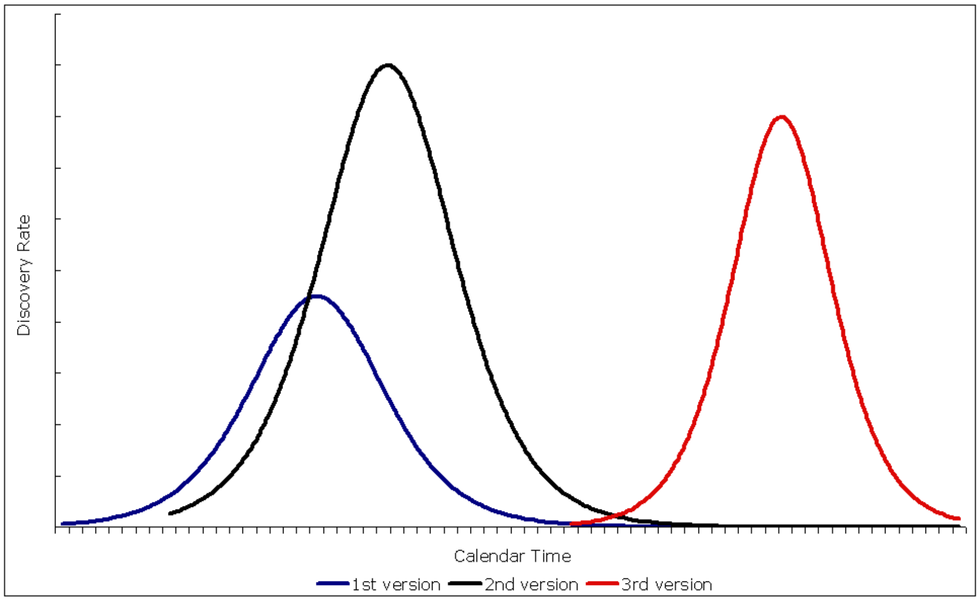

Figure 4.18: Entire Version MVDM Discovery Rate

$$\Omega(t) = \sum \alpha i \frac{Bi}{BiCie^{-AiBi(t-\epsilon i)}+1} \tag{7}$$

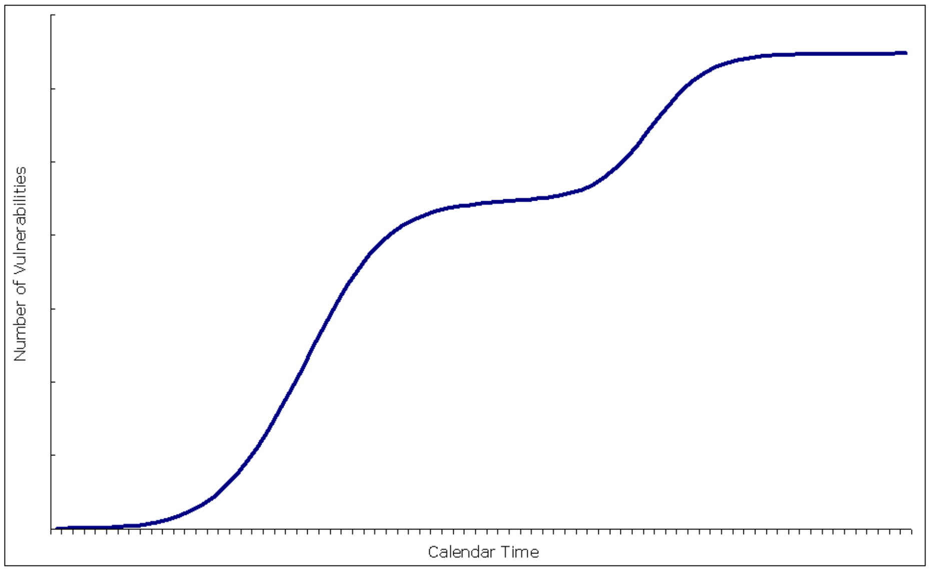

Figure 4.19: Entire Version MVDM Trend

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. The cumulative software vulnerability discovery rate is represented in Figure 4.19.

Similar to Two-humped vulnerability discovery rate, the cumulative vulnerability discovery trend in multiple version software systems has a saturation phase, and it shows why Logistic modeling is fitting to total version vulnerability discovery modeling. However, a considerable question is the relation between the source code evolution and software reliability growth. In this paper, we will not discuss that topic. Therefore, the cumulative vulnerability discovery trend can be plotted in the figure.

The weakness of this modeling is that we consider shared code, not function and modularity. This consideration causes gaps in software reliability growth modeling. Software reliability growth modeling for stable software systems may not fit to this modeling scheme. This model assumes that requirement changes, and subsequently changes to the code, will cause the model to rise for a short period time before reaching another saturation phase.

4.3 An Alternative Software Vulnerability Discovery Model

In this section, we will propose an alternative software vulnerability discovery models.

4.3.1 Asymmetric VDM

The AML software vulnerability discovery model assumes a symmetrical vulnerability discovery trend. It is based on the assumption that the rise and fall in software usage is symmetric. However, software usage may often not be symmetric because many users continue to use the system for a long time even when a new version has been released.

4.3.1.1 Weibull VDM

The use of a Weibull distribution is very common reliability and quality control work [39]. The distribution is well suited to situations where the conditions of 'strict randomness' of the exponential distribution are not satisfied. Some software vulnerability discovery trends can be illustrated as Weibull distributions, and it is easy to apply a Weibull distribution to some software systems. The assumption when using a Weibull distribution is that the software system discovery rate is not a symetric model, and its discovery rate remains high for a long period of time following a high peak software vulnerability discovery model. In Figure 4.20, this assumption is illustrated.

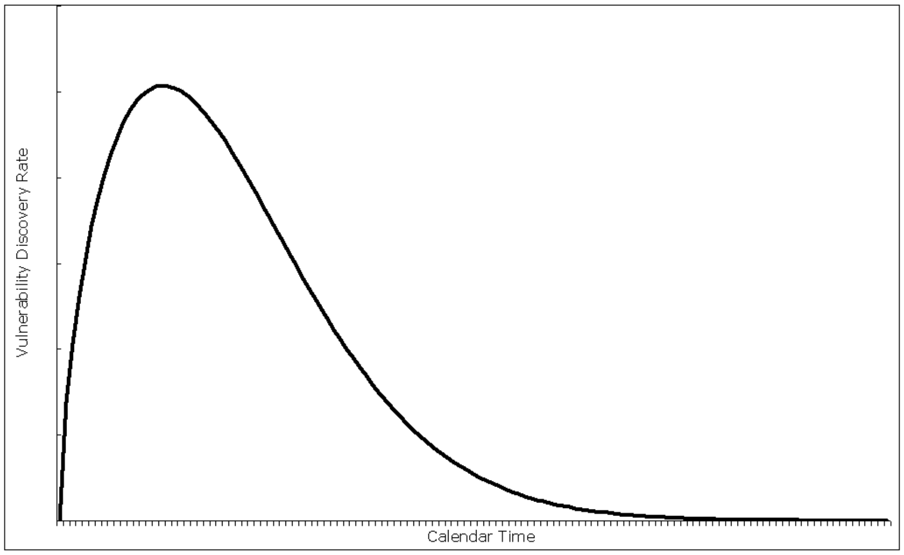

Figure 4.20: Vulnerability Discovery Rate according to Weibull VDM

These assumption is presented in Equation 8.

$$\omega(t) = \gamma\{\tfrac{\alpha}{\beta}(\tfrac{t}{\beta})^{\beta-1}\exp^{-(\frac{t}{\beta})^{\beta}}\} \tag{8}$$

In Equation 8, the α is the shape parameter which can determine the shape of the software vulnerability discovery rate. The parameter β is the scale parameter, which can stretch the time duration of software vulnerability discovery modeling. The γ value is number of estimated software vulnerabilities. Finally, the t value is real calendar time. These vulnerability discovery rate parameters are used to determine the vulnerability discovery trend.

The Weibull vulnerability discovery model assumes that software vulnerability discoveries increase, and it reaches its highest rate relatively rapidly and after that the vulnerability discovery rate goes down gradually. The vulnerability discovery model shows a similar graph with Logistic model, but the weight of this concept is useful to explain software usage and its vulnerability discovery rate. An additional advantage of this modeling is that we can get any flexible asymetrical software vulnerability discovery model by changing the alpha value. Weibull's cumulative discovery model is presented in Figure 4.21.

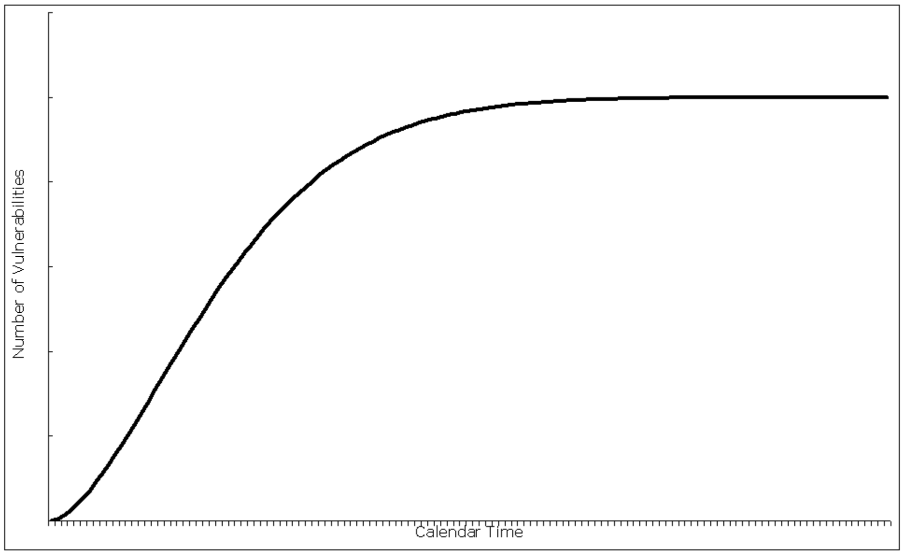

Figure 4.21: Weibull VDM Trend

$$\Omega(t) = \gamma\{1 - \exp^{-(\frac{t}{\beta})^{\alpha}}\} \tag{9}$$

Vulnerability discovery modeling using Weibull distribution shows a similar pattern graph to Logistic modeling. The most important idea of Weibull distributions is that the vulnerability discovery rate of software is not necessarily symetric, and can be modified. This idea might derive from Folded distribution, and it is illustrated in next subsection.

4.3.1.2 Folded VDM

An example of asymetric vulnerability discovery modeling is represented in the previous subsection. This section discusses the cause of the asymetrical vulnerability discovery rate. Software vulnerability discovery models sometimes shows an abnormal fitting caused by testing. Although software vulnerability discovery modeling assumes logistic modeling, because of untested part before release, it show an asymetrical fitting result. We cannot separate the vendor tested vulnerability and user tested vulnerability, so this theory is presented in this section. The theoretical vulnerability discovery model is presented in Equation 10 and Figure 4.22.

$$\omega(t) = \frac{1}{\sqrt{2\pi}\sigma}[\exp^{-(t-\varepsilon)^2/2\sigma^2} + \exp^{-(t+\varepsilon)^2/2\sigma^2}], t \geq 0 \tag{10}$$

This vulnerability discovery modeling can be plotted as in Figure 4.22. The core of this modeling is the testing results before release. Typical software testing is completed before release, however, some of them are not completed so the remaining software testing is achieved after release. This vulnerability discovery model is theretical background of asymmetrical vulnerability discovery trend in symmetrical discovery modeling. This folded vulnerability discovery modeling show how remaining vulnerability discovery rate can be changed by remaining software vulnerability discovery model. However, since the vulnerability discovery rate coming from the testing case differs from the normal vulnerability discovery rate, the initial phase can differ from a normal symmetrical logistic modeling shape.

The cumulative folded vulnerability discovery rate is illustrated in Figure 4.23. This vulnerability discovery modeling has short term learning phase and the missing learning phase make the normal distribution asymmetric.

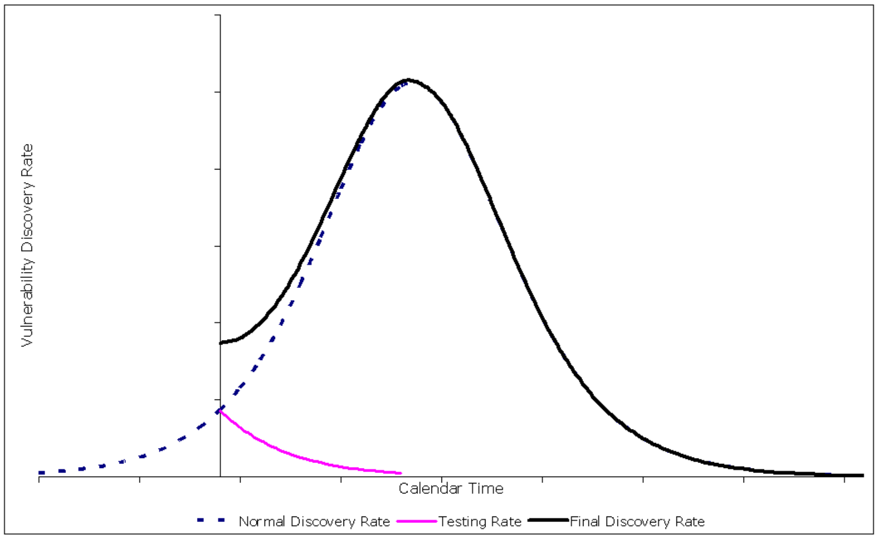

Figure 4.22: Folded VDM Discovery Rate

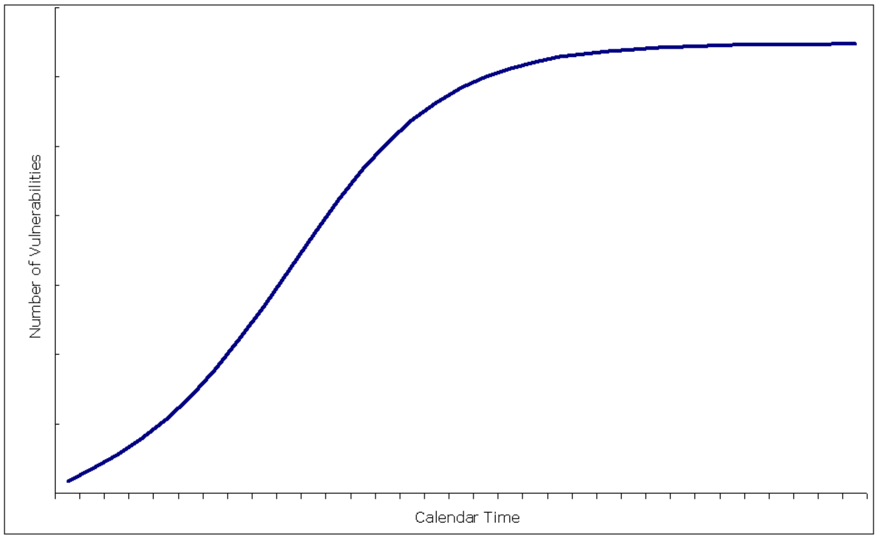

Figure 4.23: Folded VDM Trend

This asymmetrical model is similar to the Weibull vulnerability discovery model. The applicability of these models need to be demonstrated using real data, therefore, in the following chapter, we will examine the asymmetrical model using the Weibull vulnerability discovery model.

Chapter 5

Vulnerability Discovery in Open Source Software

5.1 Multiple Version Software Vulnerability Discovery Modeling for Open Source Programs

To identify the best software vulnerability discovery modeling approach for open source software, we use the multiple version software vulnerability discovery model (MVDM), and alternative models, as described in Chapter 4. For evaluating applicability of MVDM, we use vulnerability data for major versions of Apache HTTP Web server and Mysql DBMS. This decision was made because Apache HTTP Web server (58% Web Server market share in the World [40]) and Mysql DBMS (29% DBMS market share in the World [38]) are among the top web servers and DBMSs, respectively, currently in use by vendors and users. Additionally, they are being maintained by a dedicated project team. High market share and a consistent project team means that the requirements of most users are reflected in the software. As a result, the programs are being evaluated and modified by users constantly and continually. Therefore, these software systems are a good example of open source systems with multiple versions.

5.1.1 Methodology for Multiple Version Software Vulnerability Discovery

5.1.1.1 Software Source Code Analysis

For evaluating MVDM, we need to identify the fraction of the code share between two successive versions of software. Before checking the shared software source code, we need to identify the software source code distribution to determine which source code is used for core coding. We used SLOCcounter to determine the distribution [51]. The observed software source code pattern is presented in Table 5.1.

	Version 1.3	Version 1.3.37	Version 4.0.0	Version 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 5.1: Source Code Distribution for Major Software System

In Table 5.1, we can see which kinds of source code are needed for examination. Perl is a scripting language for CGI. Therefore, its source code will be ignored in this research. For determining source code sharing we stripped the comments from the source code and, using the Diff command in a Linux or Unix machine, examined the shared source code. The research method, for calculating the amount of shared source code, is to compare the most recently released version with the first release of the software.

The shared source code percentage between Apache 1.3 and 2.0 is displayed in Table 5.2. The first row indicates the percentage of lines of code remaining from the initial code, compared with the second iteration of software. The second row indicates how many lines of code are shared code out of all the code in the second version of software. We assume that the second column is the value of the alpha of the two version software vulnerability discovery modeling from Equation 6 in Chapter 3. In addition to Apache source code share, Mysql DBMS code share is also given in Table 5.2.

	Apache 1.3.24		Mysql 4.1.0
Apache 2.0.35	20.16%	Mysql 5.0.0	83.52%

Table 5.2: Inherited Percentage of Code From the Earlier Version

From Table 5.2, we can derive the shared parameter in Equation 6. First, using single Logistic modeling on pure vulnerability of first version software, determine the first A, B and C parameters. Second, using single Logistic modeling, determine the second A', B' and C' parameters. Then, using the shared source code percentage in Table 5.2, determine the shared software vulnerability model. Finally, using the pure vulnerability model and shared vulnerability model, which are already derived, get the final multiple version software vulnerability discovery model by summation.

5.1.1.2 Data Source for Software Vulnerabilities

In this thesis we use NVD as our vulnerability resource. The NVD data source is more varied than other software vulnerability database sources because its data is gathered by several vendors or individuals, and it is tested by the vulnerability management team of NVD. The data has been evaluated using the consistent vulnerability standard of the MITRE organization. Using this vulnerability data, we will examine our modeling.

5.1.1.3 Testing Goodness of Fit

For model verification with real data, we use P-value and Chi-square tests used for determining model fitting. The P-value test shows how the model fits to real data, and the result can be presented as a percentage. If the result from P-value tests is over 0.05, the model fitting is acceptable within 5% error possibility. The Chi-square test is a statistical hypothesis test, and it shows how well the hypothetical model fits to real data. The Chi-square test analysis is based on two kinds of data, one is the difference between the pair of observed and expected frequencies in each data. From the Chi-square test result, we should compare that result with the degrees of freedom (df) to get significant dataset. The degree of freedom (df) in this thesis is 0.05 probability, and this shows significant fitness for model testing.

5.1.2 Modeling Multiple Version Software Vulnerability Discovery

5.1.2.1 Vulnerability Discovery Modeling Approach

The process of Multiple version software vulnerability discovery modeling is similar to using AML modeling for pure vulnerability discovery trend of the first version of software, then examining the next software version using AML modeling. The examination period for the pure vulnerability discovery model is right after the learning time of the next version software vulnerability discovery trend. After deriving the A', B' and C' from the second version software using the sharing constant value, we examined the shared vulnerability discovery trend. Then we plotted the result for the multiple version software vulnerability discovery model through summation of the first version vulnerability discovery model and shared version vulnerability discovery modeling. The vulnerability discovery modeling test result is shown below.

5.1.2.2 Modeling Apache HTTP Web server Vulnerability Discovery

The testing result of the Apache HTTP Web server multiple version software modeling is presented in Figure 5.1. The X axis is the calendar time, and the Y axis indicates total number of vulnerabilities. Each mark on the graph indicates the cumulative number of vulnerabilities and the line illustrates the fitted AML modeling line. The representation of each point is explained in the Figure.

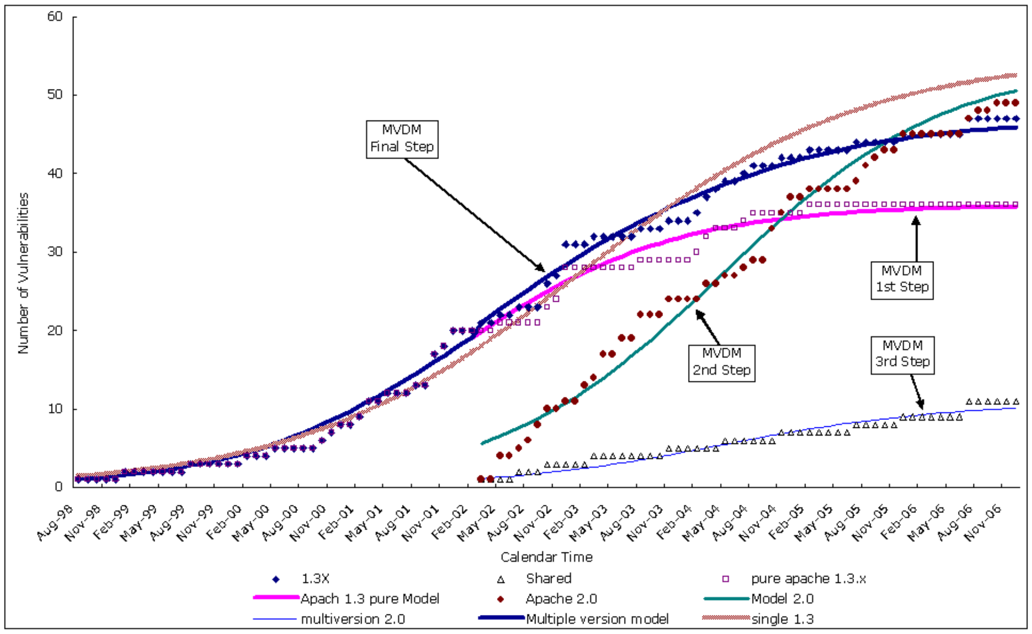

Figure 5.1: Apache Multiple Version VDM Fitting

In Figure 5.1, pure vulnerability of the first version of software shows a saturation phase in the software vulnerability life-cycle. However, when the shared vulnerability from the second version of Apache HTTP Server is added to the first version's vulnerability graph, the result is a continued increase in the vulnerability discovery rate overall. This is an example of the superpostion effect from [6], and this effect throws off predictions or estimates of the software vulnerability trend. However, in open source software, we can derive a structural analysis from its source code, the estimation of vulnerability discovery rate is more clear than in closed source software. Therefore, we will review this effect in the next Chapter. The tested parameters and fitting results are presented in Table 5.3.

	A	B	C	P-value	χ^2	χ^2critical	Result
Single AML Result	0.0012	54.939	0.701	1	27.79	125.46	Significant
MVDM 1st step	0.0024	36	1				
MVDM 2nd step	0.0015	54.207	0.214				
MVDM overall result				1	9.294	125.46	Significant

Table 5.3: Apache Multiple Version VDM Fitting Results

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 5.3.

In Table 5.3, 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.

5.1.2.3 Modeling Mysql DBMS Vulnerability Discovery

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.

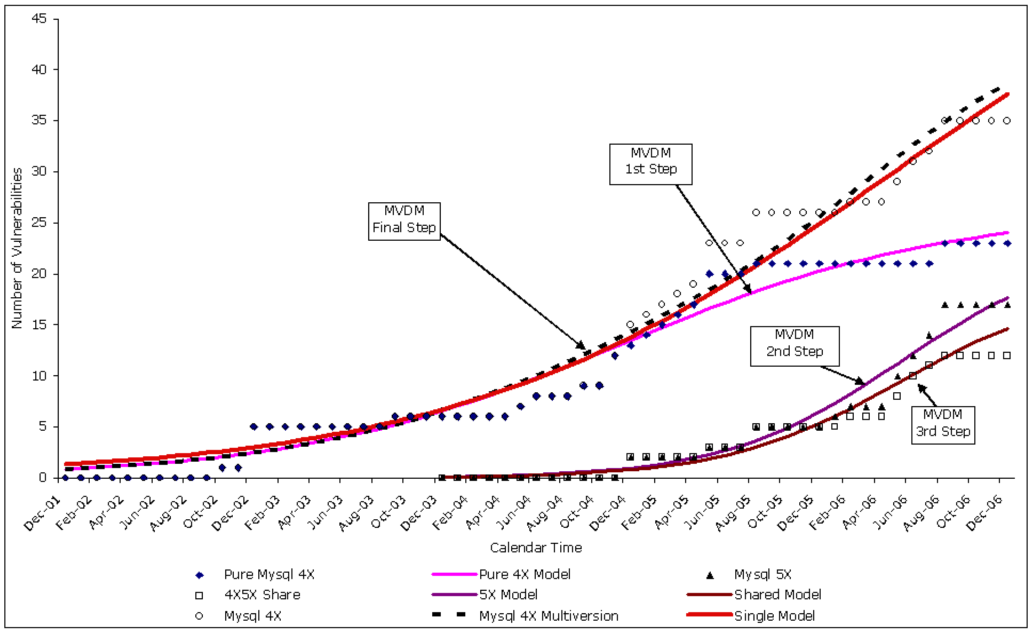

Figure 5.2: Mysql Multiple Version VDM Fitting

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^2critical	Result
Single AML Result	0.0012	60	0.8	0.99	37.12	80.232	Significant
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	Significant

Table 5.4: Mysql Multiple Version VDM Fitting Results

For the Mysql test, we can derive the parameter α from Table 5.4 because the second row value is the sharing percentage. In Table 5.4, the multiple version software vulnerability discovery modeling shows better goodness-of-fitness than single version software vulnerability discovery modeling through Chi-Square statistical test result. Additionally, we could determine the limitations of single version software vulnerability discovery modeling, which does not apply perfectly to multiversion software modeling. However, single Logistic vulnerability discovery modeling shows goodness of fit through the P-value test result.

This result shows that the multiple version software vulnerability discovery modeling has superior model fitting compared to single model fitting. The limitation of this idea is that more parameters are used for single vulnerability discovery modeling, and a model which has many parameters assures better model fitting than a model which has fewer parameters. However, these factors come from software shared code, and these approaches are related with software reliability growth, thus this modeling is meaningful for generalized software vulnerability discovery modeling.

5.2 Using Weibull and Logistic Software VDMs for Open Source Software

The Weibull software vulnerability discovery model has a short vulnerability growth phase compared with the entire lifecycle of the software as presented in the previous Chapter. That growth phase can be illustrated as the learning and running phase of Logistic vulnerability discovery modeling. In this section we will examine the software vulnerability discovery trend of Apache HTTP Web sever using AML and Weibull vulnerability discovery modeling for comparing the models.

5.2.1 Modeling Software Vulnerability Discovery for Apache HTTP Web server

The Apache Weibull software vulnerability discovery modeling result is presented in Figure 5.3. The X axis is the calendar time, and the Y axis indicates total number of vulnerabilities. The marked points of the graph indicates the cumulative vulnerability number using real data from Apache 1.3.x, and each line is the examined AML and Weibull modeling line.

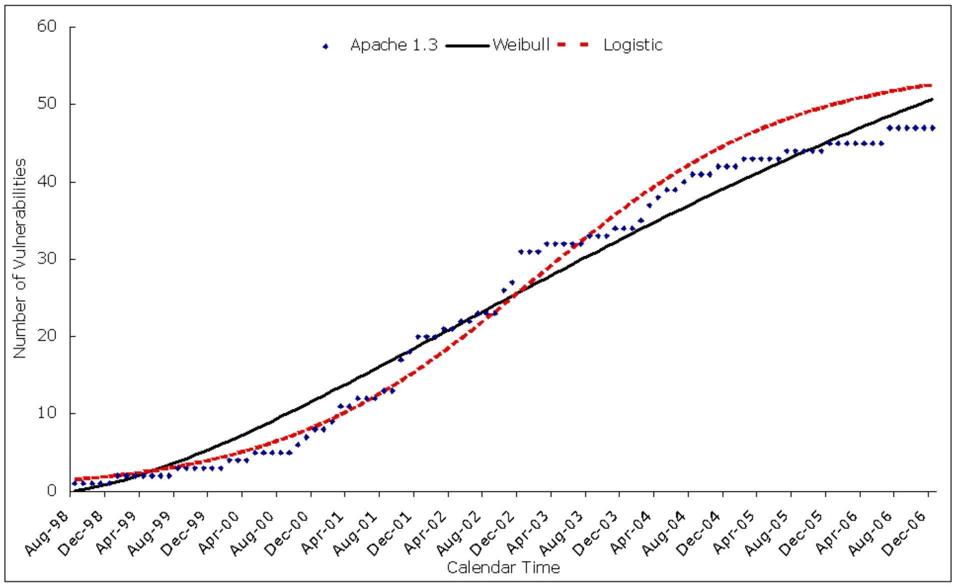

Figure 5.3: Apache 1.3 VDM Fitting

To determin the software vulnerability discovery rate, we calculate the vulnerability discovery trend. This vulnerability discovery modeling fitting result is represented in Table 5.5. This model fitting result shows how well these models are matching with real data, as we tend to expect. The modeling is an explanation for the future decreasing vulnerability discovery modeling, and this result from the Weibull vulnerability discovery model fits that trend.

	A(α)	B(β)	C(Υ)	P-value	χ^2	χ^2critical	Result
Logistic Model	0.0017	46.0001	0.9996	1	7.526	76.7778	significant
Weibull	77.8864	50.0231	77.8864	0.999	55.198	76.7778	significant

(), Weibull function value

Table 5.5: Apache 1.3 VDM Fitting Results

The software vulnerability discovery rate is presented in Figure 5.4 and Figure 5.5. The vulnerability discovery trends from the Weibull and Logistic model are presented in Figure 5.4 and Figure 5.5. From the results in Figure 5.4 and 5.5, we can derive Apache's vulnerability discovery rate. This result shows that the software life-cycle hypothesis in single and multiple version software can affect vulnerability discovery rate during the estimation process of software vulnerability discovery.

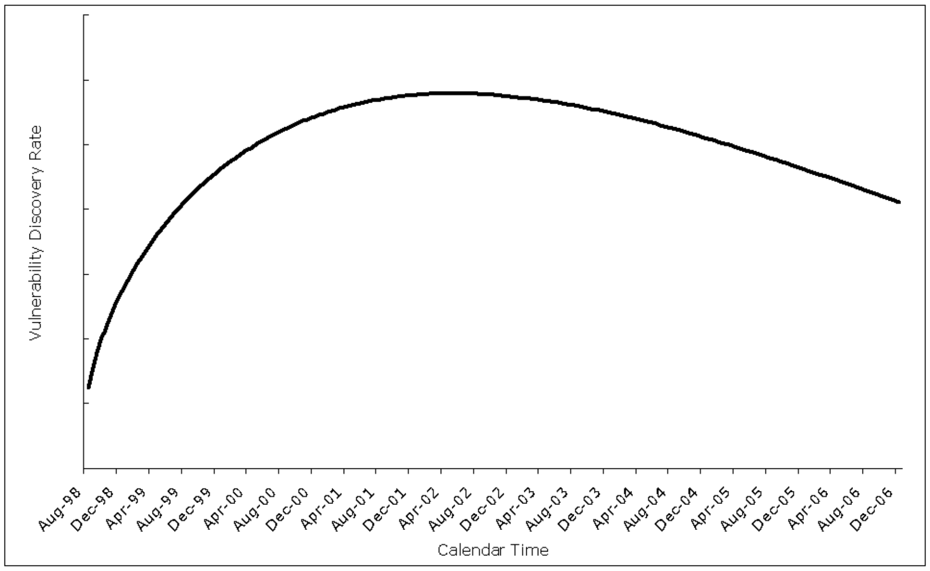

Figure 5.4: Apache 1.3 Weibull VDM Discovery Rate

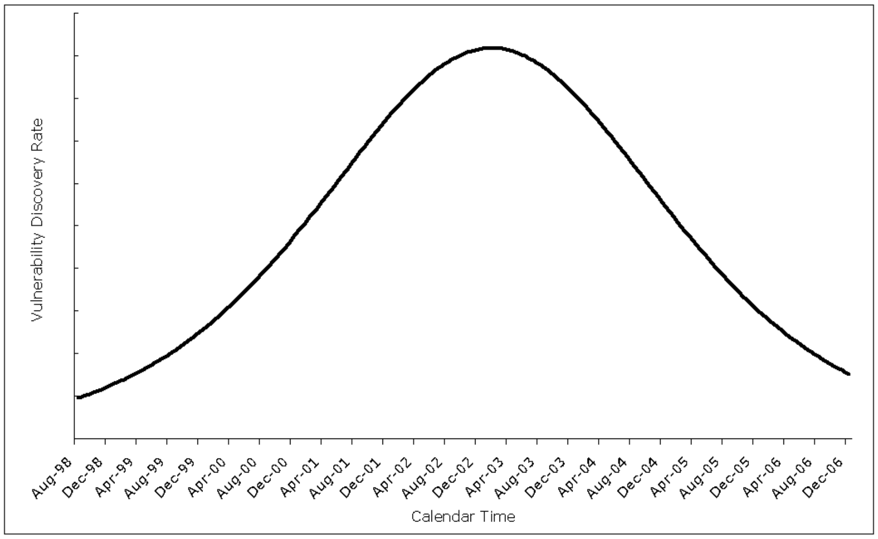

Figure 5.5: Apache 1.3 Logistic VDM Discovery Rate

5.2.2 Modeling Software Vulnerability Discovery for Mysql DBMS

For Weibull vulnerability discovery model testing, we use Mysql DBMS. The Vulnerability discovery modeling test results are in Figure 5.6.

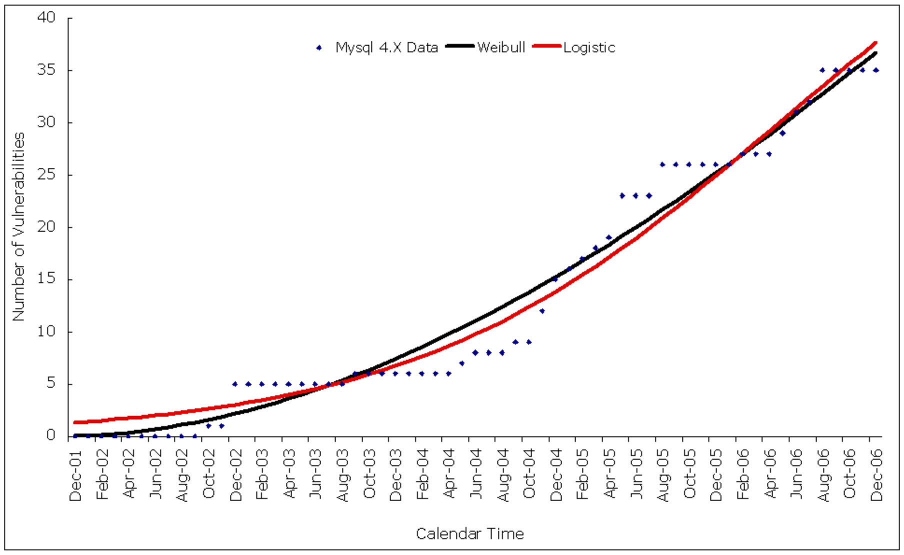

Figure 5.6: Mysql 4 Weibull VDM Fitting

In Figure 5.6, the vulnerability discovery modeling shows a similar shape to the learning and running phase in AML modeling. The testing result is presented in Table 5.6. The single Logistic model result is comes from the previous section's result date

	A(α)	B(β)	C(γ)	P-value	χ^2	χ^2critical	Result
Logistic Model	0.0019	45.357	1.193	0.999	90.557	80.232	working
Weibull	1.8938	17.427	183.635	0.999	31.532	80.232	significant

(), Weibull function value

Table 5.6: Mysql 4 VDM Fitting Results

In Table 5.6, both vulnerability discovery models fit to Mysql vulnerability discovery well, as indicated by the P value that signifies model fitting. However, the modeling result shows that the Weibull discovery show better fitness than the Logistic model by comparing Chi-square testing result. In this case, the Weibull distribution shows better goodness of fitness than Logistic vulnerability discovery, however, the vulnerability discovery rate shows an half bell shape, or exponential increasing rate from the 5.7. For verification of the software vulnerability discovery rate, we derived the software vulnerability discovery rate from model fitting. This result is presented in Figure 5.7.

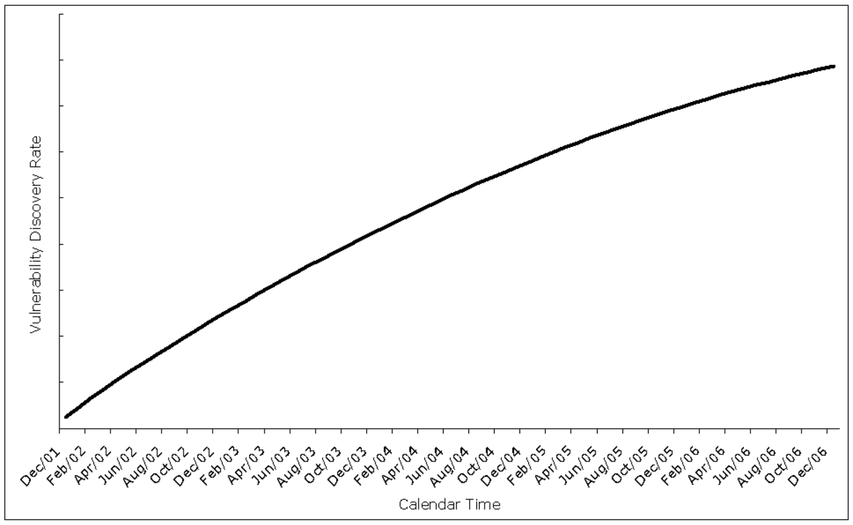

Figure 5.7: Mysql Weibull VDM Discovery Rate

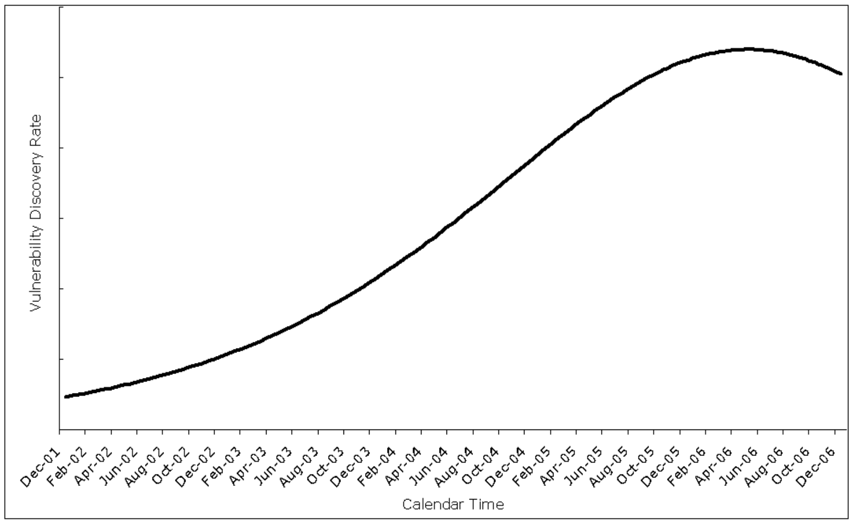

Figure 5.8: Mysql Logistic VDM Discovery Rate

The vulnerability discovery rate shows that the vulnerability discovery rate is not reduced after its release. This result shows that the vulnerability discovery rate from these models can be utilized for multiple version software as we assume. For checking the logistic vulnerability discovery model's hypothesis, though its model fitting is not significant, we derived the logistic vulnerability discovery rate from the model test result. The result is presented in Figure 5.8.

In Figure 5.7 and 5.8, the vulnerability discovery rate plotting shows different hypothesis in it. From these data, we can determine software age. In Logistic modeling, the remained age is three years, but in Weibull modeling, as we made hypothesis, the cumulative vulnerability derived from multiple version software, we can not determine age. Therefore, using just one modeling may have critical error to predict or estimate specific version vulnerability discovery trend.

Chapter 6

Vulnerability Discovery in Closed Source Software

The vulnerability discovery process in non-open source programs tends to exhibit a different discovery rate trend compared to open source software systems. In this chapter, we will examine several software systems to determine the difference in their vulnerability discovery trend. The organization of this chapter is based around multiple version software vulnerability discovery model testing, and we will see an alternative vulnerability discovery trend.

6.1 Modeling Multiple Version Software Vulnerability Discovery

Applying the MVDM to closed source software systems, we encounter a difficulty due to the inability to access the software structure or source code. Because of this limitation, finding software shared code in multiple software is not possible. Therefore, multiple version software vulnerability discovery modeling can be used as a theoretical component to explain the software vulnerability superposition effect. However, from the software vulnerability trend in open source software testing results, we can estimate the amount of shared software source code.

6.1.1 Modeling Windows OS Vulnerability Discovery

Multiple vulnerability discovery modeling tests, for closed source software, should be different than those used in open source software testing. Since there is no readily available software source code, it is hard to determine the software share constant parameter from source code. Here, we will examine the shared constant parameter, from multiple software vulnerability discovery modeling testing results. The examined software systems are Windows OS. These results can not be matched exactly to open source modeling test results, but we can estimate the shared ratio through an open source software experiment.

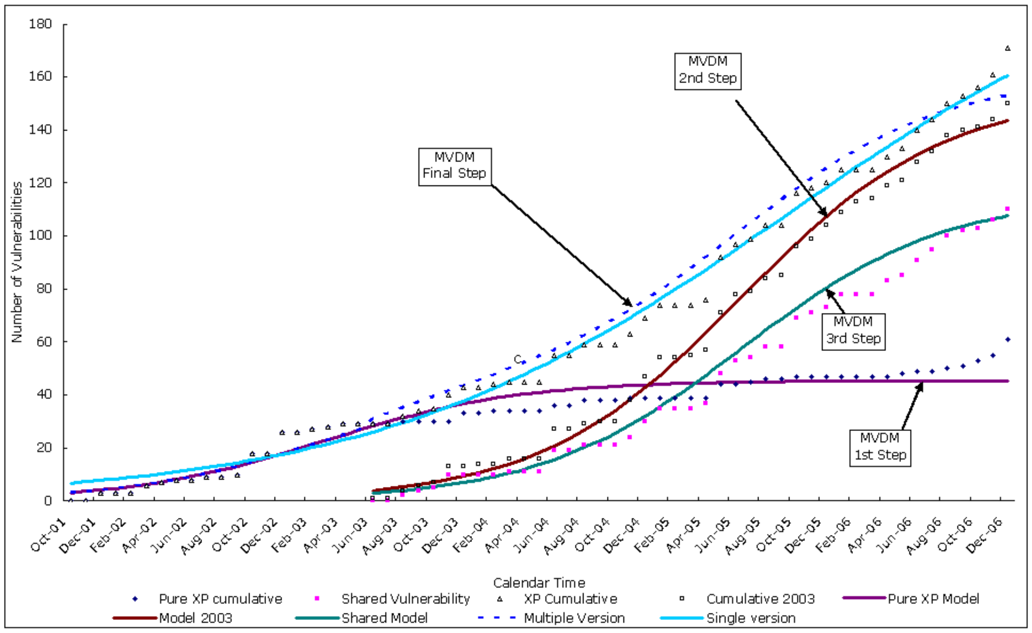

Figure 6.1: Windows XP Multiple Version VDM Fitting

In Figure 6.1, the X axis is calendar time and the Y axis indicates a cumulative vulnerability number of each version of software. Each points is the cumulative vulnerability number of each version of software based on time, lines represent the examined modeling line. In Figure 6.1 the vulnerability discovery model for pure vulnerability of Windows XP shows goodness of fit, and the model for Windows 2003 is not saturated. Using model fitting from shared vulnerability assumption, we can determine the amount of shared source code, which is reused in Windows 2003 from Windows XP. The sharing parameter is 0.75 in Windows 2003. Using the same hypothesis from the previous chapter, we could estimate the sharing ratio between two versions of software, and the estimated sharing ratio here is 75%.

	A	B	C	P-value	χ^2	χ^2critical	Result
Single AML Result	0.003	216.7	0.154	0.68	56.12	82.5287	Significant
MVDM 1st step	0.0034	45.49	0.346				
MVDM 2nd step	0.001	153.9	0.288				
MVDM overall result				0.99	19.27	82.5287	Significant

Table 6.1: Windows Multiple Version VDM Fitting Results
with α assumed to be 0.75

In Table 6.1, the result of single and multiple version software vulnerability discovery models are shown. This result indicates that single version vulnerability discovery model still works, however, the multiple software vulnerability discovery model shows better goodness of fit than single modeling.

6.2 Using Weibull and Logistic VDMs for Closed Source Software

The Weibull vulnerability discovery model still fits for some software - which has been in use for a long period of time and is less affected by testing versions, such as Windows OS. In this section, we examine the vulnerability discovery trend of windows OS and Internet Explorer using Logistic and Weibull distribution.

6.2.1 Modeling Windows OS Vulnerability Discovery

Windows OS has been adopted by many users, and the market share is significantly large. In this system, the superposition effect appears from time to time. The purpose of this subsection is to determine which single vulnerability discovery model fits best in closed source software. Because the superposition effect creates an asymmetric vulnerability discovery model, we are able to test the single model. We compare the two models using the Windows 98 vulnerability discovery trend as presented in Figure 6.2.

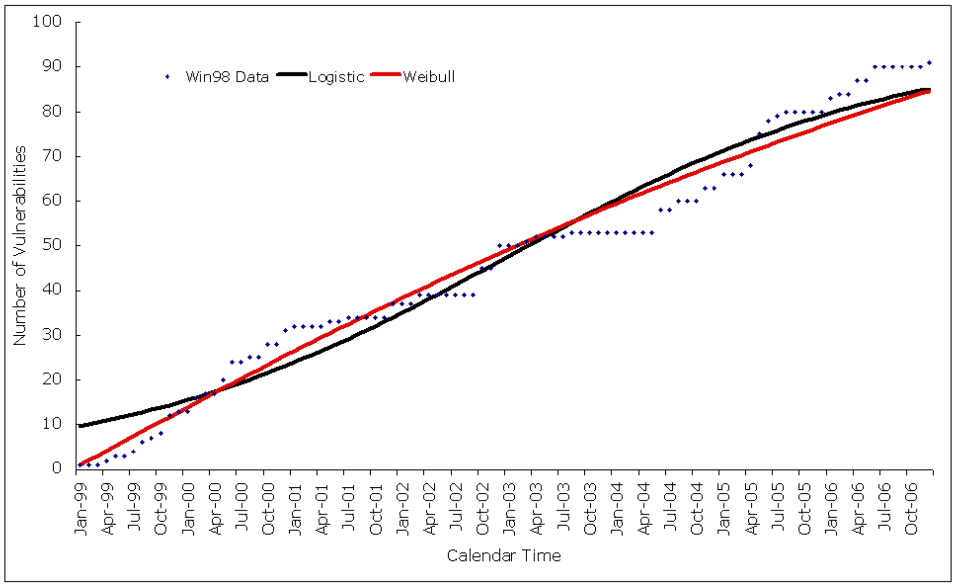

Figure 6.2: Windows 98 VDM Fitting

In Figure 6.2, the Logistic and Weibull vulnerability discovery trend shows a similar shape. Because of our assumption, about software discovery rate hypothesis, we expected opposing results from the two models. However, the result shows similarity between the models. The testing results are presented in Table 6.2.

	A(α)	B(β)	C(γ)	P-value	χ^2	χ^2critical	Result
Logistic Model	0.00047	95.0414	0.0974	0.206	106.019	119.87	significant
Weibull	1.05145	18.92987	165.3399	0.999	42.575	119.87	significant

(), Weibull function value

Table 6.2: Windows 98 VDM Fitting Results

We determined the goodness of fit of the Weibull distribution vulnerability discovery model. Similar to software vulnerability discovery modeling in the previous section, the single vulnerability discovery model still applies, however, the Chi-square test result shows better fitness of the asymmetrical hypothesis of the Weibull distribution. To examine our hypothesis, we derived the software vulnerability discovery rate using Weibull distribution. The vulnerability discovery rate is presented in Figure 6.3, the X axis is the calendar time and the Y axis indicates the vulnerability discovery rate distribution of entire software testing time. The vulnerability discovery rate shows only how many vulnerabilities of Windows 98 are still being discovered.

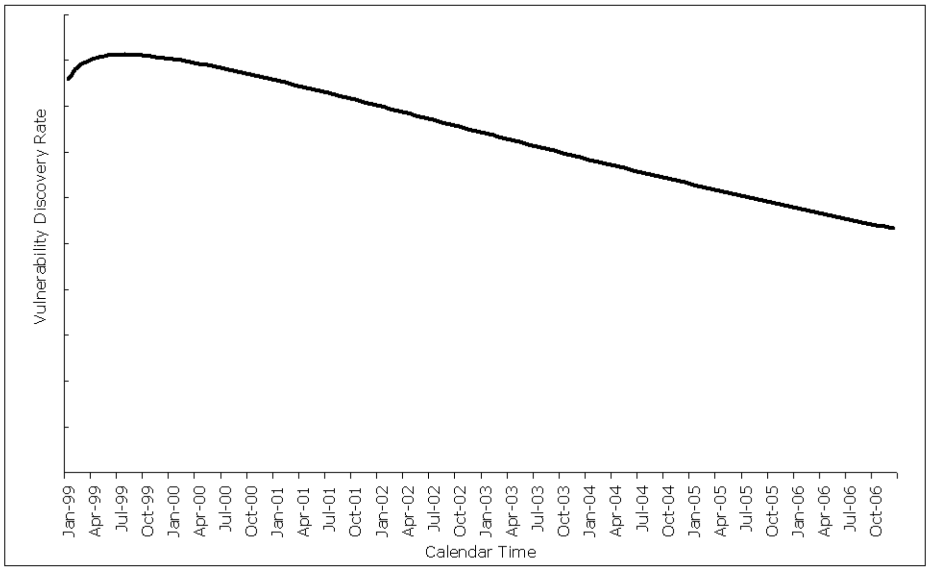

Figure 6.3: Windows 98 Weibull VDM Discovery Rate

Using the results of Table 6.3 and Figure 6.3, we find that the vulnerability discovery rate is not symmetric. This usage rate means that although the software market share is low, the vulnerability discovery rate does not decrease significantly compared to its initial discovery rate. For further testing, we examined Windows 2000 and the examined modeling is presented in 6.4.

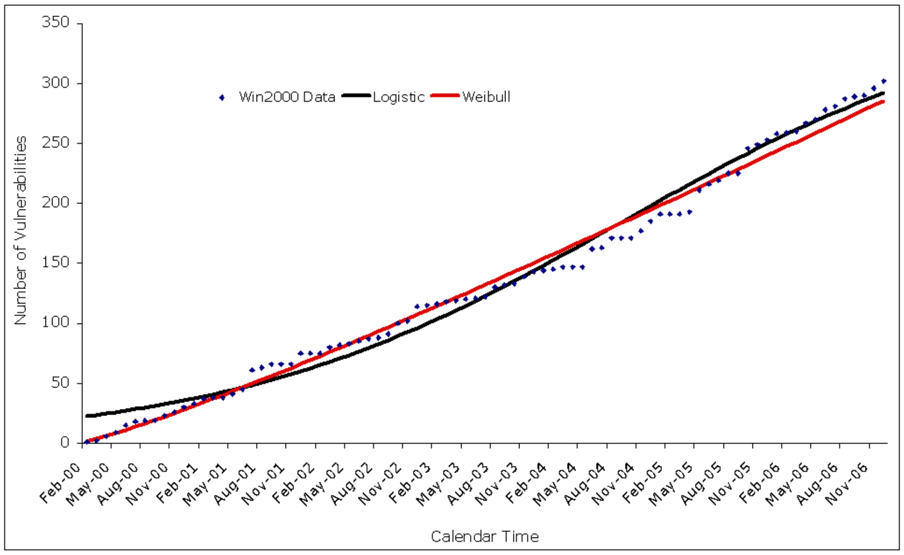

Figure 6.4: Windows 2000 VDM Fitting

The shape of the two models looks similar, however, the testing result shows opposing fitness between the two types. The result of the Windows 2000 modeling test is represented in Table 6.3.

	A(α)	B(β)	C(γ)	P-value	χ^2	χ^2critical	Result
Logistic Model	0.00014	350	0.0453	7.56E-06	149.601	105.267	Insignificant
Weibull	1.1755	92.286	2677.631	0.9992	47.138	105.267	significant

(), Weibull function value

Table 6.3: Windows 2000 VDM Fitting Results

In Table 6.3, we determine which software vulnerability modeling is better. The single Logistic model shows insignificant fitting, while Weibull vulnerability discovery modeling shows goodness of fit. In market share graph 2.1, the Windows 2000 market share shows significant decline, however, the vulnerability discovery rate shows an insignificant decreasing rate in Figure 6.4. Using the Weibull function, we derived the vulnerability discovery rate of Windows 2000 in Figure 6.5.

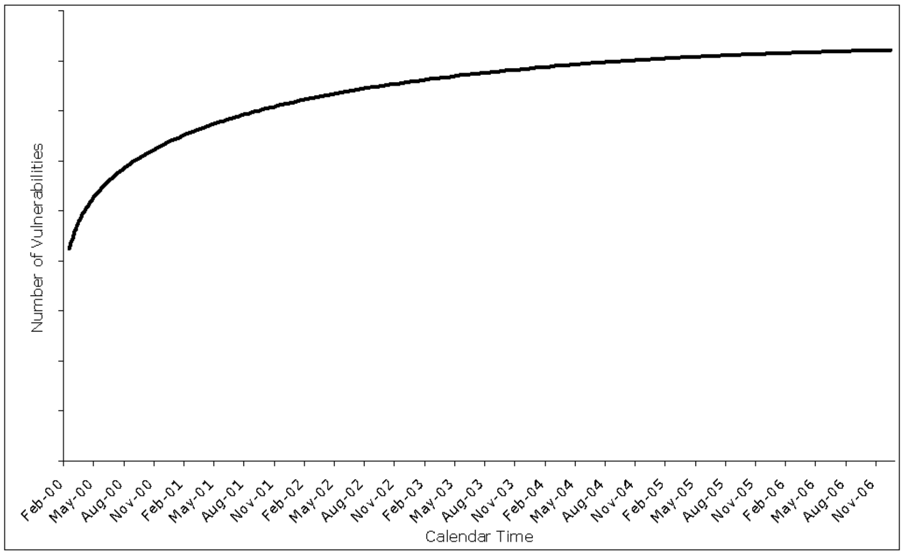

Figure 6.5: Windows 2000 Weibull VDM Discovery Rate

From this result, we can come to the conclusion that, over a long period of usage, the vulnerability discovery model may be distorted (superposition) with the release of newer software. This concept was presented in the previous chapter. For more testing to support this hypothesis, we will examine the Internet Explorer web browser in the following subsection.

6.2.2 Modeling Internet Explorer Vulnerability Discovery

We examined Internet Explorer using the single Logistic and Weibull software vulnerability discovery model to compare goodness of fit. The initial version of IE testing results are shown in 6.6. IE version 4 is one of the top market share versions of IE, and the IE line of browsers overall has always held considerable market share. This means that the usage and market share overall has been consistent, and satisfies our testing requirements. As an example, IE version 3 is shown in the previous Section.

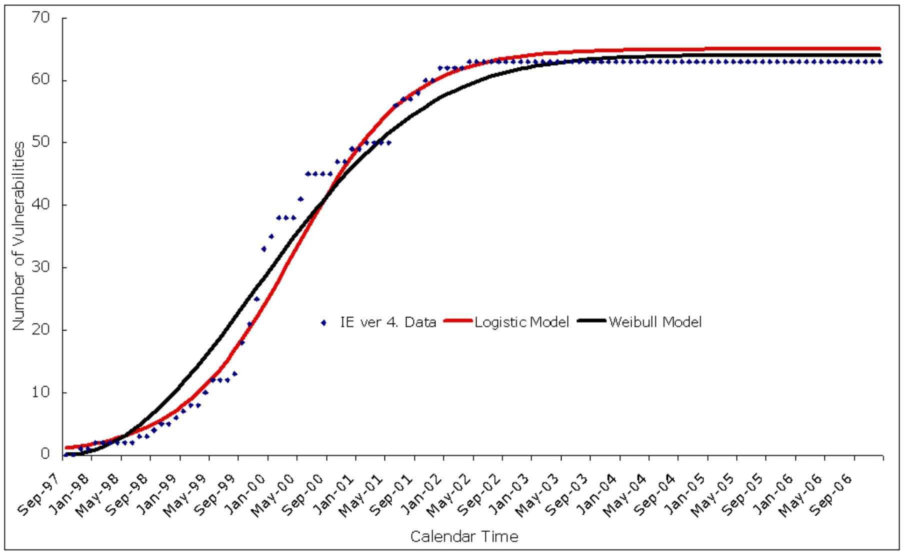

Figure 6.6: Internet Explorer 4 VDM Fitting

In vulnerability modeling using Internet Explorer version 4, a superposition effect appears in the model. However, we will ignore its effect since the effect in the single model is not considered. In Figure 6.6, the X axis is the calendar time and the Y axis indicates the cumulative vulnerability number, and each line indicates the examined modeling line. The points are the time based software version vulnerability number. These modeling results show how the vulnerability discovery modeling fits well toward the end of usage, since its software usage is almost zero in 2006. This Internet Explorer version 4 vulnerability discovery model testing result is presented in 6.4.

	A(α)	B(β)	C(γ)	P-value	χ^2	χ^2critical	Result
Logistic Model	0.002	65	1	1	30.111	137.701	Significant
Weibull	2.1609	35.001	64.004	1	46.901	137.701	Significant

(), Weibull function value

Table 6.4: Internet Explorer 4 VDM Fitting Results

In Table 6.4 vulnerability discovery modeling test, we find AML software vulnerability discovery modeling is fitting better than asymetrical vulnerability discovery modeling.

However, determining a superior vulnerability discovery model from this result is not clear, because both models show a significant fitness. The software vulnerability discovery rate coming from Logistic and Weibull distribuitons is presented in Figure 6.7.

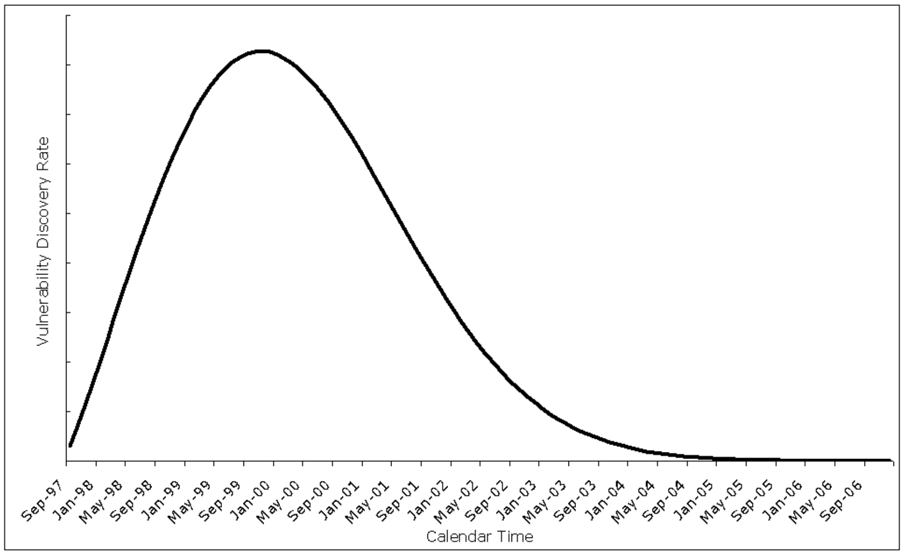

Figure 6.7: Internet Explorer 4 Weibull VDM Discovery Rate

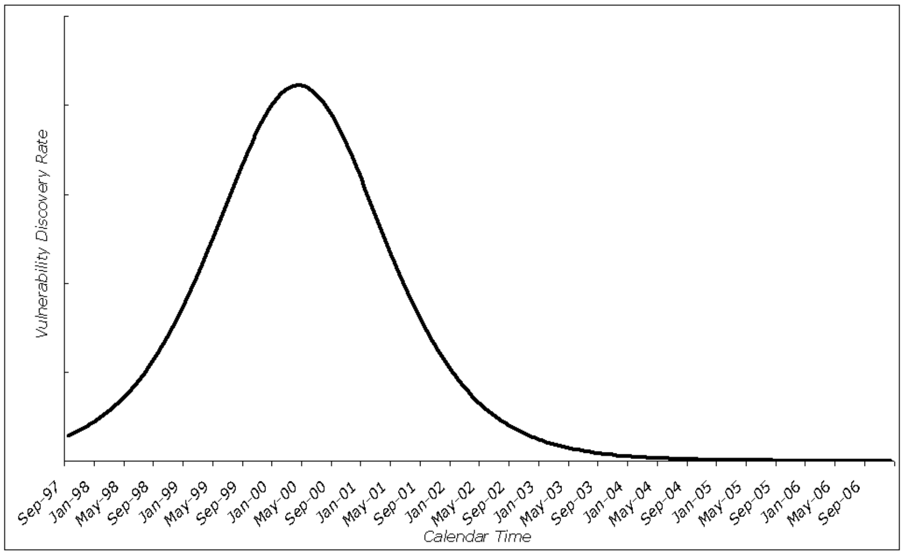

Figure 6.8: Internet Explorer 4 Logistic VDM Discovery Rate

Using Figure 6.7 and Table 6.4, we can determine that the vulnerability discovery rate is almost symmetric though use of the Weibull distribution model. However, the case study result may not fit to the software vulnerability trend, since its usage converted very quickly due to the release of the next version. Therefore, we tested if the vulnerability discovery rates are symmetric using the Logistic model in other cases.

For more testing, we chose the next version of IE, since the 5th version of IE held a significantly high market share among all the IE versions, and was in use for a long period of time. The testing result using two vulnerability discovery models is presented in Figure 6.9.

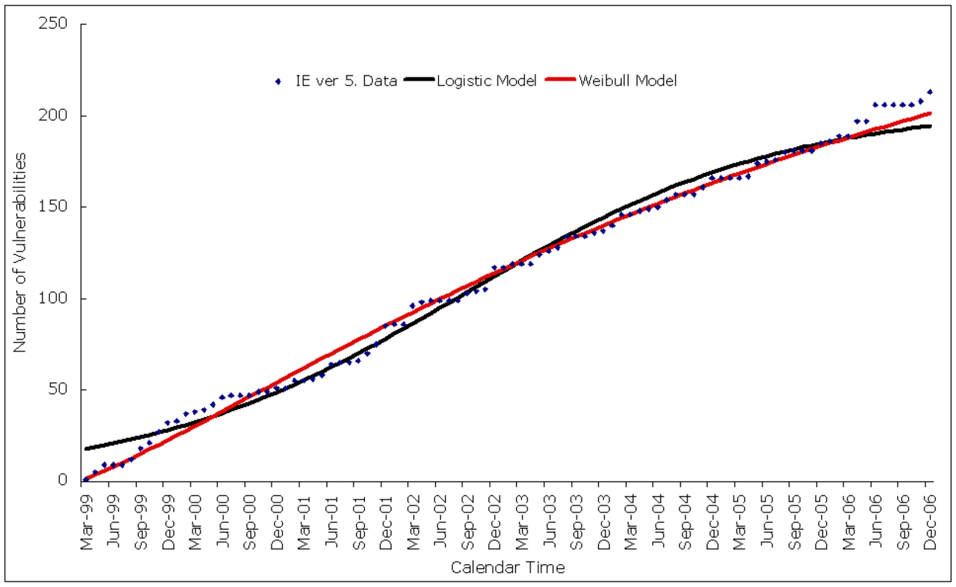

Figure 6.9: Internet Explorer 5 VDM Fitting

In Figure 6.9, a superposition effect can be seen at the end of the plot. However, because this superposition effect is not huge for single vulnerability model testing, we are able to test vulnerability discovery model fitting. In Table 6.5, the result of the modeling test is presented.

In Table 6.5, both models show goodness of fit, and we cannot determine which vulnera-

	A(α)	B(β)	C(γ)	P-value	χ^2	χ^2critical	Result
Logistic Model	0.0003	205.083	0.054	0.075	83.555	117.632	Significant
Weibull	1.1907	51.047	297.206	1	16.777	117.632	Significant

(), Weibull function value

Table 6.5: Internet Explorer 5 VDM Fitting Results

bility discovery modeling is better. This result shows that the long time usage vulnerability discovery modeling should be modified from a symmetric model to an asymmetric model, because the Weibull vulnerability discovery modeling shows a slightly greater significant fitness. Therefore, we derived the vulnerability discovery rate from the Weibull vulnerability discovery model.

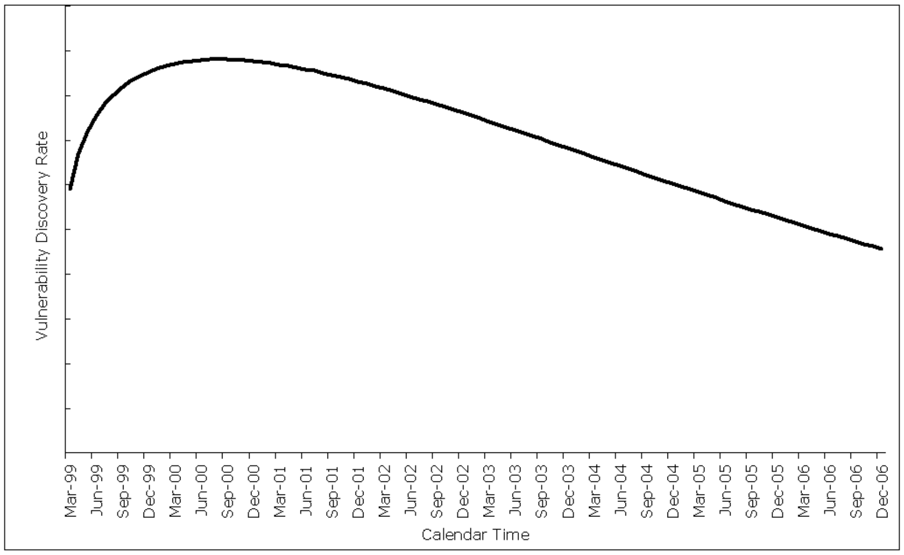

Figure 6.10: Internet Explorer 5 Weibull VDM Discovery Rate

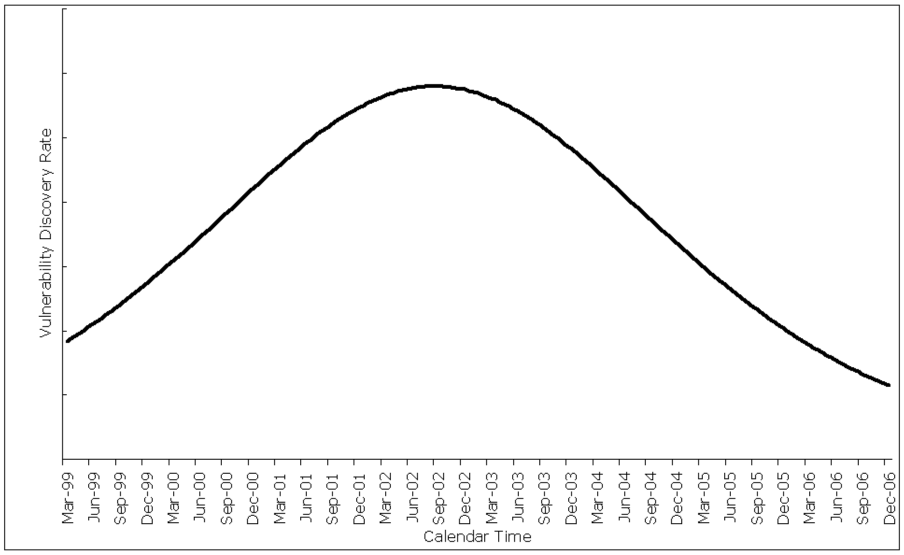

Figure 6.11: Internet Explorer 5 Logistic VDM Discovery Rate

In Figure 6.10, the Weibull vulnerability model shows goodness of fitness, and the vulnerability discover rate in Internet Explorer 5 displays an asymmetric rate. This asymmetric idea fits for almost all software systems in use for a long period of time, by many users. However, the end of this vulnerability model fitting shows a small superpostion effect. This is because the model also shows the software near its end of use. We can estimate that the superposition at this point will not be greater than that for the next version of software.

During modeling, the superposition effect can be ignored if it appears in the model toward the end of software use or vulnerability saturation, as the effect of these superpositions can be estimated. However, in special cases, the superposition effect can not be estimated if the effect has not yet appeared, or is too large to estimate. Thus, we will examine this superposition effect and modeling in the next version of IE's vulnerability discovery model tesing.

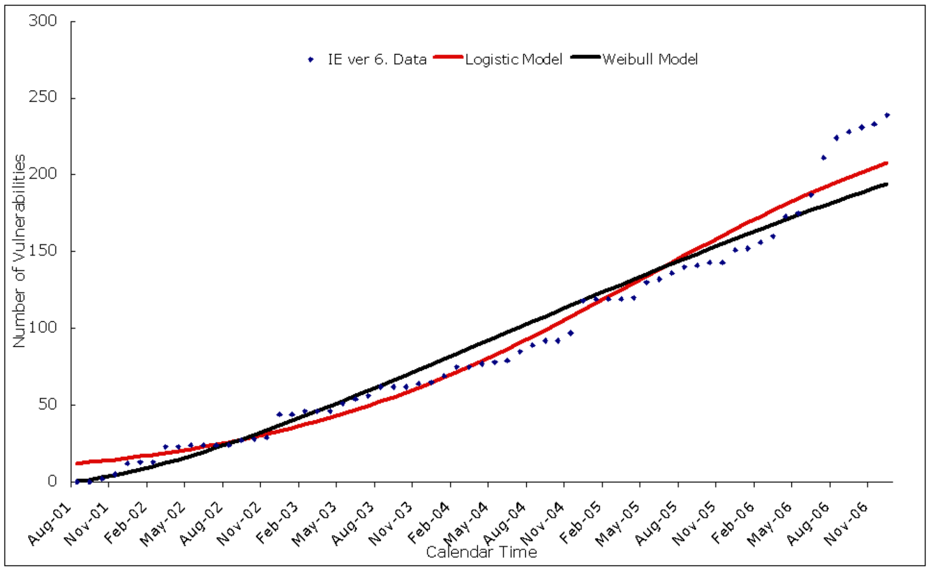

Figure 6.12: Internet Explorer 6 VDM Fitting

In Figure 6.12, as we expect, the superposition effect is larger than the previous version's superposition effect. In the Figure, Internet Explorer version 6 vulnerability discovery modeling test result shows an almost identical shape, however, the vulnerability discovery modeling result does not shows significant fitness as shown in Table 6.6.

	A(α)	B(β)	C(γ)	P-value	χ^2	χ^2critical	Result
Logistic Model	0.003	262	0.0844	0.0015	102.9	89.391	insignificant
Weibull	1.529	57	299	8.37215E-07	133.483	89.391	insignificant

(), Weibull function value

Table 6.6: Internet Explorer 6 VDM Fitting Results

In this fitting result, there may be hidden factors which are not considered. These results come from several changes of the internet environment. We focused on the cause of the superposition effect as being the next version software release. In this case, the release date of the 7th version of IE is Nov 2006. Therefore, we cannot fit modeling using the release date. However, we can consider the effect of the testing version of software. Some software vendors open their testing version software earlier than its planned release date for public testing. In the case of IE version 7, the finalized candidate version is released on August 2006. The date matches the month of the software superposition appearing. These factors are considered in Chapter 3 as Folded normal distribution. The difference between Folded distribution and superposition is that folded distribution handles later version software vulnerability discovery rate, while the superposition effect simply focuses on the previous vulnerability discovery rate. From Table 6.6, we conclude that both software vulnerability discovery models can not estimate the superposition efffect on the discovery rate. To estimate vulnerability superposition effect, we derived vulnerability discovery rate using Weibull distribution.

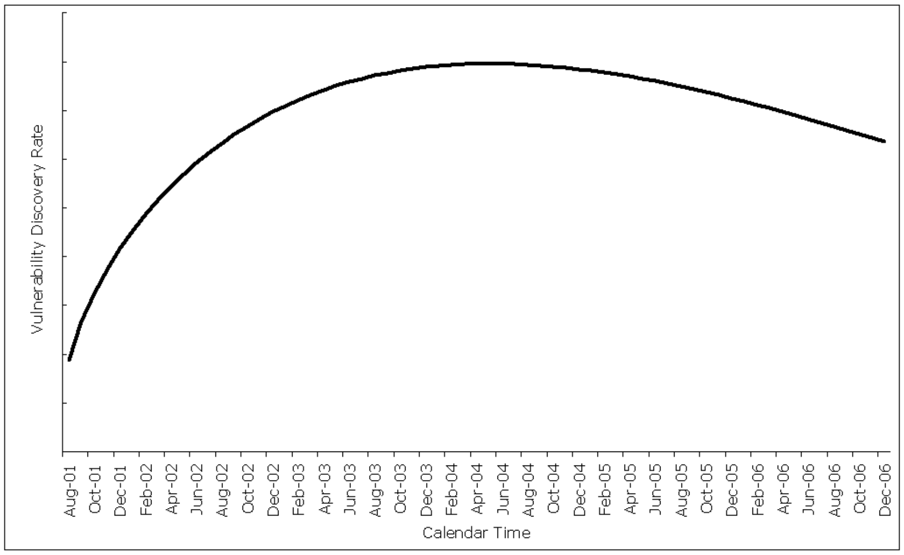

Figure 6.13: Internet Explorer 6 Weibull VDM Discovery Rate

The Internet Explorer version 6 vulnerability discovery rate shows a distorted bell shape from Weibull model though it is one of latest versions of Internet Explorer. This means that the vulnerability discovery rate , affected by superposition, cannot be estimated by using a Weibull model. However, from these results, we can see that the software system vulnerability discovery trend can be changed because of previous work in [24]. The vulnerability discovery rate is logistic analysis for IE 6 version is presented in Figure 6.14.

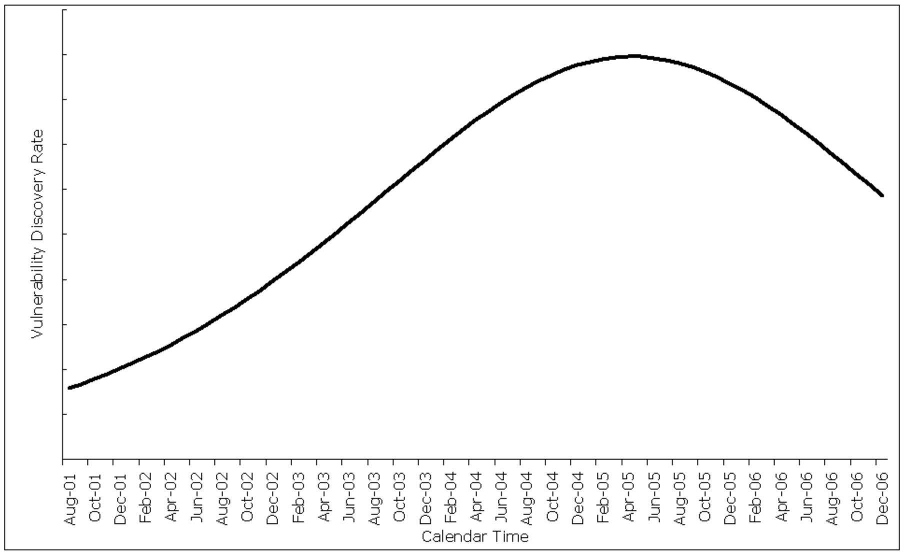

Figure 6.14: Internet Explorer 6 Logistic VDM Discovery Rate

In Figure 6.14, as we derived in previous chapter, discovery rate can be changed depends on the assumption for modeling whether it is multiple version software vulnerability discovery modeling. This shows how to choose among two model depends on factors of software.

Chapter 7

Discussion

Alternative approaches for vulnerability discovery modeling have been examined in Chapter 5 and 6. The impact of code sharing between successive version and code evolution and decay and subversions has been examined in Chapter 3.

For open source software systems, the single version AML VDM, which is presented in Chapter 4, assumes that the vulnerability discovery rate is symmetric for vulnerabilities which results are logistic modeling in Figure 4.6. However, code shared between two version causes a superposition effec due to shared vulnerabilities. The entire vulnerability discovery trend of a specific software shows an asymmetric trend which is shown in Figure 4.9. The overall trend which combines pure and shared vulnerabilities can be explained using multiple version modeling using the proposed MVDM. Applicability of the model was demonstated in Figure 5.1 and 5.2 in chapter 5. The results in Table 5.3 and 5.4 show that the multiple version vulnerability discovery model can be converted to a single version software vulnerability discovery model for open source software systems. The multiple version vulnerability discovery model shows a better fit than a single model, but the single model also works in open source software systems if we apply the single VDM depends on software life time. Since multiple version software vulnerability discovery creates an asymmetric trend, to get the comprehensive software vulnerability discovery rate, we used the Weibull distribution.

Although the The Weibull distribution fits to several software systems in Table 6.2 and 6.3, but not perfectly fit to some programs in Table 6.6 since the model assumes that the discovery rate has a constant increasing or decreasing rate. In open source software, the multiple version software vulnerability discovery model fits well, but the single model still fits acceptably.

For closed source software systems, the multiple version software vulnerability discovery model shows good goodness-of-fit in Table 6.1. However, because the source code is not available for examination, the shared source code ratio needed for modeling cannot be measured directly. Due to a significant superposition effect, typical vulnerability discovery trend changes a lot as time passes and cannot be assumed to be symmetric. Therefore, we have examined the Weibull distribution to find out its vulnerability discovery rate. The vulnerability discovery rate support our hypothesis in which the discovery rate decline is slower than the growth rate in the software life time. For Windows operating systems in particular, the Weibull distribution fits very well in Table 6.2 and 6.3. However, Internet Explorer shows only a weak fit as shown in Table 6.5 and 6.6.

The fit of each vulnerability discovery model depends on characteristics of the software system. Especially, software vulnerability rate is related to software life cycle, and modifications of software caused by requirement changes. In this study, we found that MVDM generally fits better than other models, but the single VDM still works for open source software which is shown in Table 5.3 and 5.4. The weakness of this multiple version software vulnerability discovery model is that there are additional factors that need to be considered. However, this creates a more detailed and complex model. In closed systems, the single logistic and Weibull models work well depending on the software system, even when the superposition effect is not explicitly considered. For reducing the error of the single logistic model, we can use Weibull distribution. For near future predictions with significant fit, both models are useful. However, for software used over a long period of time, an asymmetric vulnerability modeling is likely to the data better.

Chapter 8

Conclusion

Discovery and explicitation of vulnerabilities is one of the primary issues in software reliability and security. Since the effect of vulnerabilities on computer security is increasing, preventing them and repairing their impact is the main focus in software vulnerability studies. Estimating the number of software vulnerabilities is difficult during development, and many vulnerabilities are discovered after release. To address these issues, we can use a quantitative software vulnerability discovery models to estimate them. Software vulnerability discovery modeling should be considered with specific factors that can affect the discovery rate. We examined how several software vulnerability discovery models depend on product and process characteristics. The key factors in our work are source code availability usage, and the impact of multiple major and minor releases on software vulnerability discovery trend.

First, for open source software, we propose a new multiple version software vulnerability discovery modeling over previous which attempts to improve models. Two major open source programs are used as examples. In both cases, the fit of the model to the data is significant. We compared both single AML VDM and the MVDM for each case. The results show that MVDM works better for the Mysql DBMS case. However, using single AML VDM in Apache Web server testing result shows better goodness of fit than MVDM. However, the

MVDM goes into greater depth in explaining supperposition effect. We can estimate the shared code ratio for two successive releases for an open source software. This approache can be used to estimate future vulnerability.

Second, we have proposed the use of the Weibull distribution for software vulnerability discovery modeling. These result shows significant fit. However, this new approach shows how the software, even when a new release is available, affects the software vulnerability discovery rate. The Weibull vulnerability discovery modeling test results also fit several software systems. However, this approach does not explain the software superposition effect. For this we showed how the Weibull distribution can be applied to explain multiple vulnerability trends in a general form. The multiple vulnerability model can be used to show how superposition can result in the Weibull distribution. Using folded normal distribution, we also show how the Logistic vulnerability discovery trend can give rise to the Weibull distribution for the discovery rate. The limitation of our work is the need to measure the shared source code. For future validation of the proposed methods, we need to examine data from other major programs. Because of varying project characteristics, MVDM may not fit the data in some software. An example of this would be the Linux kernel. Due to its unique development, which all contributions from many independent sources, the vulnerability distribution does not follow a well-defined trend, and thus current models cannot be applied to it. Therefore, these multiple version softwares come from user participation make it difficult to estimate software vulnerability. However, assessing software vulnerability can be determined by several factors presented in chapter 2. Using the methods discussed we can accurately estimate software vulnerability.

For future work, the modularity and functionalities should be taken into account for developing a more accurate software vulnerability discovery. The annual seasonal trend of software vulnerability discovery can be an approach for accurate estimation of expected number of vulnerability to be found in next few month.

REFERENCES

[1] Fumio Akiyama. An example of software system debugging. In *IFIP Congress (1)*, pages 353–359, 1971.

[2] Omar Alhazmi, Yashwant K. Malaiya, and Indrajit Ray. Security vulnerabilities in software systems: A quantitative perspective. In *DBSec*, pages 281–294, 2005.

[3] Omar H. Alhazmi and Yashwant K. Malaiya. Modeling the vulnerability discovery process. In *ISSRE*, pages 129–138, 2005.

[4] Omar H. Alhazmi and Yashwant K. Malaiya. Measuring and enhancing prediction capabilities of vulnerability discovery models for apache and iis http servers. In *ISSRE*, pages 343–352, 2006.

[5] Omar H. Alhazmi and Yashwant K. Malaiya. Application of vulnerability discovery models to major operating system. In *IEEE Trans*, 2007.

[6] Omar H. Alhazmi, Sung-Whan Woo, and Yashwant K. Malaiya. Security vulnerability categories in major software systems. In *Communication, Network, and Information Security*, pages 138–143, 2006.

[7] Ross Anderson. Security in open versus closed systems - the dance of boltzmann, coase and moore. *Conf. on Open Source Software: Economics, Law and Policy*, pages 1–15, 2002.

[8] Ross J. Anderson. Why information security is hard-an economic perspective. In *ACSAC*, pages 358–365, 2001.

[9] Stuart E. Schechter Andy Ozment. Software security growth modeling: Examining vulnerabilities with reliability growth models. *The First Workshop on Quality of Protection*, 2005.

[10] Ashish Arora and Rahul Telang. Economics of software vulnerability disclosure. *IEEE Security & Privacy*, 3(1):20–25, 2005.

[11] Ch. Ali Asad, Muhammad Irfan Ullah, and Muhammad Jaffar-Ur Rehman. An approach for software reliability model selection. In *COMPSAC*, pages 534–539, 2004.

[12] Evelyn J. Barry, Sandra Slaughter, and Chris F. Kemerer. An empirical analysis of software evolution profiles and outcomes. In *ICIS*, pages 453–458, 1999.

[13] Google Press Center. www.google.com/press/zeitgeist. 2004.

[14] CERT. Cert coordination center. *http://www.cert.org*, 2006.

[15] Samuel D. Conte, Hubert E. Dunsmore, and Vincent Yun Shen. Software effort estimation and productivity. *Advances in Computers*, 24:1–60, 1985.

[16] Wenliang Du and Aditya P. Mathur. Testing for software vulnerability using environment perturbation. In *DSN*, pages 603–612, 2000.

[17] Stephen G. Eick, Todd L. Graves, Alan F. Karr, J. S. Marron, and Audris Mockus. Does code decay? assessing the evidence from change management data. *IEEE Trans. Software Eng.*, 27(1):1–12, 2001.

[18] Norman E. Fenton, Martin Neil, William Marsh, Peter Hearty, David Marquez, Paul Krause, and Rajat Mishra. Predicting software defects in varying development lifecycles using bayesian nets. *Information & Software Technology*, 49(1):32–43, 2007.

[19] William L. Fithen, Shawn V. Hernan, Paul F. O'Rourke, and David A. Shinberg. Formal modeling of vulnerability. *Bell Labs Technical Journal*, 8(4):173–186, 2004.

[20] Michael W. Godfrey and Qiang Tu. Evolution in open source software: A case study. In *ICSM*, pages 131–142, 2000.

[21] Internet Growth. Internet growth statistics. *http://www.internetworldstats.com*, 2006.

[22] Hitslink. Market share by net applications. *http://marketshare.hitslink.com*, 2006.

[23] Clemente Izurieta and James M. Bieman. The evolution of freebsd and linux. In *ISESE*, pages 204–211, 2006.

[24] Yashwant Malaiya Jinyoo Kim, Omar Alhazmi. Vulnerability in browsers: Trend in internet explorer and firefox. In *ISSRE*, Fast Abstract, pages 1–2, 2006.

[25] M. M. Lehman, Dewayne E. Perry, and Juan F. Ramil. Implications of evolution metrics on software maintenance. In *ICSM*, pages 208–, 1998.

[26] Paul Luo Li, James D. Herbsleb, and Mary Shaw. Forecasting field defect rates using a combined time-based and metrics-based approach: A case study of openbsd. In *ISSRE*, pages 193–202, 2005.

[27] Michael R. Lyu, Allen P. Nikora, and William H. Farr. A systematic and comprehensive tool for software reliability modeling and measurement. In *FTCS*, pages 648–653, 1993.

[28] Yashwant K. Malaiya and Pradip K. Srimani. An introduction to software reliability models. In *Int. CMG Conference*, pages 1237–1239, 1991.

[29] microsoft. microsoft product life cycle. *http://support.microsoft.com/lifecycle/search*, 2006.

[30] MITRE. Common vulnerabilities and exposures. *http://www.cve.mitre.org*, 2006.

[31] Audris Mockus, Roy T. Fielding, and James D. Herbsleb. A case study of open source software development: the apache server. In *ICSE*, pages 263–272, 2000.

[32] David Moore, Colleen Shannon, and Kimberly C. Claffy. Code-red: a case study on the spread and victims of an internet worm. In *Internet Measurement Workshop*, pages 273–284, 2002.

[33] J. D. Musa. *Software Reliability Engineering*. McGraw-Hill, 1999.

[34] John D. Musa. Software reliability measurement. *Journal of Systems and Software*, 1:223–241, 1980.

[35] John D. Musa. Software-reliability-engineered testing practice (tutorial). In *ICSE*, pages 628–629, 1997.

[36] John D. Musa and Kazuhira Okumoto. A logarithmic poisson execution time model for software reliability measurement. In *ICSE*, pages 230–238, 1984.

[37] John D. Musa and Kazuhira Okumoto. Application of basic and logarithmic poisson execution time models in software reliability measurement. In *Software Reliability Modelling and Identification*, pages 68–100, 1987.

[38] Mysql. The mysql product achieve. *http://www.download.mysql.com/achives.php*, 2006.

[39] N.Balakrishnan. *Handbook of the logistic Distribution*. Marcel Dekker.inc, 1992.

[40] Netcraft. April 2007 web server survey. *http://news.netcraft.com*, 2006.

[41] NVD. National vulnerability database. *http://nvd.nist.gov*, 2006.

[42] Andy Ozment. Milk or wine: Does software security improve with age? *The Fifteenth Usenix Security Symposium*, 2005.

[43] James W. Paulson, Giancarlo Succi, and Armin Eberlein. An empirical study of open-source and closed-source software products. *IEEE Trans. Software Eng.*, 30(4):246–256, 2004.

[44] The Code Project. http://www.codeproject.com/tools/difftool.asp. 2007.

[45] Eric Rescorla. Security holes... who cares? *12th USENIX Security Symposium*, 3(1):75–90, 2003.

[46] Eric Rescorla. Is finding security holes a good idea? *IEEE Security & Privacy*, 3(1):14–19, 2005.

[47] Juha Roning. *Communication in the Software Vulnerability Reporting Process*. Oulu University Secure Programming Group, 2003.

[48] Scurityfocus. http://www.securityfocus.com. 2006.

[49] Secunia. Vulnerability and virus information. *http://secunia.com*, 2006.

[50] Mark Sherriff. Utilizing verification and validation certificates to estimate software defect density. In *ESEC/SIGSOFT FSE*, pages 381–384, 2005.

[51] SLOCCount. Source line count. *http://www.dwheeler.com/sloccount*, 2006.

[52] SourceForge.net. http://sourceforge.net/projects/lcounter. 2007.

[53] Sufnet-Cert. Cisco security advisory, http://cert.surfnet.nl/s/2004/s-04-012.htm. 2004.

[54] Michael Sutton and Frank Nagle. Emerging economic models for vulnerability research. *The Fifth Workshop on the Economics of Information Security*, 2006.

[55] Symantec. Symantec internet security threat report: Trends for july 05–december 05. Technical report, Symantec, 2006.

[56] Xiaolin Teng and Hoang Pham. A software growth model for n-version programming systems. volume 51, 2002.

[57] Sung-Whan Woo, Omar H. Alhazmi, and Yashwant K. Malaiya. Assessing vulnerabilities in apache and iis http servers. In *DASC*, pages 103–110, 2006.