DISSERTATION

ASSESSING VULNERABILITIES IN SOFTWARE SYSTEMS: A QUANTITATIVE APPROACH

Submitted by

Omar Alhazmi

Department of Computer Science

In partial fulfillment of the requirements
For the Degree of Doctor of Philosophy
Colorado State University
Fort Collins, Colorado
Spring 2007

COLORADO STATE UNIVERSITY

November, 6th 2006

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY OMAR ALHAZMI ENTITLED ASSESSING VULNERABILITIES IN SOFTWARE SYSTEMS: A QUANTITATIVE APPROACH BE ACCEPTED AS FULFILLING IN PART THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

[illegible]
Dr. Anura Jayasumana

[illegible]
Dr. Indrakshi Ray

[illegible]
Adviser Dr. Yashwant K. Malaiya

Indrajit Ray
Co- Adviser Dr. Indrajit Ray

[illegible]
Department Head Dr. Darrell Whitley

ABSTRACT OF DISSERTATION

ASSESSING VULNERABILITIES IN SOFTWARE SYSTEMS: A QUANTITATIVE APPROACH

Security and reliability are two of the most important attributes of complex software systems. It is now common to use quantitative methods for evaluating and managing reliability. Software assurance requires similar quantitative assessment of software security, however only limited work has been done on quantitative aspects of security. The analogy with software reliability can help developing similar measures for software security. However, there are significant differences that need to be identified and appropriately acknowledged. This work examines the feasibility of quantitatively characterizing major attributes of security using its analogy with reliability. In particular, we investigate whether it is possible to predict the number of vulnerabilities that can potentially be identified in a current or future release of a software system using analytical modeling techniques.

Datasets from several major complex software systems have been collected and analyzed, they represent both open-source and proprietary software systems. They include most of the major operating systems, web servers, and web browsers currently in use. The data about vulnerabilities discovered in these software systems are analyzed to identify trends and the goodness of fit with the proposed models is statistically examined.

Vulnerability datasets are examined to determine if the *vulnerability density* in a program is a practical and useful measure. We attempt to identify the quantitative relationship between software defects and vulnerabilities. The results indicate that vulnerability density is relatively stable for specific classes of systems and therefore, is a meaningful metric.

The dynamics of vulnerability discovery is thoroughly examined in detail with the hope that it may lead us to an estimate of the magnitude of the undiscovered vulnerabilities still present in the system. We examine the *vulnerability discovery process* to determine whether models can be developed to project future trends. The prediction capabilities of the proposed quantitative methods have been investigated. The results show good prediction accuracy when applied to several of the operating systems and web-servers. Finally, vulnerabilities taxonomies were considered and the quantitative approaches were also applied to categorized vulnerability datasets as well.

Categorized vulnerabilities analysis suggests that some vulnerabilities categories are generally more severe. We also note that in some products, some categories include a larger number of high severity vulnerabilities. This fact can be used as a guideline to design better test cases that assigns a higher priority to selected categories in order to optimize test effectiveness and reduce the cost of testing.

Omar H. O. Alhazmi
Department of Computer Science
Colorado State University
Fort Collins, CO 80523
Spring 2007

ACKNOWLEDGEMENT

First of all, I would like to thank God for everything. I also would like to thank all the people who helped me during this work. I am especially thankful to my adviser, Dr. Yashwant Malaiya, for his encouragement, guidance, and support during my graduate study. Also, I would like to thank my co-adviser Dr. Indrajit Ray for his supervision and vital advices, Dr. Indrakshi Ray and Dr. Anura Jayasumana for agreeing to be on my thesis committee and reviewing my thesis.

I would like also to thank my research colleagues Sung-Whan Woo and Jin-Yoo Kim for their feedbacks and suggestions.

DEDICATION

I would like to dedicate this dissertation to my parents, brothers, and sisters for their love, encouragement and support while I was far away from home during my Ph.D. program.

I would also like dedicate it to my wife, Fahamiah Jeliadan, my daughter, Haneen, and my son, Hussien, for all their patient, love, and support over the years, which helped me completing this dissertation.

TABLE OF CONTENTS

ABSTRACT OF DISSERTATION .... ***iii***

ACKNOWLEDGEMENT .... ***v***

DEDICATION .... ***vi***

TABLE OF CONTENTS .... ***vii***

LIST OF FIGURES .... ***x***

LIST OF TABLES .... ***xiii***

CHAPTER 1 .... ***1***

INTRODUCTION ***1***

1.1 INTRODUCTION **1**

1.2 MOTIVATION **5**

1.3 CONTRIBUTIONS **7**

1.4 RELATED WORK AND LITERATURE REVIEW **7**

1.4.1 BACKGROUND 7

1.4.2 DEFINITION OF SOFTWARE VULNERABILITY 9

1.4.3 QUANTITATIVE ASSESSMENT OF SOFTWARE SECURITY 10

1.4.4 MODELING VULNERABILITIES AND INCIDENTS 12

1.5 ORGANIZATION OF THE DISSERTATION **13**

CHAPTER 2 .... ***15***

VULNERABILITY DENSITY ***15***

2.1 APPLICATIONS OF VULNERABILITY DENSITY **17**

2.2 MEASURING VULNERABILITY DENSITY OF SOME SOFTWARE SYSTEMS **18**

2.3 CONCLUSION **21**

CHAPTER 3 .... ***22***

VULNERABILITY DISCOVERY TRENDS AND VULNERABILITY DISCOVERY MODELS ***22***

3.1 INTRODUCTION **22**

3.2 VULNERABILITY TRENDS **22**

3.2.1 DATA SOURCES 23

3.2.2 VULNERABILITY DISCOVERY TRENDS IN OPERATING SYSTEMS 23

3.2.3 VULNERABILITY DISCOVERY TRENDS IN WEB SERVERS 26

3.2.4 VULNERABILITY DISCOVERY TRENDS IN WEB BROWSERS 27

3.2.5 OBSERVATIONS 28

3.3 VULNERABILITY DISCOVERY MODELS **29**

3.3.1 THE LOGISTIC VULNERABILITY DISCOVERY MODEL 29

3.3.2 THE LINEAR VULNERABILITY DISCOVERY MODEL 34

3.3.3 THE EFFORT BASED VULNERABILITY DISCOVERY MODEL 35

3.4 CONCLUSION **37**

CHAPTER 4 .... ***39***

VALIDATING VULNERABILITY DISCOVERY MODELS ***39***

4.1 INTRODUCTION **39**

4.2 FITTING THE MODELS AND EVALUATING THE GOODNESS OF FIT **39**

4.3 MODELING VULNERABILITIES IN OPERATING SYSTEMS **40**

4.3.1 MODELING OPERATING SYSTEMS VULNERABILITIES USING THE LOGISTIC MODEL (AML). 41

4.3.2 VALIDATING THE LINEAR MODEL (LM) ON OPERATING SYSTEMS 48

4.3.3 DISCUSSION 54

4.4 MODELING VULNERABILITIES IN WEB SERVERS **55**

4.4.1 MODELING WEB SERVERS VULNERABILITIES USING THE LOGISTIC MODEL (AML) 58

4.4.2 MODELING WEB SERVER VULNERABILITIES USING THE LINEAR MODEL (LM) 62

4.4.3 DISCUSSION 65

4.5 MODELING VULNERABILITIES IN WEB BROWSERS **65**

4.5.1 MODELING WEB BROWSERS VULNERABILITIES USING THE LOGISTIC MODEL (AML) 67

4.5.2 MODELING WEB BROWSERS VULNERABILITIES USING THE LINEAR MODEL (LM) 71

4.5.3 DISCUSSION 74

4.6 MODELING VULNERABILITIES USING THE EFFORT-BASED MODEL **75**

4.6.1 VALIDATING THE EFFORT-BASED MODEL ON OPERATING SYSTEMS 76

4.6.2 VALIDATING THE EFFORT-BASED MODEL ON HTTP SERVERS 77

4.6.3 DISCUSSION 79

4.7 SHARED VULNERABILITIES IN SUCCESSIVE VERSIONS OF SOFTWARE SYSTEMS **80**

4.8 CONCLUSION **82**

CHAPTER 5 .... ***84***

COMPARING VULNERABILITY DISCOVERY MODELS ***84***

5.1 INTRODUCTION **84**

5.2 PREVIEW OF VULNERABILITY DISCOVERY MODELS **86**

5.2.1 ANDERSON THERMODYNAMIC MODEL (AT) 86

5.2.2 ALHAZMI-MALAIYA LOGISTIC MODEL (AML) 87

5.2.3 RESCOLA QUADRATIC MODEL (RQ) 88

5.2.4 RESCOLA EXPONENTIAL MODEL (RE) 90

5.2.5 LOGARITHMIC POISSON MODEL (LP) 91

5.2.6 THE LINEAR MODEL (LM) 92

5.3 COMPARING THE MODELS **92**

5.3.1 THE DATA SOURCES 92

5.3.2 EVALUATION OF GOODNESS OF FIT 94

5.3.3 MODELS ADEQUACY CRITERIA 94

5.4 FITTING THE DATA SETS TO THE MODELS **95**

5.5 CONCLUSIONS **103**

CHAPTER 6 .... ***106***

ESTIMATING THE NUMBER OF VULNERABILITIES ... ***106***

6.1 INTRODUCTION ... **106**

6.2 EVALUATING THE PREDICTION CAPABILITY OF THE VULNERABILITY DISCOVERY MODELS ... **108**

6.2.1 THE STATIC APPROACH ... 108
6.2.2 THE DYNAMIC APPROACH ... 109
6.2.3 THE COMBINED DYNAMIC/STATIC APPROACH ... 110

6.3 PREDICTION CAPABILITIES FOR ESTIMATING VULNERABILITIES IN OPERATING SYSTEMS ... **110**

6.4 PREDICTION CAPABILITIES FOR ESTIMATING VULNERABILITIES IN WEB SERVERS ... **114**

6.4.1 LONG TERM PREDICTION ... 116
6.4.2 LONG TERM PREDICTION ACCURACY TEST ... 117
6.4.3 ADAPTIVE LONG-TERM PREDICTION ACCURACY TEST ... 120
6.4.4 ESTIMATING VULNERABILITIES FOR THE FOLLOWING YEAR ... 123
6.4.5 ESTIMATING THE FOLLOWING YEAR'S VULNERABILITIES BY DIRECT MODEL APPLICATION. 125
6.4.6 ESTIMATING THE FOLLOWING YEAR'S VULNERABILITIES BY MODEL APPLICATION WITH ADAPTATION ... 127

6.5 DISCUSSION ... **129**

6.6 CONCLUSION ... **130**

CHAPTER 7 ... ***132***

TAXONOMIZED VULNERABILITIES: ANALYSIS AND MODELING ... ***132***

7.1 INTRODUCTION ... **132**

7.2 VULNERABILITY CLASSIFICATIONS BY CATEGORY ... **132**

7.3 VULNERABILITY CLASSIFICATIONS BY SEVERITY ... **134**

7.4 MODELING VULNERABILITIES CLASSIFIED BY CATEGORY ... **135**

7.5 MODELING VULNERABILITIES CLASSIFIED BY SEVERITY LEVEL ... **140**

7.6 ANALYZING THE LINK BETWEEN VULNERABILITY CATEGORIES AND VULNERABILITY SEVERITY LEVELS ... **142**

7.7 CONCLUSIONS ... **146**

CHAPTER 8 ... ***148***

CONCLUSIONS AND FUTURE WORK ... ***148***

8.1 SUMMARY AND CONCLUSIONS ... **148**

8.1.1 MODELING OF VULNERABILITIES ... 148
8.1.2 VULNERABILITY DENSITY ... 149
8.1.3 PREDICTION CAPABILITIES ... 150
8.1.4 COMPARING VULNERABILITY DISCOVERY MODELS ... 151
8.1.5 MODELING CATEGORIZED VULNERABILITIES AND TESTING FOR VULNERABILITIES ... 153

8.2 FUTURE WORK ... **154**

REFERENCES ... ***156***

LIST OF FIGURES

Figure 3-1. The cumulative number of vulnerabilities of some Windows systems.......... 24
Figure 3-2. The cumulative number of vulnerabilities of Red Hat Linux versions 6.2 and 7.1 25
Figure 3-3. The cumulative number of vulnerabilities of Red Hat Fedora.......... 25
Figure 3-4. The cumulative number of vulnerabilities of Solaris operating systems versions 7, 8 and 9 26
Figure 3-5. The cumulative number of vulnerabilities of Apache versions 1.x and 2.x... 27
Figure 3-6. The cumulative number of vulnerabilities of Microsoft IIS 4 and IIS 5 web servers 27
Figure 3-7. The cumulative number of vulnerabilities of Microsoft Internet Explorer versions 4, 5 and 6 28
Figure 3-8.The basic 3-phase S-shaped model 30
Figure 3-9. Alhazmi-Malaiya Logistic (AML) model 32
Figure 3-10. AML model with the transition points and mid point shown 34
Figure 3-11. The percentages of installed base of some Windows systems.......... 36
Figure 3-12. Hypothetical plot of the Effort-Based Model.......... 37
Figure 4-1. Fitting Windows 95 cumulative Vulnerabilities to the AML model 41
Figure 4-2. Fitting Windows 98 cumulative Vulnerabilities to the AML model 42
Figure 4-3. Fitting Windows XP cumulative Vulnerabilities to the AML model 43
Figure 4-4. Fitting Windows 2000 cumulative Vulnerabilities to the AML model 43
Figure 4-5. Fitting Windows NT 4.0 cumulative Vulnerabilities to the AML model...... 44
Figure 4-6. Fitting Red Hat Linux 6.2 cumulative Vulnerabilities to the AML model.... 45
Figure 4-7. Fitting Red Hat Linux 7.1 cumulative Vulnerabilities to the AML model.... 45
Figure 4-8. Fitting Red Hat Fedora cumulative Vulnerabilities to the AML model 46
Figure 4-9. Fitting Solaris 7.0 cumulative Vulnerabilities to the AML model.......... 46
Figure 4-10. Fitting Solaris 8.0 cumulative Vulnerabilities to the AML model.......... 47
Figure 4-11. Fitting Solaris 9.0 cumulative Vulnerabilities to the AML model.......... 47
Figure 4-12. Fitting Windows 95 cumulative Vulnerabilities to the LM model 49
Figure 4-13. Fitting Windows 98 cumulative Vulnerabilities to the LM model 49
Figure 4-14. Fitting Windows XP cumulative Vulnerabilities to the LM model 50
Figure 4-15. Fitting Windows NT 4.0 cumulative Vulnerabilities to the LM model....... 50
Figure 4-16. Fitting Windows 2000 cumulative Vulnerabilities to the LM model 51
Figure 4-17. Fitting Windows Red Hat Linux 6.2 to the linear model (LM) 51
Figure 4-18. Fitting Windows Red Hat Linux 7.1 to the linear model (LM) 52
Figure 4-19. Fitting Windows Red Hat Fedora to the linear model (LM).......... 52
Figure 4-20. Fitting Solaris 7.0 cumulative Vulnerabilities to the LM model 53
Figure 4-21. Fitting Solaris 8.0 cumulative Vulnerabilities to the LM model 53
Figure 4-22. Fitting Solaris 9.0 cumulative Vulnerabilities to the LM model 54
Figure 4-23. Fitting Apache 1 cumulative Vulnerabilities to the AML model 59
Figure 4-24. Fitting Apache 2 cumulative Vulnerabilities to the AML model 60
Figure 4-25 Fitting IIS 4.0 cumulative vulnerabilities to the logistic model (AML) 60
Figure 4-26. Fitting IIS 5.0 cumulative vulnerabilities to the logistic model (AML) 61
Figure 4-27. Fitting Apache 1 cumulative vulnerabilities to the linear model.......... 62
Figure 4-28. Fitting Apache 2 cumulative vulnerabilities to the linear model (LM) 63

Figure 4-29. Fitting IIS 4.0 cumulative vulnerabilities to the linear model (LM)............ 63
Figure 4-30. Fitting IIS 5.0 cumulative vulnerabilities to the linear model (LM)............ 64
Figure 4-31. Fitting IE 4.0 cumulative Vulnerabilities to the AML model....................... 67
Figure 4-32. Fitting IE 5.0 cumulative Vulnerabilities to the AML model....................... 68
Figure 4-33. Fitting IE 6.0 cumulative Vulnerabilities to the AML model....................... 69
Figure 4-34. Fitting Firefox 1.0 cumulative Vulnerabilities to the AML model.............. 69
Figure 4-35. Fitting Firefox 1.5 cumulative Vulnerabilities to the AML model.............. 70
Figure 4-36. Fitting IE 4 cumulative vulnerabilities to the linear model 71
Figure 4-37. Fitting IE 5 cumulative vulnerabilities to the linear model 72
Figure 4-38 Fitting IE 6 cumulative vulnerabilities to the linear model 72
Figure 4-39. Fitting Firefox 1.0 cumulative vulnerabilities to the linear model............... 73
Figure 4-40. Fitting Firefox 1.5 cumulative vulnerabilities to the linear model............... 73
Figure 4-41. Fitting Windows 98 data (Effort-Based Model) 76
Figure 4-42. Fitting Windows NT 4.0 (Effort-Based Model)... 77
Figure 4-43. Fitting Apache vulnerability data to the Effort Based Model....................... 78
Figure 4-44. Fitting IIS vulnerability data to the Effort Based Model 78
Figure 4-45. Plot of the Vulnerabilities of Windows NT 4.0 and Windows 2000 showing shared vulnerabilities 80
Figure 4-46. Plot of the Vulnerabilities of Fire Fox web browser versions 1.0 and 1.5 showing shared vulnerabilities.. 81
Figure 4-47. Plot of the Vulnerabilities of Fire Fox web browser versions 1.0 and 1.5 showing shared vulnerabilities.. 82
Figure 5-1. Hypothetical plots of the Anderson Thermodynamic (AT) model 86
Figure 5-2. Hypothetical plots of the Alhazmi-Malaiya Logistic (AML) model 88
Figure 5-3. Hypothetical plot of the Rescorla Quadratic (RQ) model............................... 89
Figure 5-4. Hypothetical of the Rescorla exponential (RE) model 90
Figure 5-5. Hypothetical plots of the Logarithmic Poisson (LP) model........................... 91
Figure 5-6. Fitted VDMs with actual Windows 95 data.. 95
Figure 5-7.Fitted VDMs with actual Windows 95 data... 96
Figure 5-8. Fitted VDMs with actual Red Hat Linux 6.2 data 97
Figure 5-9. Fitted VDMs with actual Red Hat Linux 6.2 data 98
Figure 5-10. Fitted VDMs with actual Windows XP data... 100
Figure 5-11. Fitted VDMs with actual Windows XP data... 100
Figure 5-12. . Fitted VDMs with actual Fedora data.. 102
Figure 5-13. Fitted VDMs with actual Fedora data.. 102
Figure 6-1. Estimation error values for Windows 98 112
Figure 6-2. Estimation error values for Windows 2000 113
Figure 6-3. Estimation error values for Red Hat Linux 7.1 113
Figure 6-4. Accuracy of the models in estimating Apache 1.x vulnerabilities data....... 118
Figure 6-5. Accuracy of the models to estimate Apache 2.x vulnerabilities data 118
Figure 6-6. Accuracy of the models for estimating IIS 4.0 vulnerability data 119
Figure 6-7. Accuracy of the models for estimating IIS 5.0 vulnerability data 120
Figure 6-8. Accuracy of the adaptive models in estimating Apache 1.x vulnerabilities data 121
Figure 6-9. Accuracy of the adaptive models in estimating Apache 2.x vulnerabilities data 121

Figure 6-10. Accuracy of the adaptive models in estimating IIS 4.0 vulnerability data 122
Figure 6-11. Accuracy of the adaptive models in estimating IIS 5.0 vulnerability data 123
Figure 6-12. Estimations of vulnerabilities for the following year (quarterly)............. 125
Figure 6-13. Estimations of vulnerabilities for the following year (quarterly)............. 126
Figure 6-14. Estimations of vulnerabilities for the following year (quarterly)............. 126
Figure 6-15. Estimations of vulnerabilities for the following year (quarterly)............. 127
Figure 6-16. Adaptive estimations of vulnerabilities for the following year (quarterly) 128
Figure 6-17. Adaptive estimations of vulnerabilities for the following year (quarterly) 128
Figure 7-1. The plots for individual vulnerability categories data fitted to the models for Windows XP 135
Figure 7-2. The plots for individual vulnerability categories data fitted to the models for Windows 2000 136
Figure 7-3. Plots for individual vulnerability categories data fitted to the models for Red Hat Fedora.. 137
Figure 7-4. Fitting Internet Explorer's Vulnerabilities by Category to AML 138
Figure 7-5. Fitting Firefox's Vulnerabilities by Category to AML............................... 139
Figure 7-6. Apache by severity level vulnerabilities fitted to the AML........................ 140
Figure 7-7. IIS by severity level vulnerabilities fitted to the AML 140
Figure 7-8. Vulnerabilities of Internet Explorer classified by severity level................. 141
Figure 7-9.Vulnerabilities of Firefox classified by severity level 142
Figure 7-10: Vulnerabilities by category in Internet Explorer, Firefox and Mozella..... 143

LIST OF TABLES

Table 2-1: Vulnerability density versus defect density measured for some software systems 19
Table 4-1: Goodness of fit test results for the logistic model (AML), showing parameter values 48
Table 4-2: Goodness of fit test results for linear fit (LM), showing parameter values..... 54
Table 4-3. Market share, Number of vulnerabilities and release dates of some web servers 56
Table 4-4. Goodness of Fit Results for the examined web server's datasets, fitted to the logistic model (AML) 61
Table 4-5. Goodness of Fit Results for the examined web server's datasets, fitted to the linear model (LM) 64
Table 4-6. Goodness of Fit Results for the examined web browsers datasets on the logistic model (AML) 70
Table 4-7. Goodness of Fit Results for the examined web browsers datasets on the linear model (LM) 74
Table 4-8. χ^2 Goodness of fit test results for fitting the Effort-Based Model to vulnerabilities of some versions of Windows 77
Table 4-9. χ^2 Goodness of fit test results for fitting the Effort-Based Model to some HTTP servers 79
Table 5-1. Some attributes of the examined operating systems 93
Table 5-2. Goodness of Fit and AIC results for Windows 95 goodness of fit 96
Table 5-3. Goodness of Fit and AIC results for Red Hat Linux 6.2 99
Table 5-4. Goodness of Fit and AIC results for Windows XP 101
Table 5-5. Goodness of Fit and AIC results for Red Hat Fedora 103
Table 6-1: Estimating the number of vulnerabilities using the static approach 111
Table 6-2: The Average Error (AE) and Average Bias (AB) for estimations using the dynamic approach 113
Table 6-3. Average Error and Average Bias for Prediction of Ω 120
Table 6-4. Average Error and Average Bias for Prediction After 65% of Time Elapsed 120
Table 6-5. Average Error and Average Bias for Prediction of Ω (Adaptive Estimation) 122
Table 6-6. Average Error and Average Bias for Prediction of Ω after use of 65% of the Data Is Used (Adaptive Estimation) 123
Table 6-7. Absolute Average Error and Absolute Average Bias for Non-Adaptive Short Term Prediction (in vulnerabilities per year) 127
Table 6-8. Absolute Average Error and Average Bias for Adaptive Short Term Prediction (in vulnerabilities per year) 129
Table 7-1. Windows XP Goodness of fit results 136
Table 7-2. Windows 2000 Goodness of fit results 136
Table 7-3. Red Hat Fedora Goodness of fit results 137

Table 7-4 Goodness of Fit Test Results for Total Number of Vulnerabilities of categorized vulnerabilities 139
Table 7-5.Web servers vulnerability Goodness of fit results for data classified by severity fitted to the AML 141
Table 7-6. Goodness of fit results for IE, and Firefox and datasets classified by severity level fitted to the AML 142
Table 7-7. Vulnerabilities Type vs. Severity for Windows 2000 145
Table 7-8. Vulnerabilities Type vs. Severity for Windows XP 145
Table 7-9. Vulnerabilities Type vs. Severity for Red Hat Fedora 145
Table 7-10. Vulnerabilities Type vs. Severity for Apache 146
Table 7-11. Vulnerabilities Type vs. Severity for IIS 146

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Connectivity of computing systems through the internet has brought the security of software systems under intense scrutiny. Until recently, much of the work on security has been qualitative, focused on detection and prevention of vulnerabilities in these systems. This study has attempted to provide a basis for developing systematic quantitative approaches for characterizing security. Quantitatively security can be characterized by factors such as the number of vulnerabilities present in a particular software system and the related discovery, remedy and exploitation processes. A software *vulnerability* may be defined as a "defect which enables an attacker to bypass security measures" [65]. Alternatively, a vulnerability may be defined as a defect "which enables an attacker to bypass security measures," [71]. Hence, a software system with a smaller number of vulnerabilities can be viewed as less risky than a software system with a larger number of vulnerabilities. In this study the quantitative characterization of security focuses on the vulnerability discovery process. The analysis starts by observing the cumulative number of vulnerabilities plotted against calendar time, and then appropriate models are identified able to capture the repeated behavior. This study has satisfactorily established the feasibility of quantitative characterization of security risks in terms of vulnerabilities present in a software system.

Several models have been in use in the field of software reliability engineering where the number of defects and the defect finding rate can be projected using the metrics such as the software size and the defect density, and by using the Software Reliability Growth models (SRGMs). We have analyzed the available vulnerability datasets for major security-critical software applications to identify trends to develop Vulnerability Discovery Models (VDMs). Fortunately, software vulnerability databases are openly available and are becoming more standardized. This availability of real data has helped us address some major questions. For example, based on available datasets, do we note any similarity of behavior when we observe new vulnerability discovery rates for different systems so that we can develop suitable models?

Researchers in software reliability engineering have developed methods that use plots of cumulative number of defects found against time. Methods have been developed to project the mean time to failure (MTTF), mean time between failures (MTBF), etc. [49][57] [58]. However, very little quantitative work has been done to characterize security vulnerabilities along the same lines. One major difference makes analyzing the vulnerability discovery rate more difficult—namely, that throughout the lifetime of the software after its release, it encounters a wide variation in the effort devoted to identifying and exploiting the vulnerabilities. Consequently, a security analysis has to assume that failures are caused intentionally, resulting in security failure rates that depend on life cycle phase of the product, its popularity, and attackers' familiarity with the system. This is in contrast to the traditional dependability analysis to date, which usually assumes that the failures are caused by random events that cause hardware or software malfunction.

A new static software metric has recently been proposed that depends on the number of vulnerabilities, termed Vulnerability Density [63], which is analogous to defect density in software reliability engineering. In software engineering, it has been noted that for a given state of technology, the defect density of software systems at release tends to fall within a certain range [46]. This had made defect density a suitable measure for managing software projects. This research attempts to evaluate and examine the stability of the vulnerability density metric. If this metric possesses the required attributes, then it is potentially measurable and can be used for assessing risks in various software systems.

Quantitative methods need multiple components; here we use these two - the rate of vulnerability discovery and the vulnerability density metric. These two can be used to describe the security status of a particular software system.

In this work, we aim to establish a standard approach for assessing the security of software similar to the one already established in the area of software reliability engineering. This will allow practitioners to be able to integrate security into the phases of the development of software systems and during the lifetime of the software system.

The number of vulnerabilities in a software system is a key security attribute of the software. Unfortunately, consumers who make decisions in purchasing a new software system can be misled by marketing and popular views and often can not determine which software is likely to be more secure. The consumers need quantitative assessment tools to get an estimate of the number of vulnerabilities to compare and choose new software systems. An approach for estimating the number of vulnerabilities is proposed here and its feasibility is examined.

Recently a few vulnerability discovery models have been proposed, including those which were developed as a part of this research. The proposed models will be validated using several vulnerability datasets for major systems. The results of fitting the data to the models will be analyzed. An acceptable goodness of fit results indicates that the models are complying with the data; however, this does not necessarily mean good prediction capabilities. Therefore, the prediction capability of the vulnerability discovery models is separately examined.

In the initial part of this research, we treat software vulnerabilities as if they are equal. However, the analysis would be richer if vulnerabilities are categorized; so, the proposed logistic model will be validated using data obtained by partitioning the vulnerabilities into well defined categories. Vulnerabilities can be classified in several ways. Here, we classify vulnerabilities using two different attributes, using categories based on origin, and by the severity level. The relationship between categories and severity level was analyzed to see if some categories tend to be associated with specific severity levels.

The outcome of this research will allow development of guidelines for developers' practices that will improve the security of their software systems by identifying the most vulnerable part of the software and the common mistakes introduce security faults. Furthermore, the results can be used to identify the types of vulnerabilities more likely to be encountered.

External factors that can increase or decrease risks, such as the attackers' reward factor, need to be examined. By looking at changes in vulnerability discovery rates, we can attempt to get a good perspective on the nature of vulnerability discovery and exploitation, which will allow better risk assessment.

In software engineering, one cannot expect to find a single solution that is immediately applicable to every software system since software development and test environments vary in nature. The developers should choose from among a range of approaches for quantitative assessment; for each case a specific estimation approach may be optimal. Several software reliability growth models and application methodologies exist, one needs to choose the approach that is most likely to work in a specific case. In this research multiple models and approaches are considered and evaluated using actual data.

1.2 MOTIVATION

Several quantitative methods have been proposed by researchers which can be used to estimate the number of defects in a particular release version of software. These methods can be used to project the resources needed to achieve the target reliability levels by the release date. Quantitative characterization of vulnerabilities can be similarly used to develop guidelines for allocation of resources for security testing, scheduling, and development of security patches. Furthermore, it can be utilized by the users for assessing risk and estimating needed redundancy in resources and procedures to handle potential breaches. These measures help in determining the resources that need to be allocated to test a particular piece of software. It would be very valuable to create similar methods specifically for addressing the potential vulnerabilities present. Because the software system vulnerabilities—the faults associated with maintaining security requirements—are a special type of software defects, a similar measure for estimating security vulnerabilities is warranted.

Quantitative assessment ought to be integrated into the testing process as well. During the testing phase, estimating the number of vulnerabilities in a system will support the critical decision to stop testing for vulnerabilities in order to release a stable version. Vendors are usually faced with deadline pressures to release the new software system;

consequently, a compromise between achieving a high security level and meeting the deadlines must be made. Research studies have confronted this issue in the past. For example, in [18] Beattie et al. have suggested that administrators should delay applying security patches to ensure a system's stability. Clearly such decisions can be better arrived at by using quantitative rather than subjective methods.

Software developers have struggled with the discovery of vulnerabilities, because of the risk that they represent, and the effort needed to develop patches to fix newly discovered vulnerabilities. The developers can overestimate the resources needed to maintain the system's security and allocate a too high fraction of the resources to maintain the system, or, alternatively, they may underestimate the risk and allocate insufficient resources and thus fall behind in developing patches.

In a typical situation vulnerabilities bulletin boards and databases post information about new vulnerabilities, and administrators review newly announced vulnerabilities. Typically, administrators need to decide whether to disable parts of the system that contains the new vulnerability and sacrifice some functionality. Alternatively, they can wait for new patches to be applied. Nonetheless, patches developed in haste can sometimes result in the instability of the system, either because the patch itself contains a vulnerability or because the patch conflicts with the existing configurations at certain locations.

Just as it is impossible to have a software system without defects, security flaws will inevitably be discovered after the software's release. Consequently, developers will need to schedule updates and patches for fixing the software systems; carefully planning for this process will help in utilizing resources efficiently.

1.3 CONTRIBUTIONS

This work proposes some vulnerability discovery models that model how vulnerabilities are discovered. The process is analyzed and discussed to identify factors influencing the process.

The models' prediction capabilities are examined to predict the total number of vulnerabilities and vulnerability discovery rates. The vulnerabilities classified by coding error type and by severity are analyzed; and the applicability of the model to these classified datasets is evaluated.

We compare several existing vulnerability discovery models and identify the strengths and weaknesses of each model. The comparison identifies which is the best applicable model based on actual vulnerability datasets.

The work presents an novel analysis of the vulnerabilities discovery process and the factors impacting the vulnerabilities discovery. We go further to look into individual vulnerabilities classified by category and we show that there is a relationship between taxonomy and severity levels. The results show that some vulnerability categories of tends to carry more risk than other categories, a fact which can help designing effective test cases.

1.4 RELATED WORK AND LITERATURE REVIEW

1.4.1 BACKGROUND

Traditional research in computer security has focused on topics like the security models (e.g. [26]), access control including encryption schemes (e.g. [67]), and intrusion detection systems (e.g. [35]). There have been significant advances in these and other related research areas of security in the past three decades. However, before applying a new security technique, there is a need to quantitatively assess how effective the scheme will be. The main focus of this work is to introduce effective, stable, and practical security assessment methods and tools.

Software security community admits that it is practically impossible to build a sizeable software system that is perfectly secure; thus, currently some research is directed toward fault tolerance techniques to ensure that the system survives attacks and malfunctions despite malicious activities [66]. Consequently, there is growing interest among researchers in establishing some applicable quantitative characterization of systems security that can quantitatively validate the efficacy of proposed system designs objectively.

Presently, most attempts at validation of security have been qualitative, focusing more on the process used to build a system that should be secure. Much of the research aimed at quantitative validation of security has usually been based on formal methods (e.g. [38]). This work differs by focusing on the process of vulnerability detection as an important factor for determining the risk associated with using a software system.

The analogy between security and software reliability is very useful, because the latter has some well established quantitative tools. Approaches to evaluate the software security from a reliability point of view has been suggested, for example Littlewood et al. in their work [23][41], have considered the relationship between reliability and security. At the same time, however, they have acknowledged that there are some significant differences between software reliability and security. Littlewood et al. have proposed the use of effort rather than time to characterize the accumulation of vulnerabilities, however, they did not specify how to assess effort. The work of Littlewood et al.

emphasizes the importance of quantitative evaluation as an evolving area of research, which this research is attempting to develop.

In the following sections we shall look into related work. First, we examine different definitions of the term software vulnerability, then we will discuss approaches for quantitative assessment of security, which leads to vulnerability modeling.

1.4.2 Definition of Software Vulnerability

Several definitions of the term software vulnerability are available in the literature. Definitions can belong to one of three classes: fuzzy, access control or state based definitions [36]. One access control definition is offered by Schultz et al. is: "a vulnerability is defined as a defect which enables an attacker to bypass security measures" [71]. This definition is the one we prefer in this reliability and security context. Krusl has defined software vulnerability with respect to the security policy as follows: "A software vulnerability is an instance of an error in the specification, development, or configuration of software such that its execution can violate the security policy" [36]. Pfleeger has defined vulnerability as "a weakness in the security system that might be exploited to cause loss or harm" [65].

Some sources considered giving alternative definitions. The book *The Data & Computer Security Dictionary of Standards, Concepts, and Terms* by Longly and Shain [43] defines computer vulnerability as:

- "A weakness in automated systems security procedures, administrative controls, internal controls, etc., that could be exploited by a threat to gain unauthorized access to information or to disrupt critical processing.", or

- "A weakness in the physical layout, organization, procedures, personnel, management, administration, hardware, or software that may be exploited to cause harm to the ADP system or activity. The presence of a vulnerability does not itself cause harm. A vulnerability is merely a condition or set of conditions that may allow the ADP system or activity to be harmed by an attack.", or
- "A weakness or error existing in a system. The attack or harmful event, or the opportunity available to a threat agent to mount that attack".

It should be kept in mind that vulnerabilities are flaws that can be counted. Thus the term "vulnerability", it is not similar to "reliability" or "availability" which are probabilities and thus take a value between zero and one.

1.4.3 Quantitative Assessment of Software Security

Most of the conventional security studies involving vulnerabilities have been qualitative, with studies of how specific types of vulnerabilities impact security, such as those critical vulnerabilities causing denial of service attacks [33], or specific types of vulnerabilities such as those caused by buffer overflow [39]. Very little quantitative work has been done to characterize security vulnerabilities along the same lines as general defects. In [11] the authors propose a metric termed Software Vulnerability Index (SVI). SVI is calculated using some predefined rules as a heuristic that can take values between 0 and 1. SVI aims to assess the vulnerability of a specific system using some preset conditions and rules based on detailed configurations of that system. Also, some studies have focused on operational security methodologies that can quantify the amount of security provided by a particular system-level approach, using techniques such as attack trees [60]. Also, in [25] Dacier et al., another approach was proposed. They suggested an operational security evaluation in a UNIX system using Markov modeling. These are examples of real-time evaluation of the security of systems.

Vulnerability density, a quantitative measurement, was suggested by Ounce labs [63], it has referred to it as V-density; however, it is proprietary and not clearly defined. In this research, we define it, apply it to some software systems, and investigate its feasibility. Furthermore, this research studies density of defects and compares it with density of vulnerabilities.

Researchers in software reliability engineering have developed methods that use plots of the cumulative number of defects found over time. Methods have been developed to project the mean time to failure (MTTF) and mean time between failures MTBF, etc. That will result after a specific testing period [44][50][53]. Software defect density [49][57][58] has been a widely used metric to measure the quality of a program and is often used as a release criterion for a software project. Since operating system vulnerabilities—the faults associated with maintaining security requirements—are considered to be a special kind of software defect, a similar measure for estimating security vulnerabilities is warranted.

This work differs by focusing on the process of how vulnerabilities are detected as an aspect of how to look at the overall security of a software system. Hence, factors like the rate of vulnerability discovery is analogous to Assessment of reliability of a software system based on the rate of defect detection. Software Reliability Growth models (SRGMs) are valuable widely used tools to assess the reliability of software systems [44]. This analogy between security and reliability is investigated to evaluate how it could provide some security assessment tools; for example, by examining Vulnerability Discovery Models, which correspond to SRGMs.

1.4.4 Modeling Vulnerabilities and Incidents

Modeling vulnerabilities discovery process has recently attracted researchers as they recognize the need to analyze how the vulnerability discovery process affects the security of software systems and make forecasts that can be used for planning. A study by Rescorla has examined vulnerability discovery trends in some software systems. He has fitted a linear model (the linear model is considered quadratic if used with cumulative data) and an exponential model to vulnerability data; however, the fit was insignificant in both cases [69].

Anderson [14] proposed a model for a vulnerability-finding rate using a thermodynamics analogy. The applicability of the model proposed by Anderson [14] has not yet been considered, nor has any comparison of the VDMs been carried out, his model is examined in this thesis.

A technical report by Gopalakrishna and Spafford have studied common trends among vulnerabilities; however, no models were suggested [31]. Recently, in [64] Ozment has suggested using SRGMs as Security Growth models. The author has fitted some OpenBSD vulnerability data to some software growth models, namely, Musa's logarithmic model, and a geometric model. Ozment has found that Musa's logarithmic model was the most accurate in next point estimation.

Arbach et al. [15] and Browne et al. [20] have examined several systems using the incidents reported by CERT. Browne et al. examined some actual statistical data to study the trends occurring in incidents. They have suggested that the cumulative number of incidents is linearly related to the square-root of time. The rates of Exploitations were examined by Browne et al. in [20], where *Vulnerability exploitation models* (VEMs) were first presented; they examined the exploitation rate of some vulnerabilities, they also presented a modeling scheme. Shim et al. [76] have applied a software reliability growth model to the incidents data. The security intrusion process has also been examined by Johnson and Olovsson [35] and Madan et al.[45].

During this research we have identified and examined major open source and closed source software systems. Recently there have been a number of comparisons between the two paradigms [14] [58]. This is not, however, the focus of this research. Rather, our work is concerned with the entire process of vulnerability detection, focusing on an analysis that can lead to a reduction in the number of vulnerabilities in a system and a method to cope effectively with the existence of such vulnerabilities.

In [15], researchers have taken some common types of vulnerabilities and suggested to designers how to improve their techniques in order to reduce the number of vulnerabilities. These researchers have evaluated the ranges of defect densities typically encountered during different phases of the software life cycle using data from available sources [53]. Other researchers have focused on modeling and designing tools that make some security assessment possible [71].

1.5 ORGANIZATION OF THE DISSERTATION

In the next chapter, we discuss the static metric vulnerability density. In chapter 3, several vulnerability datasets are previewed, and then the vulnerability discovery models are presented. In chapter 4, the models' goodness of fit is examined. In chapter 5, we present the comparison between several vulnerability discovery models. In chapter 6, the prediction capabilities are measured for some of the models and some prediction techniques. In chapter 7, we look into categorized vulnerability datasets and we validate the model on them, and discuss some observations. Finally, chapter 8 concludes and discusses possible directions for future work.

CHAPTER 2

VULNERABILITY DENSITY

A new metric, *vulnerability density,* describes one of the major aspects of security. Vulnerability density is a normalized measure, given by the number of vulnerabilities per unit of code size.

To measure the code size, we have two options. First, we can use the size of the installed system in bytes. The advantage of this measure is that information is readily available; however, this measure will vary from one installation to another. The second measure is the number of source lines of code. Here, we chose this measure for its simplicity and its correspondence to the defect density metric in the software engineering domain. A conceptually similar notion is defect density[46], a measurement that has been used for several years and that has become a common measurement in the field of software reliability and dependability. We now present a definition of vulnerability density (V_D):

Definition: Vulnerability density is the number of vulnerabilities in the unit size of a code.

The vulnerability density is given by

$$V_D = \frac{V}{S} \qquad (1)$$

where S is the size of the software and V is the number of vulnerabilities in the system. Following the common practice in software engineering, we consider one thousand source lines as the unit code size. When two systems, one large and one small, have the same defect density, they can be regarded as having similar maturity with respect to dependability. In the same manner, vulnerability density allows us to compare the quality of programming in terms of how secure the code is. If the instruction execution rates and other system attributes are the same, a system with a higher defect or vulnerability density is likely to be compromised more often.

Estimating the exact vulnerability density would require us to know the number of all the vulnerabilities of the system. Consequently, we define another measure in terms of the known vulnerabilities.

The *residual vulnerability density* (V_{RD}) is given by:

$$V_{RD} = V_D - V_{KD} \tag{2}$$

It is actually the residual vulnerability density (depending on vulnerabilities not yet discovered) that contributes to the risk of potential exploitation. Other aspects of the risk of exploitation include the time gap between the discovery of a vulnerability and the release and application of a patch. Our focus is mainly on vulnerabilities and their discovery.

The goal is to probe the suitability of vulnerability density and vulnerabilities as metrics that can be used to assess and manage components of the security risk.

2.1 APPLICATIONS OF VULNERABILITY DENSITY

Vulnerability density can be used to compare software systems within the same category (e.g., operating systems, web-servers, etc.). Vulnerability density can also be used in estimating the number of residual vulnerabilities in a newly released software system, given the size of the software. A newly released software system should have vulnerability density that is close to that of its comparable predecessor, given that it is designed assuming the same resources and technology. To acknowledge changes to improvements in the design of the new software or improvement in the attackers' capabilities, a range for the vulnerability density is considered. This application of vulnerability density can be utilized by users to assess the risk when considering the purchase of a new software system. Because software systems do not come with a definite number showing the risks associated with the usage of the product or how vulnerable it is, vulnerability density gives an approximation of the number so users can use it as a guideline.

Developers can use vulnerability density in better planning for their testing process and in order to answer questions such as when to stop testing, when to release the software and what the risks associated with its early release are. Testing is very costly both in terms of resources and time needed for competition purposes. Consequently, when the vendor has some objective assessment of potential vulnerabilities in a software system, the vendor can decide that whether the system is secure enough to be released, and the developers can reduce the cost and be better able to compete with existing products.

Another application of vulnerability density is for maintenance planning. This integrates the use of vulnerability density with vulnerability discovery models, resulting in better estimations of the rate of vulnerability discovery, thereby enabling developers to obtain an objective estimation of the numbers of vulnerabilities that will be discovered within some future period of time. This is what can also be used by users and administrators to decide whether to use security patches or whether it is safe to delay the patching in order to avoid destabilizing the software system, a trade-off that has raised some discussion recently [24].

2.2 MEASURING VULNERABILITY DENSITY OF SOME SOFTWARE SYSTEMS

Table 2-1 presents values of the known defect density D_{KD} and known vulnerability density V_{KD} based on data from several sources [51] [56] [61] [70] [62] as of January 2005. Windows 95, 98 and XP are three successive versions of the popular Windows client operating system. We also include Windows NT and Windows 2000, which are successive versions of the Windows server operating systems.

The known defect density values for Windows 95 and Windows 98 client operating systems are 0.3333 and 0.5556 per thousand lines of code, respectively. The high defect density for Windows XP is attributed to the fact the data belongs to the beta version of Windows XP. We can expect the actual released version of Windows XP has significantly fewer defects. The defect density values for Windows NT and 2000 are 0.625 and 1.8, respectively.

The Known Vulnerabilities column gives a recent count of the vulnerabilities discovered since the release date. We note that the vulnerability densities of Win 95 and 98 are quite close. The known vulnerability density for Win XP is 0.0022, much lower than the values for the two previous Windows versions. This is due to the fact that at this time V_{KD} represents only a fraction of the overall V_D. We can expect the number to go up significantly, perhaps to a value more comparable to that of the two previous versions.

We note that the vulnerability density for Windows NT 4.0 is about three times that of Win 95 or Win 98. There are two possible reasons for this. Since NT is a server, a larger fraction of its code involves external access, resulting in about three times the number of vulnerabilities. In addition, as a server operating system, it must have gone through more thorough testing, resulting in the discovery of more vulnerabilities. Windows 2000 also demonstrates nearly as many vulnerabilities as in Windows NT 4.0, although due to its larger size, the vulnerability density is lower than that of NT 4.0.

Table 2-1: Vulnerability density versus defect density measured for some software systems

Systems	*Msloc*	*Known Defects*	*Known Defect Density (per Ksloc)*	*Known Vulnerabilities*	*V_{KD} (per Ksloc)*	*V_{KD}/ D_{KD} Ratio (%)*	*Release Date*
Windows 95	15	5000	0.3333	50	.0033	1.00%	Aug 1995
Windows 98	18	10000	0.5556	66	.0037	0.66%	Jun 1998
Windows XP	40	106500	2.6625	88	.0022	0.08%	Oct 2001
Windows NT 4.0	16	10000	0.625	179	.0112	1.79%	Jul 1996
Win 2000	35	63000	1.80	170	.0049	0.27%	Feb 2000
R H Linux 6.2	17	2096	0.12329	118	.00694	5.63%	Mar 2000
R H Linux 7.1	30	3779	0.12597	164	.00547	4.34%	Apr 2001
Fedora Red Hat	76	-	-	154	.00203	-	Nov 2003

One significant ratio to examine is ${}^{V_{KD}}\!/_{D_{KD}}$, which gives the fraction of defects that are vulnerabilities. Longstaff [42] hypothetically assumed that vulnerabilities may represent 5% of the total defects. Anderson [13] assumed a value of 1%. Our results show that the values of the ratio are 1.00% and 0.66% for Win95 and Win98, respectively. For

Windows XP, the number of known defects is given for the beta version and is therefore higher than the actual number at release. In addition, since it was released last, a smaller fraction of XP vulnerabilities have been found thus far. This explains why the ratio of 0.08% for XP is significantly lower. We believe that this should not be used in a comparison with other Windows versions. It is interesting to note that the ratio of 1% assumed by Anderson is within the range of the values shown in Table 2-1. Windows 2000 was an update of NT, with a significant amount of added code, much of which did not deal with external access, thus accounting for its relatively low ratio.

In Table 2-1 [53][68], we observe that although the code size for Linux 7.1 is twice as large as that of Linux 6.2, the defect density and vulnerability density values are remarkably similar. We note that the V_{KD} values for the two versions of Red Hat Linux are significantly higher than for Windows 95 and 98, and are approximately in the same range as for Windows 2000. However, V_{KD} alone should not be used to compare the two competing operating system families.

It is not the discovered vulnerabilities but rather the vulnerabilities remaining undiscovered that form a significant component of risk. In addition, the exploitation patterns and the timing of the patch releases also impact the risk. The V_{KD} value for Red Hat Linux 7.1 can be expected to rise significantly in the near future, just like those of Windows XP. It is interesting to note that the ${}^{V_{KD}}\!/_{D_{KD}}$ ratio values for Linux are close to the value of 5% postulated by Longstaff [42].

In systems that have been in use for a sufficient time, V_{KD} is probably close to V_D. However, for newer systems we can expect that a significant number of vulnerabilities will be discovered in the near future. For a complete picture, we need to understand the process that governs the discovery of the remaining vulnerabilities, as discussed in the next sub-section.

2.3 CONCLUSION

Vulnerability density is discussed in this chapter as an important metric that can characterize vulnerabilities in a piece of software in a normalized manner, and therefore, it considers the size of the software. In many cases larger software systems may reflect more functions and features. Hence, it provides a fair base to compare risks associated with different software systems.

Vulnerability density can be used as a static metric to predict vulnerabilities in newly released system, as it can be compared to its comparable predecessor; however, other factors should be considered when the metric is applied such as changes in programming techniques, development process, and the technology used.

Table 2-1 shows that vulnerability density can be compared to defect density in many cases. Defect density is a well established metric used in many software engineering disciplines. The stability shows a strong relationship between vulnerabilities and the total defects population.

There are some limitations on using vulnerability density, such as that some software systems change consistently and, therefore, the new code added may have more vulnerabilities, as it has not been tested as rigorously as the older more mature code has.

CHAPTER 3

VULNERABILITY DISCOVERY TRENDS AND VULNERABILITY DISCOVERY MODELS

3.1 INTRODUCTION

In this chapter we plot several vulnerability datasets against time, preview the plots, comment, and identify common patterns and observations.

After the datasets are plotted and discussed, three vulnerability discovery models are presented based on observations and common patterns identified in the trends. The models are two time-based models (the logistic model and the linear model) and an effort-based model, which is an exponential model.

3.2 VULNERABILITY TRENDS

Considering vulnerability data plotted against calendar time, we consider data from complex software systems that are subject to connectivity challenges: Operating Systems, Web Servers, and Web Browsers.

The systems previewed include some open source systems, like Linux Red Hat versions 6.2, 7.1, and Fedora. Also, we considered Apache web server versions 1 and 2, and the open source browser Firefox. Also, the proprietary systems considered are Windows 95, 98, XP as client operating systems, some server operating systems Windows NT4 and Windows 2000. Also, Solaris operating systems versions 7, 8, and 9 is considered. Besides, propriety web server IIS 4 and IIS 5, also the Internet Explorer was considered as a proprietary web browser.

As we can see these systems were selected because of the diversity in how they were developed, how they function and when they were released. Furthermore, the versions selected are versions that are stable and have gained some popularity.

3.2.1 Data Sources

Several security bulletins report, maintain, and document software vulnerabilities in various software systems. These groups usually provide information about the vulnerability and some of its characteristics. One of these sites is National Vulnerability Database sponsored by NIST [61].

CERT and other databases keep track of reported vulnerabilities. The numbers of users of the internet and the environment in which incidents take place have made potential exploitation of vulnerabilities very attractive to criminals with suitable technical expertise. The vulnerability discovery trends are based on data acquired from the National Vulnerability Database [61] and data published by Mitre Corporation in [56]. The trends include several Windows systems. We should note that some systems were designed as server operating systems (e.g., Windows NT4.0 and Windows 2000), while others were intended for clients (Windows 95, 98 and XP).

3.2.2 Vulnerability Discovery Trends in Operating Systems

Microsoft's Windows family of operating systems has dominated the OS market for the past decade. Records of each discovered vulnerability is available. Consequently, these are good examples of complex systems to study.

Vulnerability data is collected from vulnerability databases and bulletins. The cumulative vulnerabilities are distributed among various factors, such as calendar time or equivalent effort. Observations of patterns are then analyzed with consideration of external factors. Several models have been suggested for capturing these patterns. The models will then be fitted to the data and examined using statistical goodness of fit tests.

Consecutive versions of software systems often share some vulnerabilities. So far, the shared vulnerabilities have not been addressed. Subsequently, the shared vulnerabilities need to be modeled.

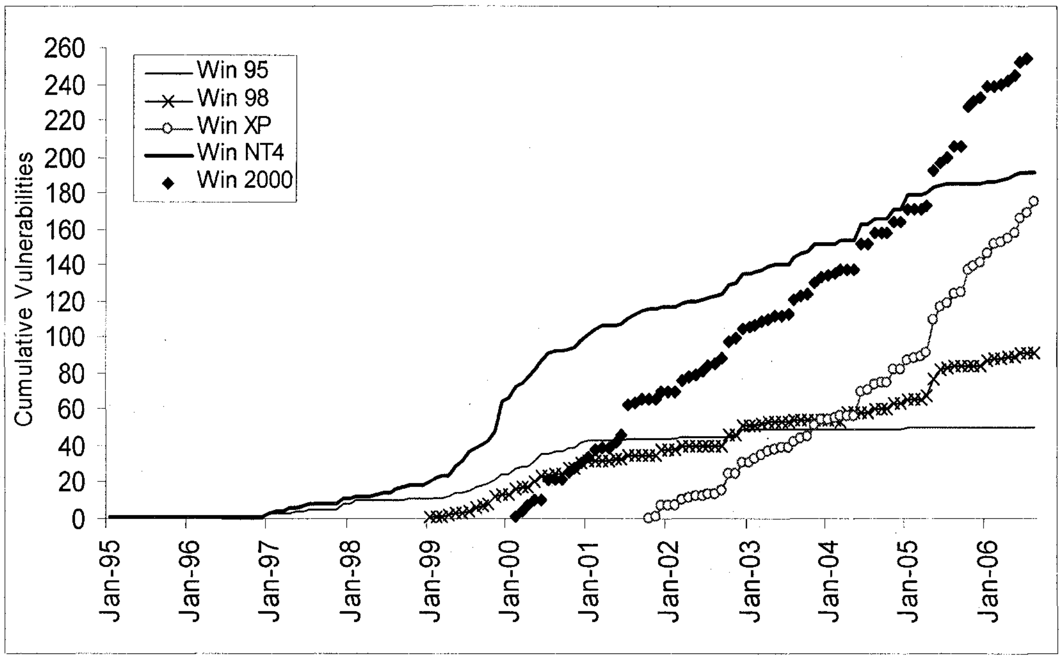

Figure 3-1. The cumulative number of vulnerabilities of some Windows systems

The plot in Figure 3-1 shows a common pattern. The discovery of new vulnerabilities begins slowly and then rises with a steeper slope. Eventually they show saturation. This behavior serves as the basis for the logistic calendar time model presented later.

Other operating systems have shown similar behaviour which supports our analysis; see Red Hat Linux shown by Figure 3-2 below. Similar but less obvious behaviour can be observed in Red Hat Fedora (see Figure 3-3).

Cumulative vulnerabilities of Solaris operating systems versions 7, 8 and 9 shown by Figure 3-4 shows a slow start followed by higher discovery rate, followed by some slowdown; however, overall the saturation was less significant when compared with Red Hat Linux in Figure 3-2.

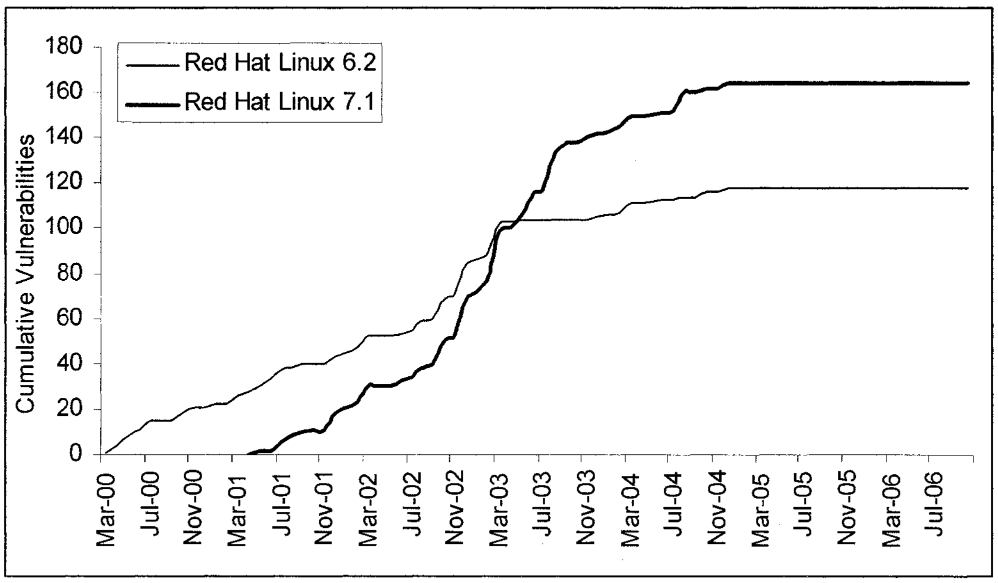

Figure 3-2. The cumulative number of vulnerabilities of Red Hat Linux versions 6.2 and 7.1

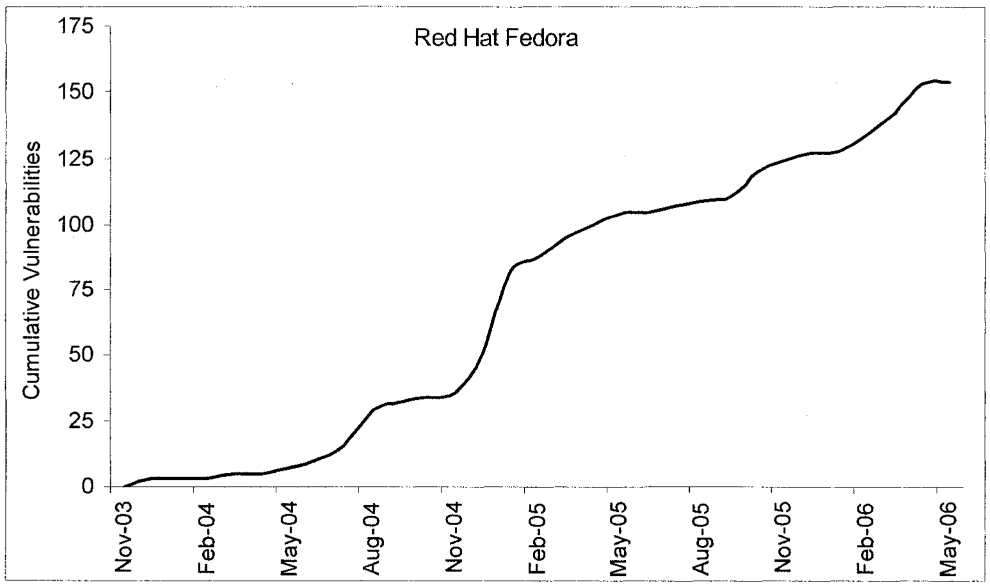

Figure 3-3. The cumulative number of vulnerabilities of Red Hat Fedora

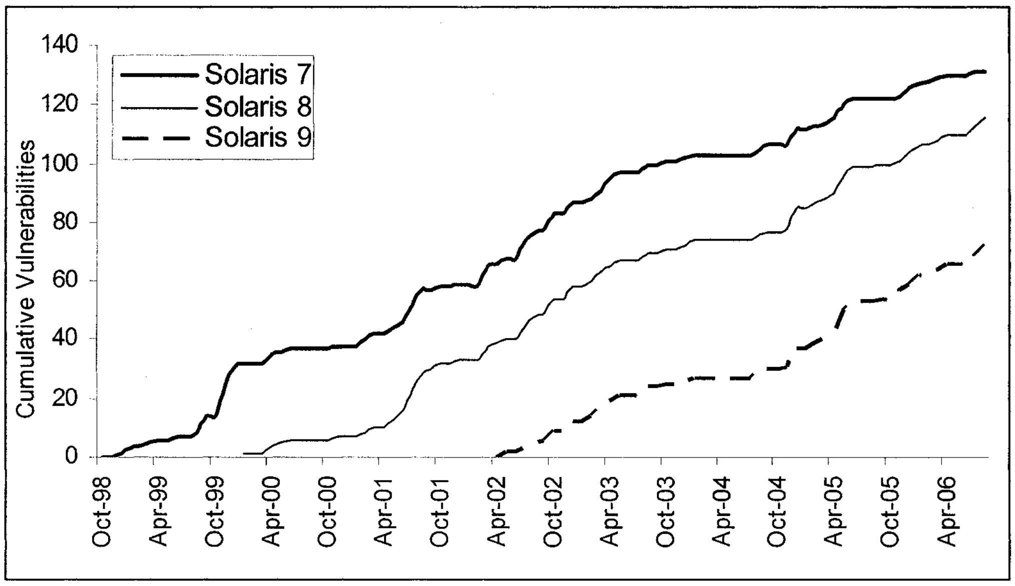

Figure 3-4. The cumulative number of vulnerabilities of Solaris operating systems versions 7, 8 and 9

3.2.3 Vulnerability Discovery Trends in Web Servers

Here we preview the data of the top two web servers in the market, Apache and IIS. Figure 3-5 below shows the trends of two versions of Apache 1.x and 2.x. The Apache 1.x trend has shown a slower beginning followed by a persistent vulnerability discovery rate, then followed by a modest saturation at the end, while Apache 2 shows an immediate steady rate that keeps its pace until August 2006. Apache 2 is still young and will probably keep the current vulnerability discovery rate for some time to come.

Figure 3-6 shows the cumulative vulnerabilities of IIS 4 and IIS 5. Both systems have shown a steady vulnerability discovery rate followed by saturation. Both systems have shown a fast discovery rate from the end of 1998 to the end of 2002; then a strong saturation appears from mid-2003 until August 2006 as the plot shows. This saturation indicates that vulnerabilities are becoming hard to discover, and at the same time that these two versions were replaced by IIS 6.0, which was released later.

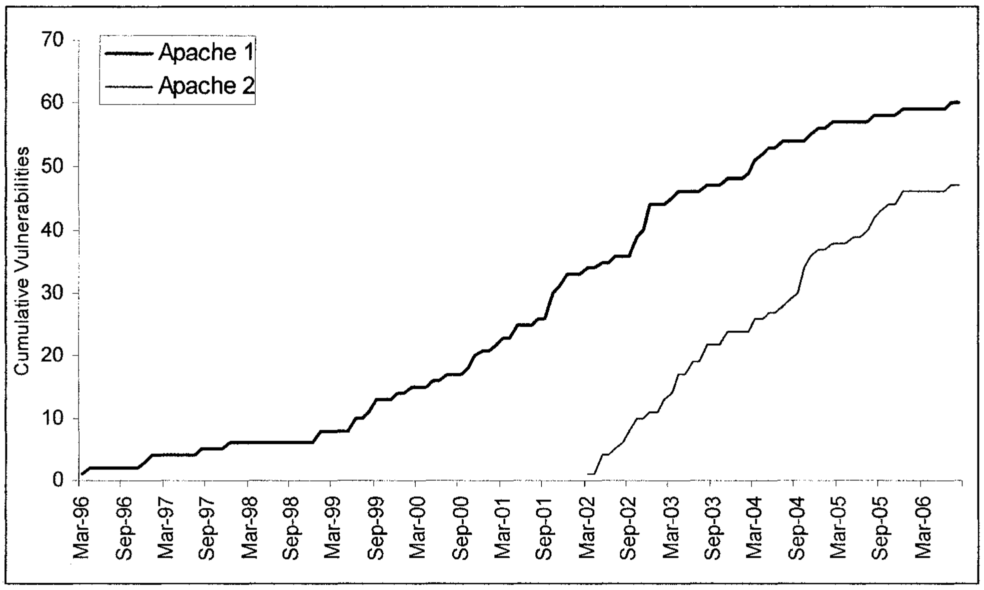

Figure 3-5. The cumulative number of vulnerabilities of Apache versions 1.x and 2.x.

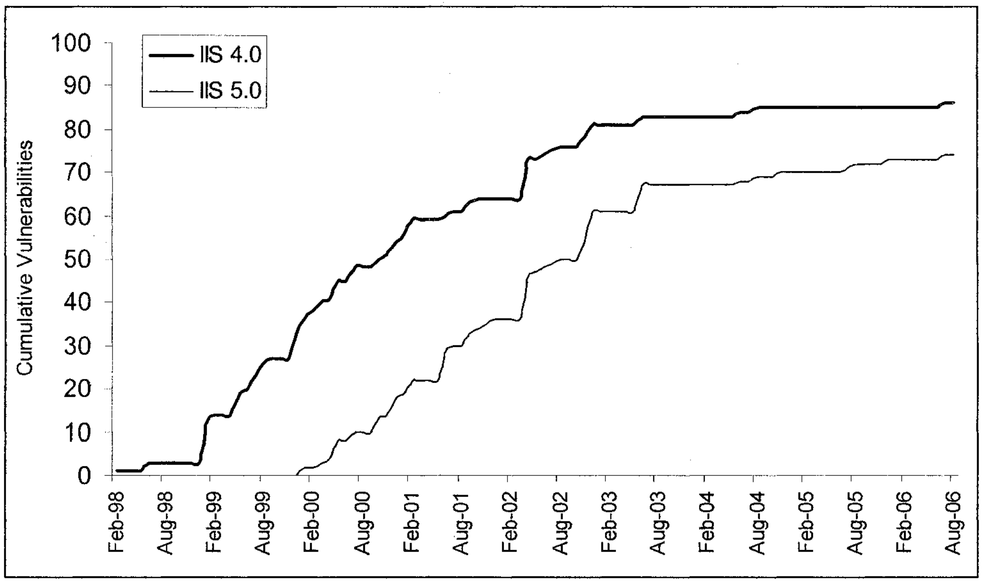

Figure 3-6. The cumulative number of vulnerabilities of Microsoft IIS 4 and IIS 5 web servers

3.2.4 Vulnerability Discovery Trends in WEB Browsers

Figure 3-7 below shows the three successive versions of Internet Explorer web browsers. First, the older IE 4.0, which started earlier, clearly shows that it has been saturated and no new vulnerabilities have been discovered in the past several years. Then, we can see that IE 5.0 shows a steady linear trend without saturation despite being on the market for several years, with a declining market share. However, the vulnerability data indicates that a significant number of the vulnerabilities discovered later in IE 5.0 are shared with IE 6.0, giving us an explanation of the reason behind the continuing linear trend. The third trend is IE 6.0, which appears as a steady linear trend showing no decline, which can be explained by a large pool of vulnerabilities still undiscovered, combined with a significant popularity and market share.

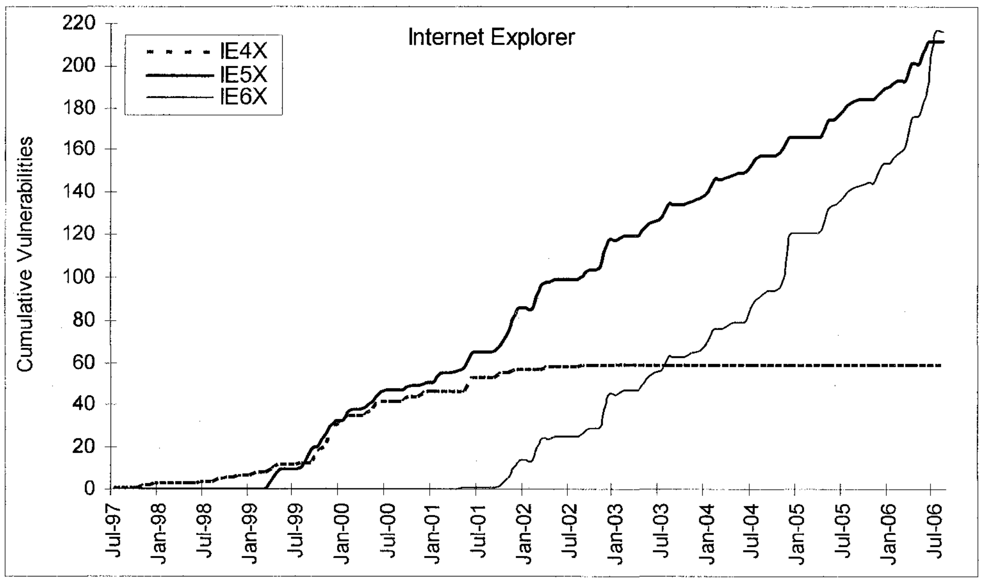

Figure 3-7. The cumulative number of vulnerabilities of Microsoft Internet Explorer versions 4, 5 and 6

3.2.5 Observations

The figures previewed here show a general trend of starting with a slow pace, followed by a faster accumulation rate for a significant period of time, usually ending saturation. However, in some cases, a linear trend dominates the picture such as in Windows XP (see Figure 3-1), and IE 6.0 (see Figure 3-7). These systems have one thing in common; they are still being actively used without a decrease in intensity, while most of the saturated software systems have seen some decline in popularity, making the market share a potential factor in the vulnerability discovery process. Hence, we need to look into potential models that can capture the patterns we have seen here.

3.3 VULNERABILITY DISCOVERY MODELS

To be able to understand the vulnerability discovery process, mathematical modeling can provide an essential solution that can enrich our understanding of how and when the vulnerability discovery rate changes using actual reported data. Therefore, here we present three vulnerability discovery models: the logistic model, the linear model, and the effort based model. In the following sections we will describe and illustrate each model.

3.3.1 THE LOGISTIC VULNERABILITY DISCOVERY MODEL

After viewing several vulnerability discovery datasets (see Figure 3-1 to Figure 3-7), we noted certain common patterns with respect to other factors. Each software system passes through several phases: the release of the system, increasing popularity, peak, and stability, followed by decreasing popularity that ends with the system eventually becoming obsolete.

During these three phases, the usage environment changes, and that impacts the vulnerability detection effort. In the beginning, there is a *learning phase*, in which the software testers (including malicious hackers and crackers) begin to understand the target system and gather the knowledge of the system needed to break into it successfully. After the learning phase, the new system starts to attract a significant number of users and to face greater challenges. This continues to increase until the operating system reaches the peak of its popularity. This is termed as the *linear phase* since the number of vulnerabilities tends to grow linearly. It will remain at that point for some time until the system begins to be replaced by a newer system. The technical support for that version and hence the frequency of update patches will begin to decline as users start to switch or upgrade to a more state-of-the-art system. At the same time, attackers will start to lose interest in the system because they are usually attracted to the most recent technology. This is termed the *saturation phase.*

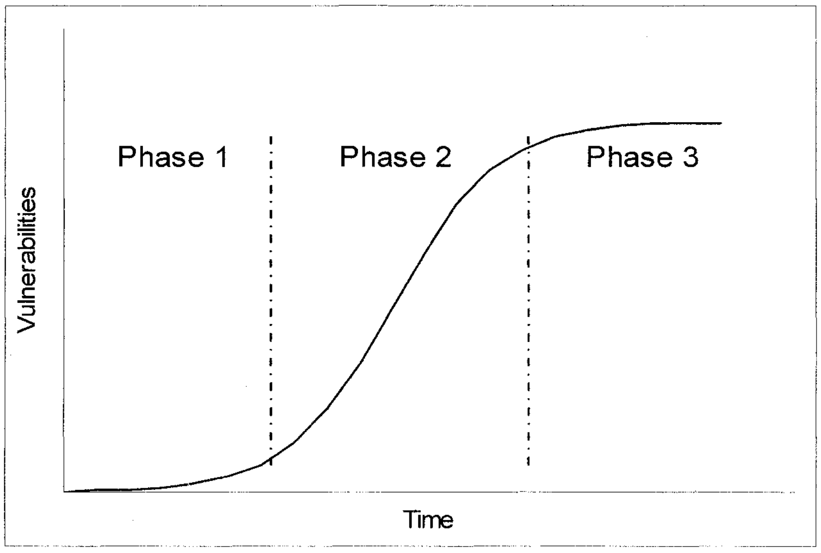

Figure 3-8.The basic 3-phase S-shaped model

The learning phase may be very brief if the adaptation of the software system is fast, or if the new system is relatively similar to its predecessor. The linear phase is the most important phase, since this is when most of the vulnerabilities will be found. Saturation may not be seen if a significant number of vulnerabilities are still present and continue to be found.

Here, we propose a model describing the relationship between cumulative vulnerabilities and calendar time. This is somewhat similar to reliability growth models; however, we recognize that there is significant change in the effort spent in finding the vulnerabilities. This effort is small in the learning phase and grows as use of the software system becomes more common, but drops again in the saturation phase.

In this model we assume that the rate of change in the cumulative number of vulnerabilities *Ω* is governed by two factors, as given in Equation (3) below. One of the factors declines as the number of undetected remaining vulnerabilities declines. The other factor rises with time to take into account the rising share of the installed base. The saturation effect is modeled by the first factor.

Let us assume that the vulnerability discovery rate is given by the differential equation:

$$\frac{d\Omega}{dt} = A\Omega(B - \Omega) \tag{3}$$

where Ω is the cumulative number of vulnerabilities, t is the calendar time, initially t=0. A and B are empirical constants determined from the recorded data. By solving the differential equation, we obtain:

$$\Omega(t) = \frac{B}{BCe^{-ABt} + 1} \tag{4}$$

where C is a constant introduced in solving Equation (3) . It is thus a three-parameter model. In Equation (4), as t approaches infinity, Ω approaches B. Thus, the parameter B represents the total accumulated vulnerabilities that will eventually be found. The model given by Equation (4) will be referred to as the Alhazmi Malaiya Logistic model (AML), which is a *time-based model.*

This model assumes that the vulnerabilities found in an operating system depend on its own usage environment. It should be noted that phase 3 may not be seen in a software system that has not been present for a sufficiently long time. In addition, in some cases phase 1 may not be significant if the initial adaptation is quick due to better prior publicity.

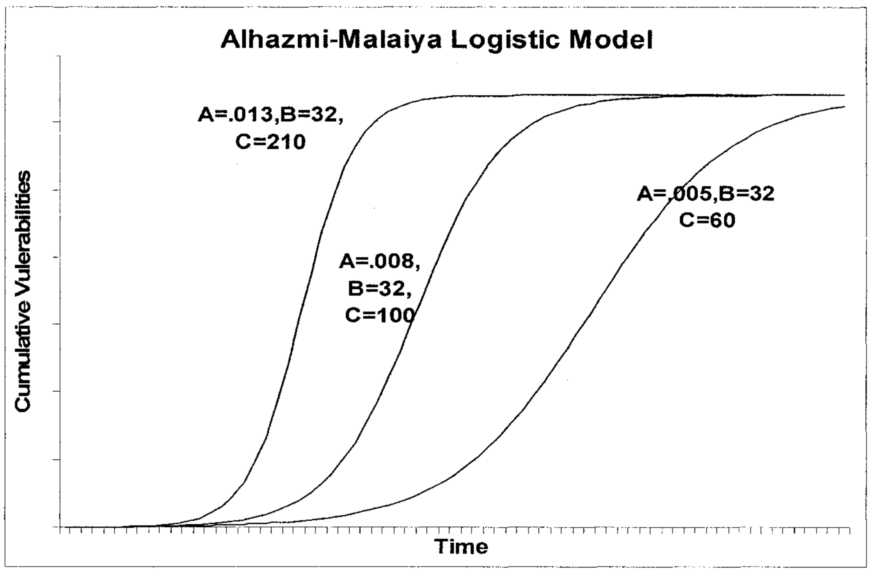

Figure 3-9. Alhazmi-Malaiya Logistic (AML) model

This model addresses the fact that the vulnerabilities found in an operating system depend on its own usage environment. It should be noted that the saturation phase may not be seen in a software system which has not been present for a sufficiently long time. Also, if the initial adaptation is quick due to better prior publicity, in some cases the early learning phase (when the slope rises gradually) may not be significant.

We can demonstrate that the maximum slope is given by $AB^2/4$, which occurs at $\Omega = B/2$. It can be shown that the two transition points are $2.63/AB$ time period apart. Therefore, the duration of the linear phase decreases as AB grows and increases as AB drops. B may be obtained by noting the size of the software and using the typical vulnerability density values of similar software. If we regard B, the total number of vulnerability to be constant, we conclude that the parameter A controls the duration of the linear phase. An estimate of the duration between the two transition points can be obtained from the data from prior software systems. We will use this mathematical result later to constrain the range of the regression parameter values in order to have a more accurate long-term projection.

Parameter C impacts the location of the first transition point. As we can observe from Equation (4), the influence of parameter C weakens as t increases. An initial estimate of the curve bends at two *transition points*. To identify the transition points, we take the derivatives of Equation (4) with respect to time t.

$$\frac{d\Omega}{dt} = \frac{AB^3Ce^{-ABt}}{(BCe^{-ABt}+1)^2} \tag{5}$$

From Equation (5), the highest vulnerability discovery rate occurs at the midpoint of Figure 3-10 at time:

$$T_m = \frac{-\ln\left[\frac{1}{BC}\right]}{AB} \tag{6}$$

The second derivative is:

$$\frac{d^2\Omega}{dt^2} = \frac{2A^2C^2B^5e^{-(ABt)^2}}{(BCe^{-ABt}+1)^3} - \frac{A^2B^4Ce^{-ABt}}{(BCe^{-ABt}+1)^2} \tag{7}$$

The second derivative exhibits a maximum and a minimum. Thus two transition points in Figure 2 are obtained by equating the third derivative to zero. The transition points occur at times t equal to:

$$T_1 = \frac{-\ln\left[\frac{2-\sqrt{3}}{BC}\right]}{AB} \quad \text{and} \quad T_2 = \frac{-\ln\left[\frac{2+\sqrt{3}}{BC}\right]}{AB}.$$

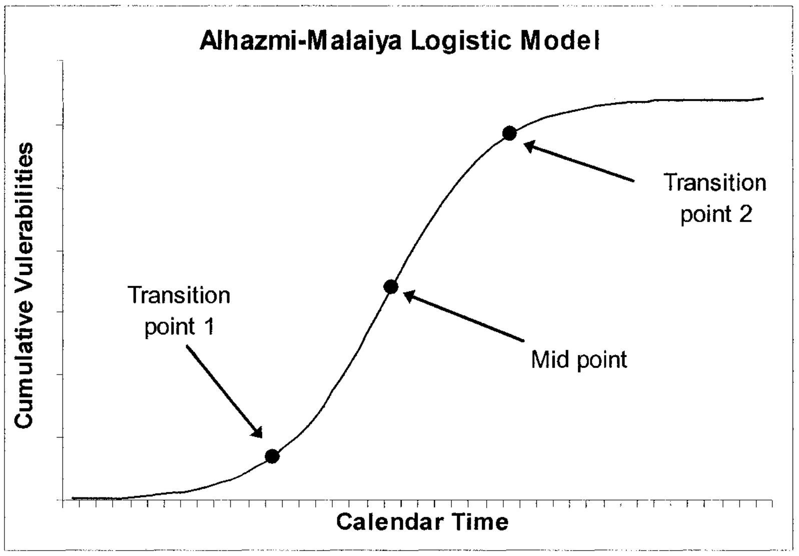

Figure 3-10. AML model with the transition points and mid point shown

3.3.2 The Linear Vulnerability Discovery Model

Occasionally, the learning phase may be very brief, especially if the adaptation of the software system is fast. The linear phase is the most important phase, since this is when most of the vulnerabilities will be found. Saturation may not be seen if a significant number of vulnerabilities are still present and continue to be found. Consequently, a dominating linear behavior was observed throughout a considerable percentage of the datasets examined; therefore, the linear model can be considered in certain cases.

This model assumes a constant vulnerability discovery rate. The linear model can be seen as an approximation to the logistic S-shaped model. The linear vulnerability model is as follows:

$$\Omega(t) = (S \times t) + k, \tag{8}$$

where S is the slope (i.e. vulnerability discovery rate) and k is a constant to adjust the model.

It is expected that the linear model will perform better in the presence of certain factors. For example, when the saturation phase has not yet been reached, especially in younger software systems where attackers have not lost their interest in finding new vulnerabilities, or when the vulnerability dataset contains a significant number of shared vulnerabilities with a later system. In this case, the data appeared to be a superimposition of two or more consecutive S-shaped models. In addition, some software systems are also expected to have a shorter learning phase due to some easily discovered vulnerabilities occurring early after the release or when there is close relationship with existing software, so that the learning phase is significantly shortened.

Alternatively, the S-shaped would appear to fit most of the datasets because it addresses the learning, linear and saturation phases. Therefore, more mature systems are expected to work better with the S-shaped model.

3.3.3 The Effort Based Vulnerability Discovery Model

Vulnerabilities are usually reported using calendar time as the main factor. The reason behind this is that it is easy to record vulnerabilities and link them to the time of discovery. However, this does not consider the changes occurring in the environment during the lifetime of the system. A major environmental factor is the number of installations, which depends on the share of the installed base of the specific system. It is much more rewarding to exploit vulnerabilities that exist in a large number of computers.

It can be expected that a larger share of the effort going into the discovery of vulnerabilities, both in-house and external, would go toward a system with a larger installed base.

Using effort as a factor was first discussed in [22][23]. However, these researchers did not suggest a unit or a way to measure effort. Here, we examine the use of a measure termed *Equivalent Effort* (*E*), which is calculated using

$$E = \sum_{i=0}^{n} (U_i \times P_i) \qquad (9)$$

where U_i is the total number of users of all systems at the period of time *i*, and P_i is the percentage of users using the system for which we are measuring its *E* (see Figure 3-11[4][27][34]). Here, we use a month as a unit of time and thus E would be in *user-months*. Using available data [61], *E* can be calculated for some versions of Windows operating systems.

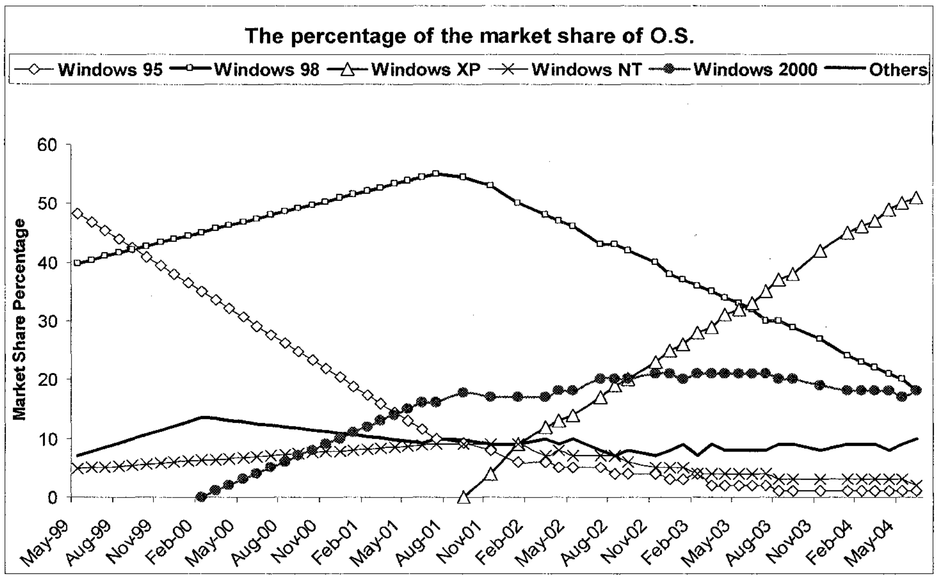

Figure 3-11. The percentages of installed base of some Windows systems

Now let us use another model that uses effort as a factor to model vulnerability discovery. Equivalent effort reflects the effort that would have gone into finding vulnerabilities more accurately than time alone. This is somewhat analogous to using CPU time for software reliability growth models (SRGMs) [44]. If we assume that the vulnerability detection rate with respect to effort is proportional to the fraction of remaining vulnerability, then we get an exponential model, just like the exponential

SRGM. The model can be given as follows:

$$\Omega(E) = B(1 - e^{-\lambda_{vu} E}) \tag{10}$$

where λ_{vu} is a parameter analogous to failure intensity in SRGMs and B is another parameter. B represents the number of vulnerabilities that will eventually be found. We will refer to the model given by Equation (10) as the *Effort-based model.* Figure 3-12 below shows a hypothetical plot of the effort based model.

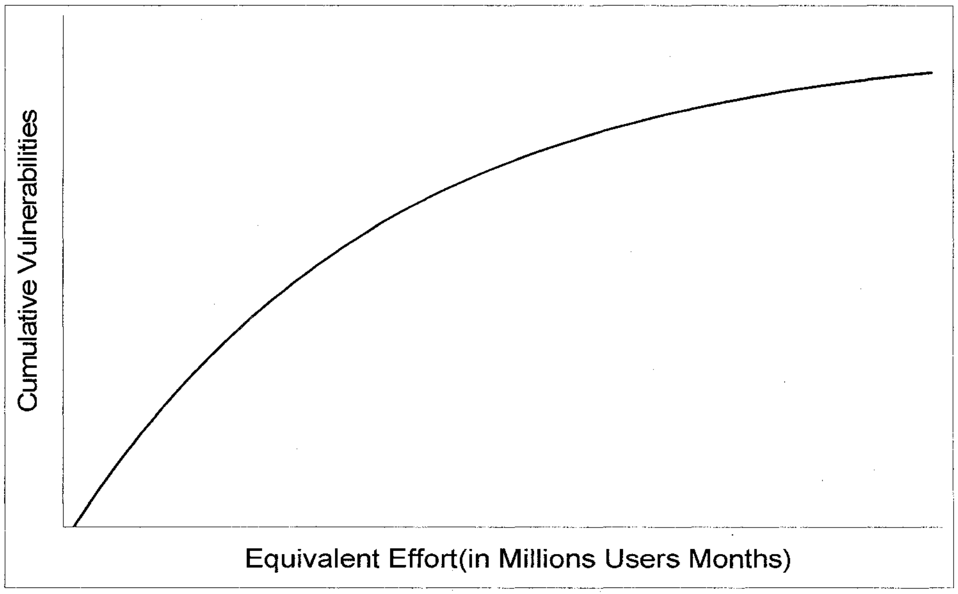

Figure 3-12. Hypothetical plot of the Effort-Based Model

3.4 CONCLUSION

In this chapter we have previewed several vulnerability datasets plotted against time, and we have noticed a common pattern that can be referred to as a three-phase pattern with learning, linear, and saturation phases. The logistic model was proposed as a strong candidate because it considers the three phases; moreover, a linear model was proposed as a simplified model to fit some of the datasets that have shown linear behavior. Also, an equivalent effort model was presented as a model that takes into consideration the usage environment. Next, the models need to be fitted to the datasets and the goodness of fit must be evaluated to validate the models.

CHAPTER 4

VALIDATING VULNERABILITY DISCOVERY MODELS

4.1 INTRODUCTION

Three datasets were previewed in CHAPTER 3, and three vulnerability discovery models were presented. Here, we use the least square fit method to fit the datasets to the models and chi-square (χ^2) is used to evaluate the goodness of fit and determine whether the fit is significant or not.

The following sections will preview the methodology used, then will show the fitting results for the models on different groups of software systems and discuss the outcome of the goodness of fit tests.

4.2 FITTING THE MODELS AND EVALUATING THE GOODNESS OF FIT

To fit the models, a standard statistical fitting technique is used, the least square fit, where the datasets are fitted to the model using regression analysis by minimizing the sum of squared errors:

$$SSE = \sum_{i=1}^{n} (o_i - e_i)^2 , \qquad (11)$$

Therefore, the regression analysis will suggest the optimal values of the model's parameters. After the fit is performed, the following step is performed using chi-square (χ^2) goodness of fit test.

The goodness of fit test for the three presented models, the logistic model, the linear model, and the effort based model. The standard statistical goodness of fit is chi-square

(χ^2). The following section briefly reviews χ^2, and then the three later sections show the goodness of fit results.

We will apply two goodness-of-fit tests. The first is the Chi-square goodness of fit test. The Chi-square (χ^2) statistic is calculated as follows:

$$\chi^2 = \sum_{i=1}^{n} \frac{(o_i - e_i)^2}{e_i}, \tag{12}$$

where o_i is the observed value and e_i is the model's expected value.

For the fit to be acceptable, the Chi-square statistic should be less than the critical value for a given alpha level and degrees of freedom. The P-value is the probability that a value of the χ^2 statistic at least as high as the value calculated by the above formula could have occurred by chance. We use an alpha level of 5%; i.e., if the P-value of the Chi-square test is below 0.05, then the fit will be rejected. A P-value closer to 1 indicates a better fit. P-value is calculated by using the number of degrees of freedom of the dataset and the Chi-square distribution.

4.3 MODELING VULNERABILITIES IN OPERATING SYSTEMS

Operating Systems face escalating security challenges because connectivity is growing and the overall number of incidents is increasing. Moreover, operating systems are responsible for managing and protecting the critical resources and assets of any system. Hence, operating systems are considered a potential target for attackers, and studying data from operating systems can give us a clear picture of how the vulnerabilities are discovered and how to better analyze the discovery process.

The Microsoft's Windows family of operating systems has dominated the OS market for the past decade. Records of each discovered vulnerability is available. Therefore, they are good examples of complex systems to study. Linux, on the other hand, represents open source operating systems, which makes it interesting to compare the vulnerability discovery process. Furthermore, we choose to include the Solaris operating system to look at a broader picture of a diverse group of datasets examined.

In this section we examine the data for some versions of Windows, Linux, and Solaris. The accumulated numbers of vulnerabilities are fitted to the Logistic model presented earlier. We use the Chi-square goodness of fit test for each dataset to test whether the fit is significant or not.

4.3.1 Modeling Operating Systems Vulnerabilities Using The Logistic Model (AML)

In this section we apply the logistic model (AML), which is the time-based model proposed in Chapter 4, to several versions of Windows operating systems. The AML model is given by Equation (4), and the plot of Windows 95 (see Figure 4-1) shows a clear compliance linking the model and the actual data.

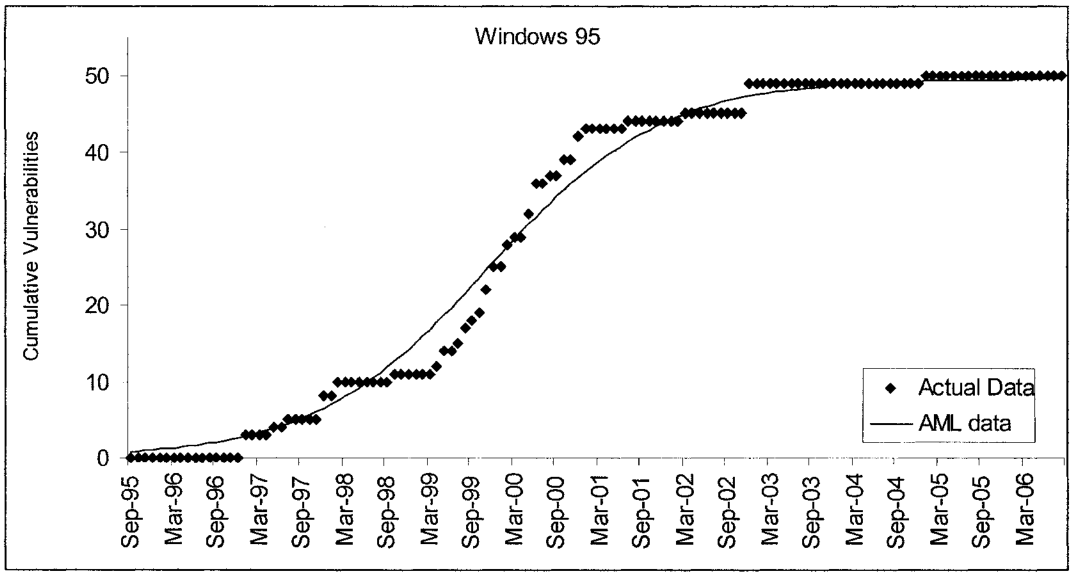

Figure 4-1. Fitting Windows 95 cumulative Vulnerabilities to the AML model

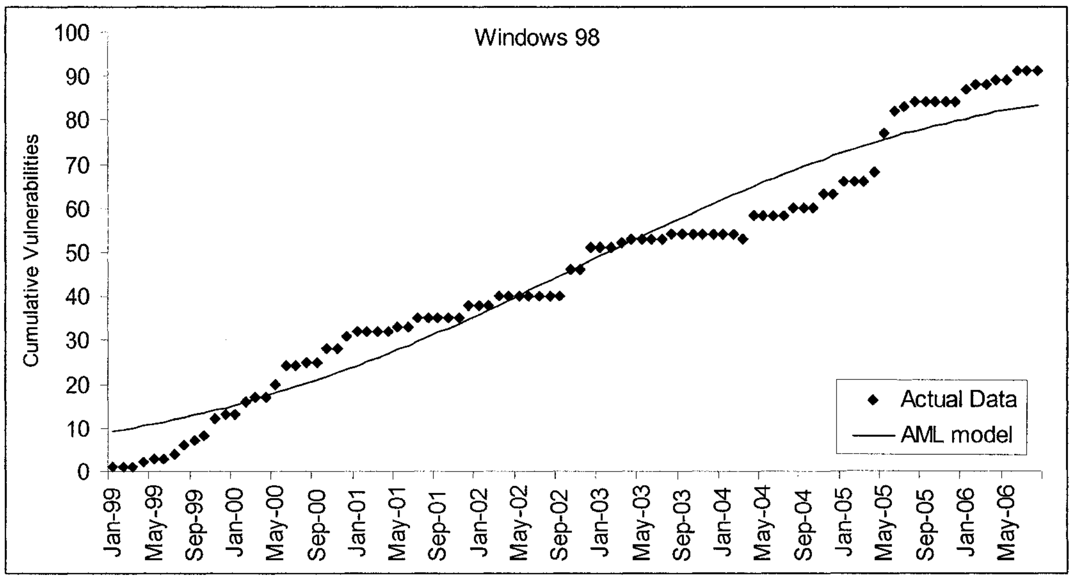

Figure 4-2. Fitting Windows 98 cumulative Vulnerabilities to the AML model

The Windows 98 data shown in Figure 4-2 above fitted the AML model too. However, it is not as clear because there is a jump in the number of vulnerabilities reported due to vulnerabilities discovered in Windows XP and shared with Windows 98. This is because, some of the vulnerabilities reported for Windows 98 were actually found in Windows XP. So, the actual Windows 98 data is a superimposition of more than one of the AML models.

For Windows XP plots in Figure 4-3, there is a significant fit too; however, here we do not observe the complete S-shaped curve. We can only see the first half of the curve because Windows XP's data are still in the linear phase and not matured yet.

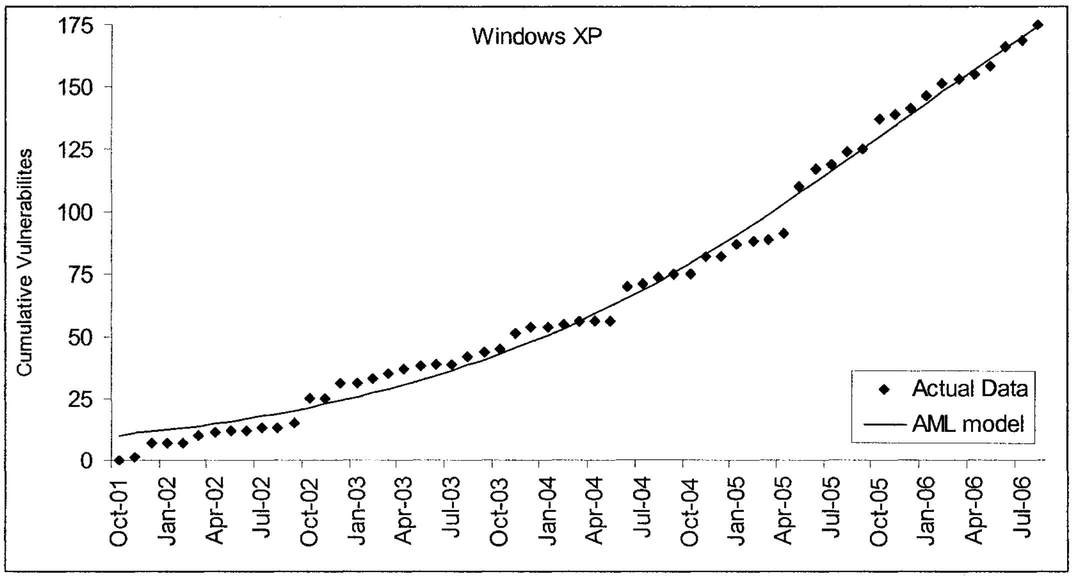

Figure 4-3. Fitting Windows XP cumulative Vulnerabilities to the AML model

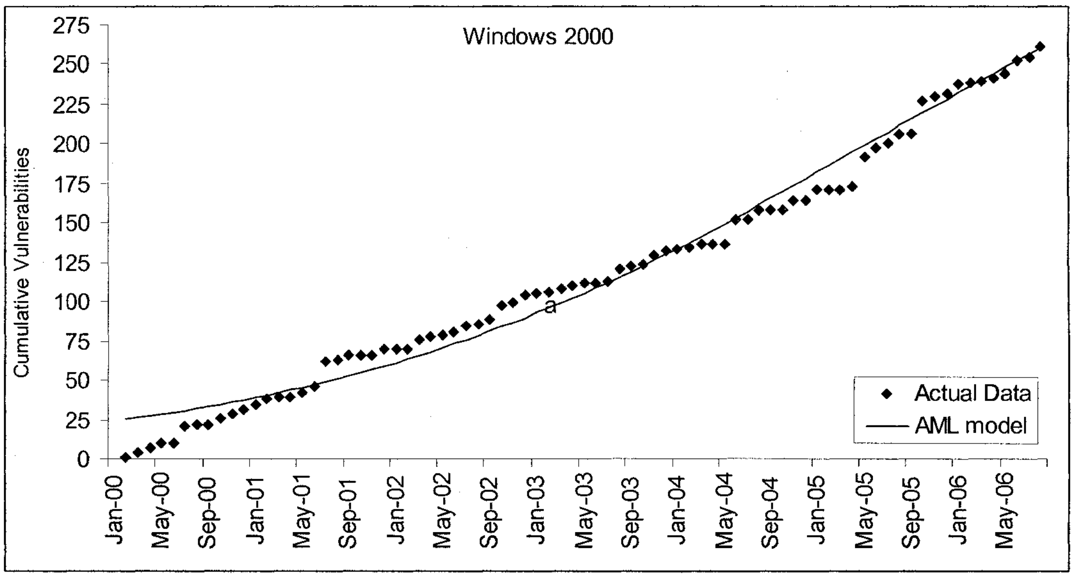

Figure 4-4. Fitting Windows 2000 cumulative Vulnerabilities to the AML model

For Windows 2000 plots in Figure 4-4 there is a significant fit; also here we do not observe the complete S-shaped curve; we can only see the first half of the curve, extended to show the beginning of the saturation phase because Windows 2000's data are just leaving the linear phase. We have also observed that a number of vulnerabilities discovered later are actually shared with Windows XP.

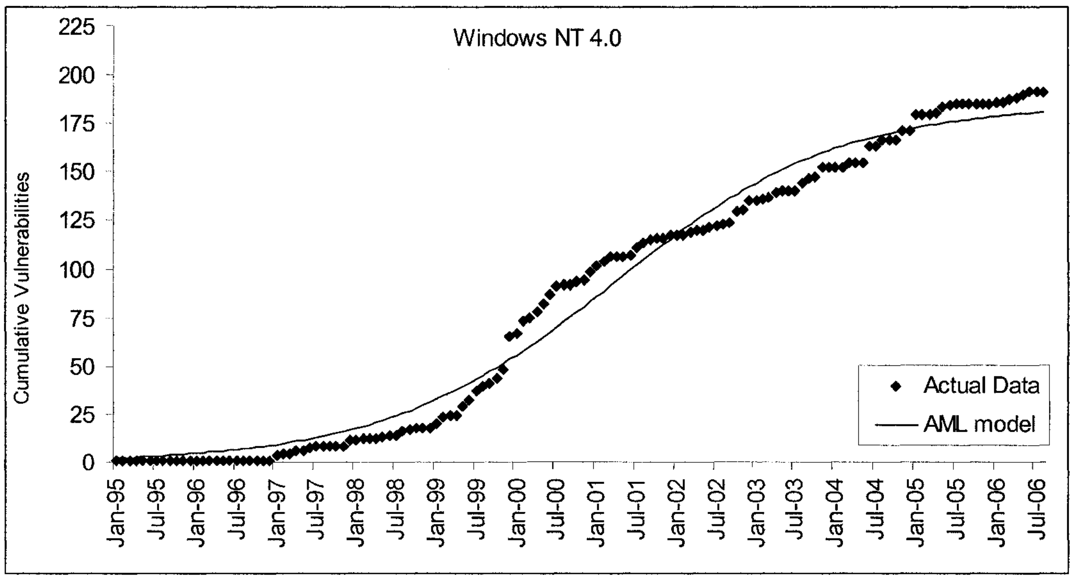

Figure 4-5. Fitting Windows NT 4.0 cumulative Vulnerabilities to the AML model

Figure 4-5 above shows the Windows NT 4.0. Here we can note that there are 2 S-shaped patterns; the first one extends until the beginning of 2003, and the other one starts at that time. We note that the model can not fit if applied directly because of the superimposition of two consecutive models. So, in cases like this, the model applies to the first part, which is the actual number discovered, while the second S-shape is a result of shared vulnerabilities found in Windows 2000.

Figure 4-6 shows that Red Hat Linux 6.2 conforms to the model. The fit looks very acceptable especially during the saturation and learning phases. Moreover, Figure 4-7 shows the vulnerabilities in Red Hat Linux 7.1. Here, too the fit is quite perfect. Despite the short learning phase.

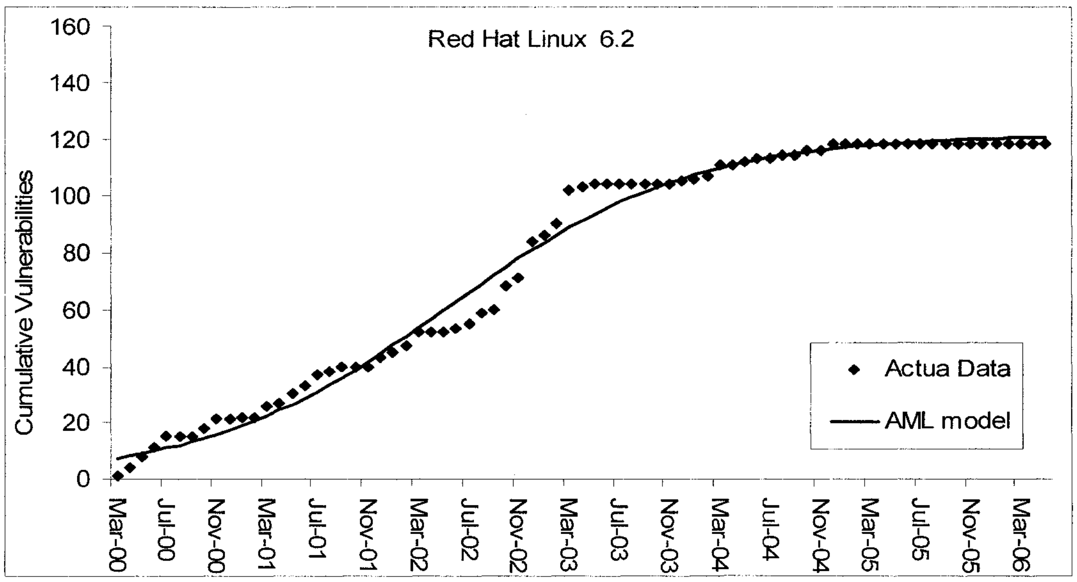

Figure 4-6. Fitting Red Hat Linux 6.2 cumulative Vulnerabilities to the AML model

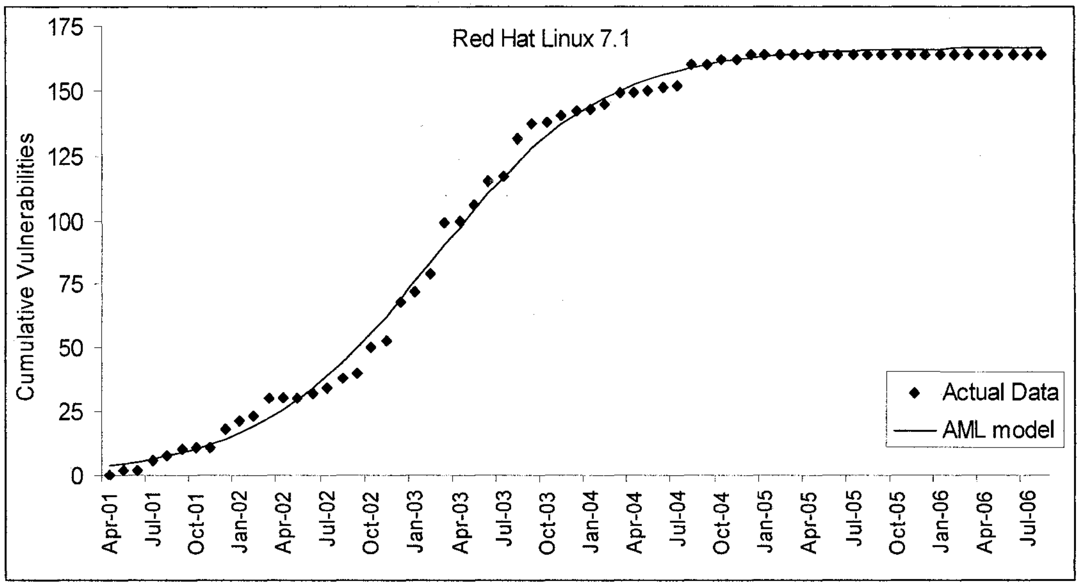

Figure 4-7. Fitting Red Hat Linux 7.1 cumulative Vulnerabilities to the AML model

Figure 4-8 shows the Red Hat Fedora. The Red Hat Fedora has 5 releases, Core 1 through Core 5, and soon there will be Core 6, and this data includes all versions. Therefore, the model could also show a super imposition of more than one S-shaped curve and that is what might have caused the re-acceleration after some saturation in the year 2006 as we can notice in the actual accumulation of vulnerabilities.

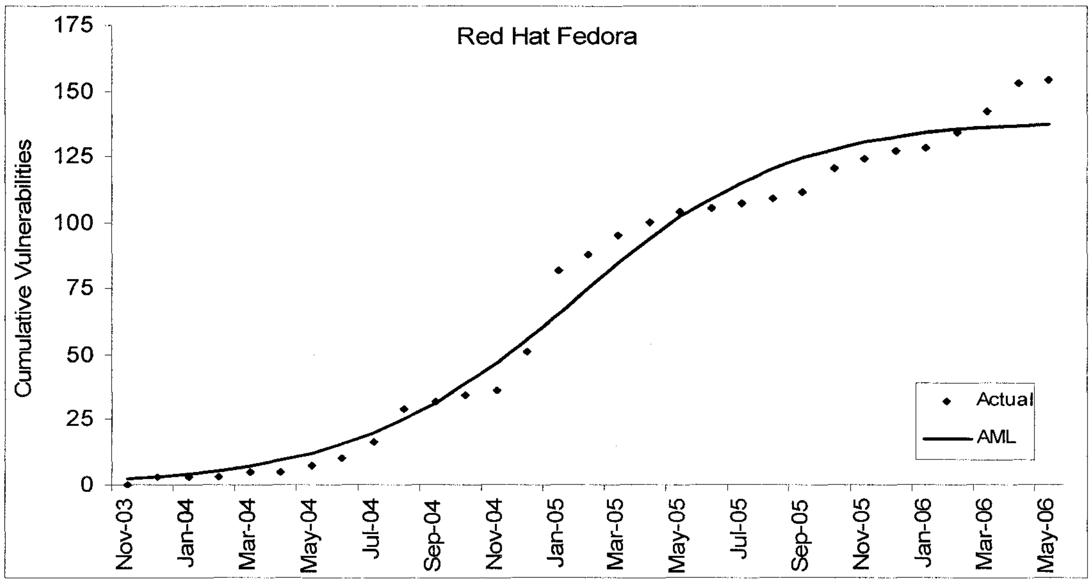

Figure 4-8. Fitting Red Hat Fedora cumulative Vulnerabilities to the AML model

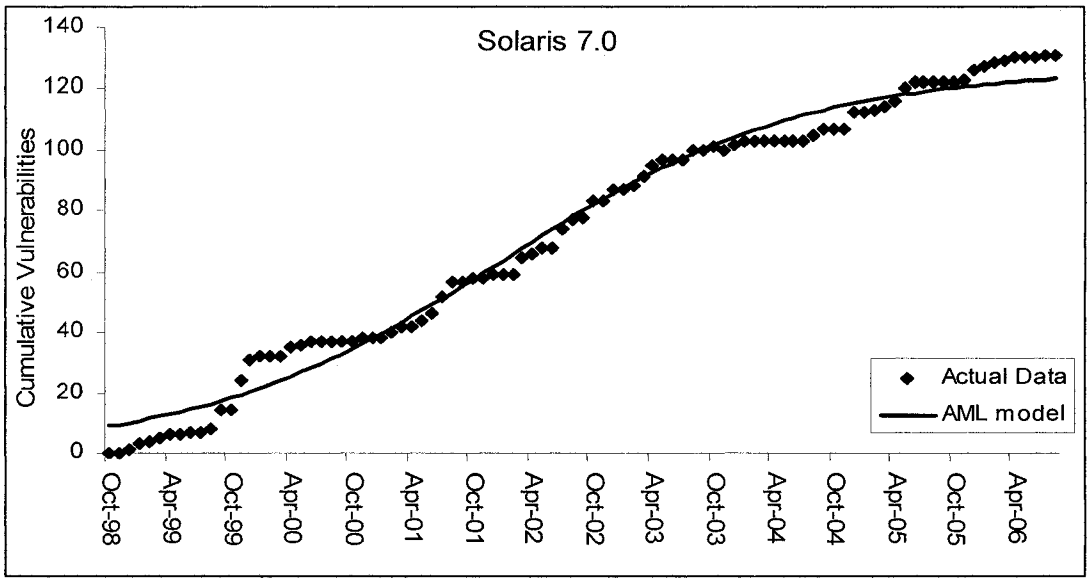

Figure 4-9. Fitting Solaris 7.0 cumulative Vulnerabilities to the AML model

Figure 4-9 shows that Solaris 7.0 fits the model with a P-value of only 0.1759. Here too, there is a significant number of vulnerabilities shared with later Solaris versions 8 and 9; however, the model has a semi-linear trend.

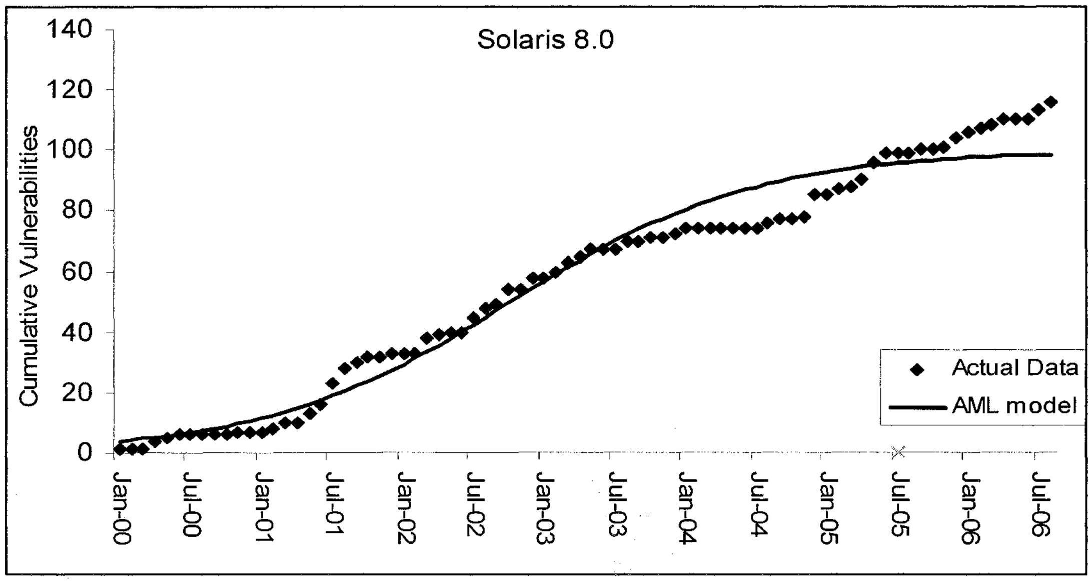

Figure 4-10. Fitting Solaris 8.0 cumulative Vulnerabilities to the AML model

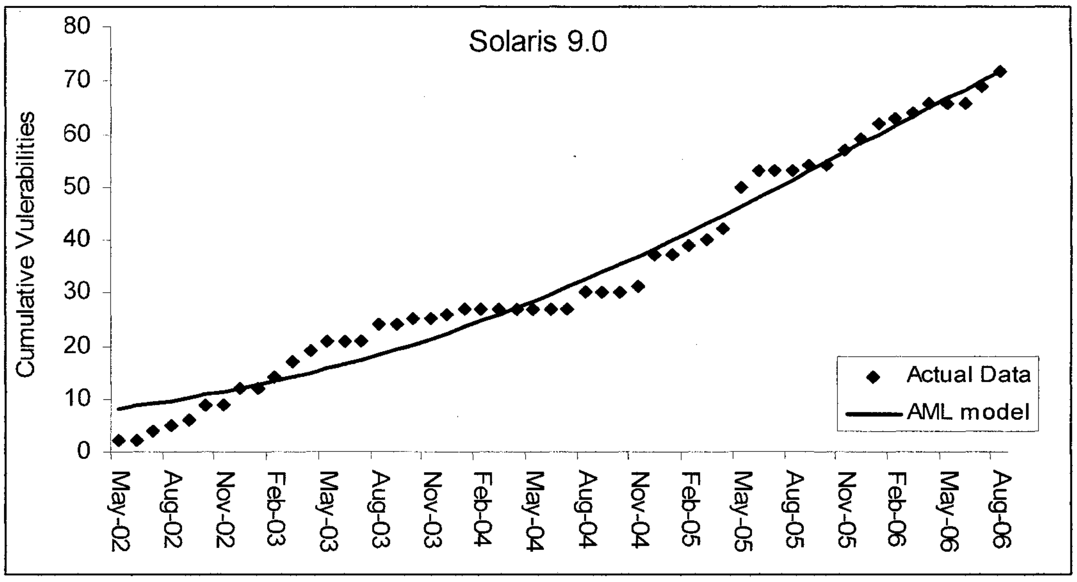

Figure 4-11. Fitting Solaris 9.0 cumulative Vulnerabilities to the AML model

Other versions of Solaris are version 8.0 shown in Figure 4-10 and version 9.0 shown in Figure 4-11. Both versions fit the s-shaped model with a complete s-shape in the case of Solaris 8.0 and with an incomplete S-shape in the case of Solaris 9.0. By looking at the goodness of fit table (see Table 4-1) we see that all datasets fit the AML model except for Windows NT 4.0, and that is because of the nature of Windows NT 4.0, which seems to be a super imposition of more than one AML model.

Table 4-1 shows the parameter values with parameter A typically less than 0.002 except in rare cases, while B is always close to the total number of vulnerabilities; on the other side parameter C ranges from 0.038595 to 1.32092.

Table 4-1: Goodness of fit test results for the logistic model (AML), showing parameter values

System	A	B	C	DF	χ^2	χ^2_{critial} (5%)	P-value	Fit Result
Win 95	0.001652	49.5512	1.32092	132	45.748	159.8135	1	Significant
Win 98	0.000522	92.6789	0.10233	92	107.001	115.3898	0.135785	Significant
Win XP	0.000232	280.513	0.09994	59	54.6261	77.93052	0.637299	Significant
Win NT 4.0	0.000317	184.78	0.44439	140	267.173	168.613	0	Insignificant
Win 2000	0.0001096	391.984	0.0386	79	95.1500	100.7486	0.104079	Significant
Red Hat Linux 6.2	0.000855	121.235	0.13973	75	36.19688	96.21667	0.999956	Significant
Red Hat Linux 7.1	0.169437	166.735	0.29531	65	39.6227	84.82064	0.99456	Significant
Red Hat Fedora	0.002014	139.045	0.53497	31	31.81382	44.98534	0.425800	Significant
Solaris 7.0	0.000526	126.32	0.11076	95	107.701	118.7516	0.1759	Significant
Solaris 8.0	0.000961	99.815	0.26380	80	69.7045	101.8795	0.7877	Significant
Solaris 9.0	0.000528	114.25045	0.11979	52	34.964	69.83216	0.9665	Significant

4.3.2 Validating The Linear Model (LM) On Operating Systems

Windows 95 dataset is strongly s-shaped; therefore, the linear model could not fit the dataset (see Figure 4-12). However, in Windows 98, the linear model fits the data perfectly (see Figure 4-13). Table 4-2 below clearly shows the fitting result with a P-value of 0.000016 for Windows 95 and a significant fit for Windows 98 with P-value of 0.999962.

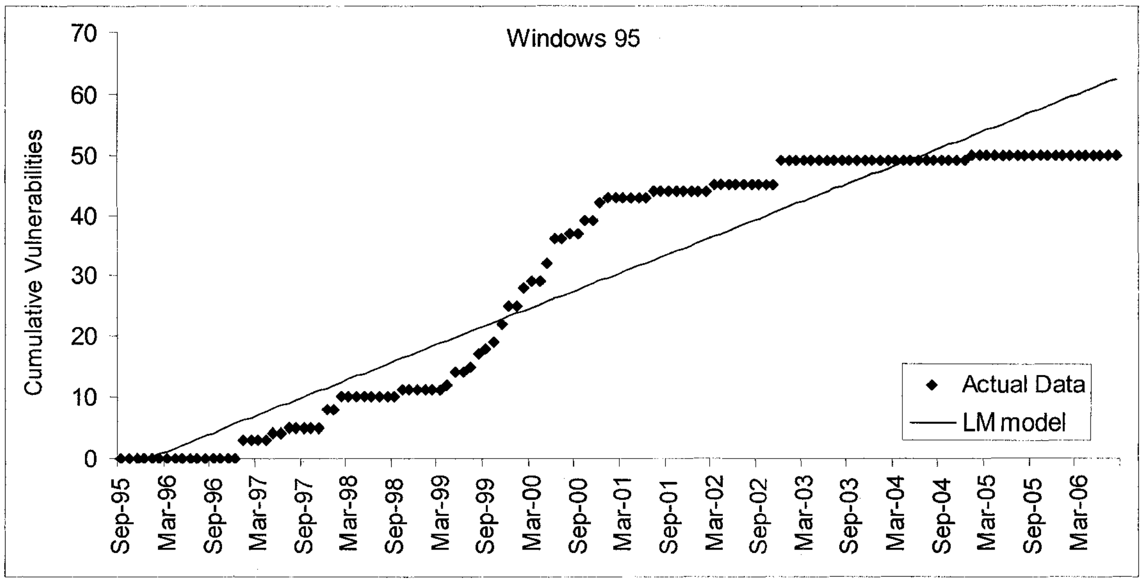

Figure 4-12. Fitting Windows 95 cumulative Vulnerabilities to the LM model

In Figure 4-14, Windows XP fails to fit the linear model (LM) with a slightly lower than the critical alpha value of 0.05. In Windows XP, the P-value is only 0.026955 (see Table 4-2).

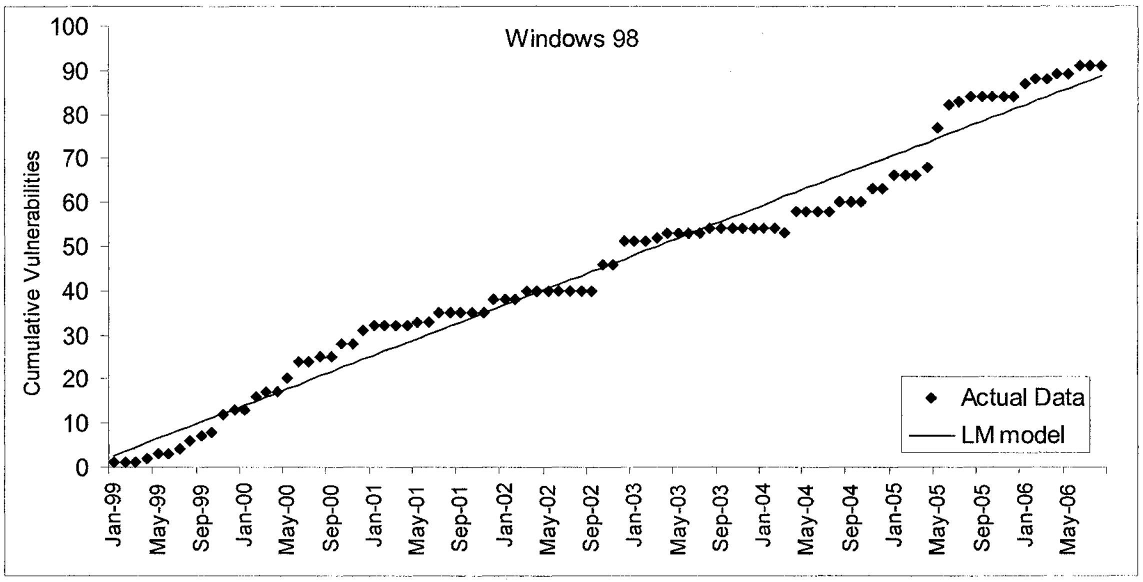

Figure 4-13. Fitting Windows 98 cumulative Vulnerabilities to the LM model

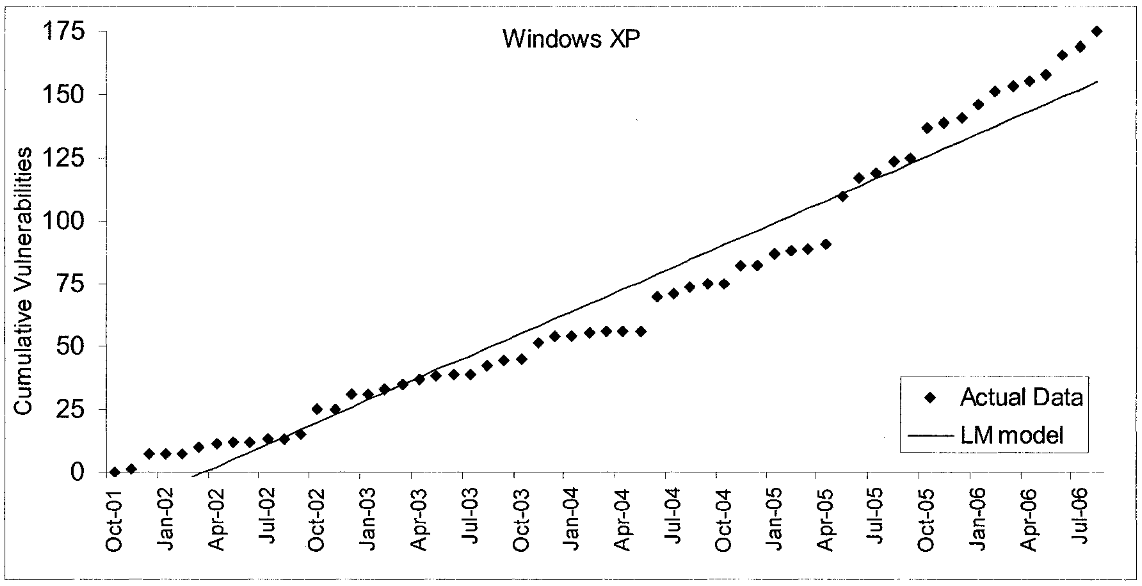

Figure 4-14. Fitting Windows XP cumulative Vulnerabilities to the LM model

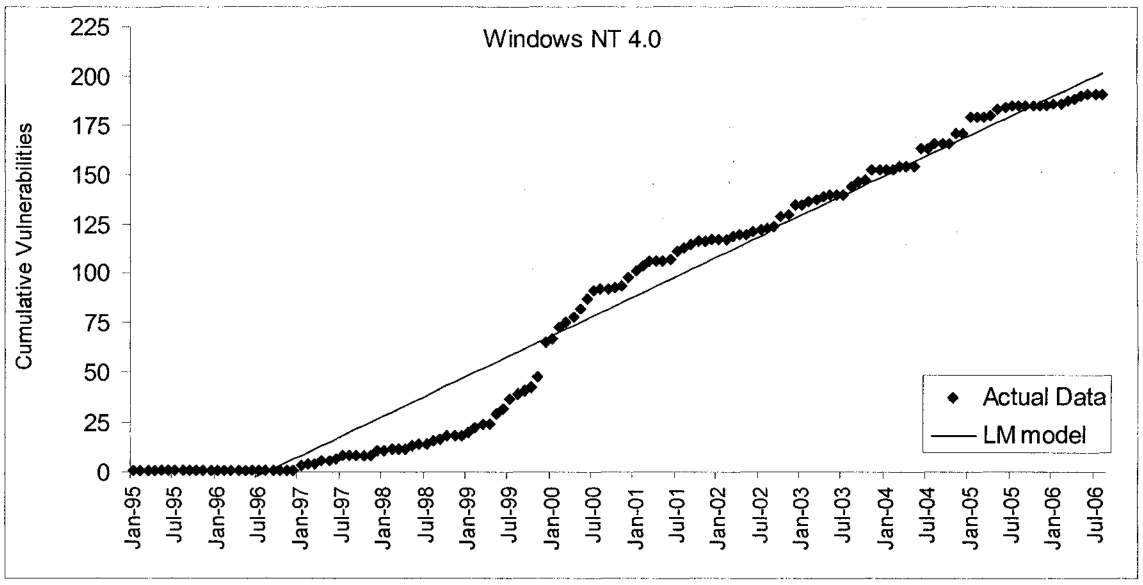

Figure 4-15. Fitting Windows NT 4.0 cumulative Vulnerabilities to the LM model

In Figure 4-15, we can see that Windows NT 4.0 does not fit the linear model with a P-value of 0, shown in Table 4-2. However, the Windows 2000 fitting shown in Figure 4-16 shows an acceptable fit, with a P-value of 0.815443, indicating a significant fit (see Table 4-2).

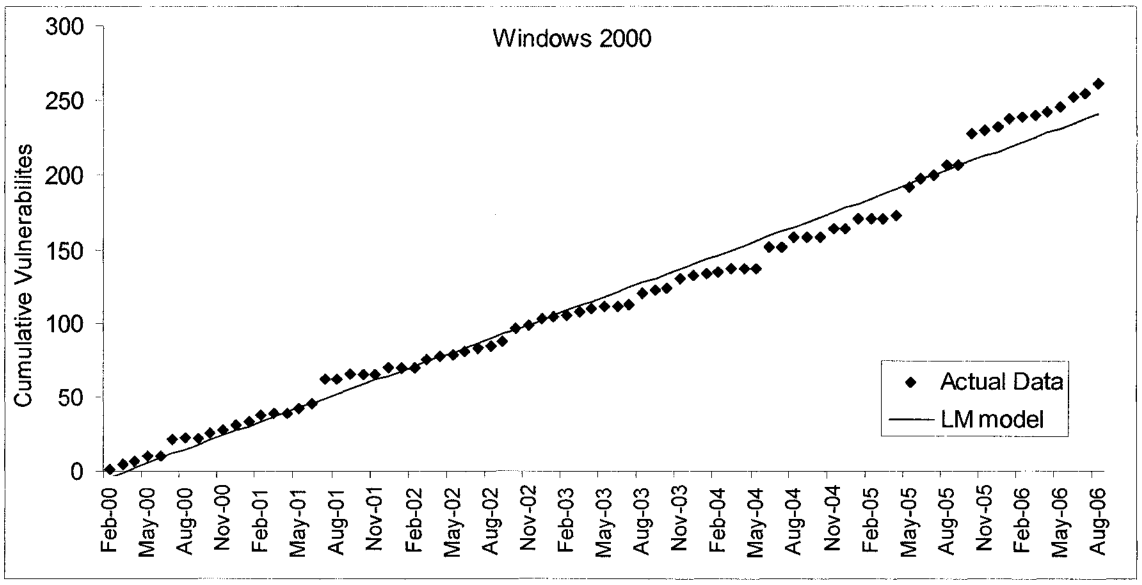

Figure 4-16. Fitting Windows 2000 cumulative Vulnerabilities to the LM model

Red Hat Linux 6.2 strongly opposes the linear model as illustrated in Figure 4-17, while the Red Hat Linux 7.1 shows a linear behavior that enabled the dataset to fit the linear model (see Figure 4-18), where P-values are 0.000002 and 1 respectively (see Table 4-2).

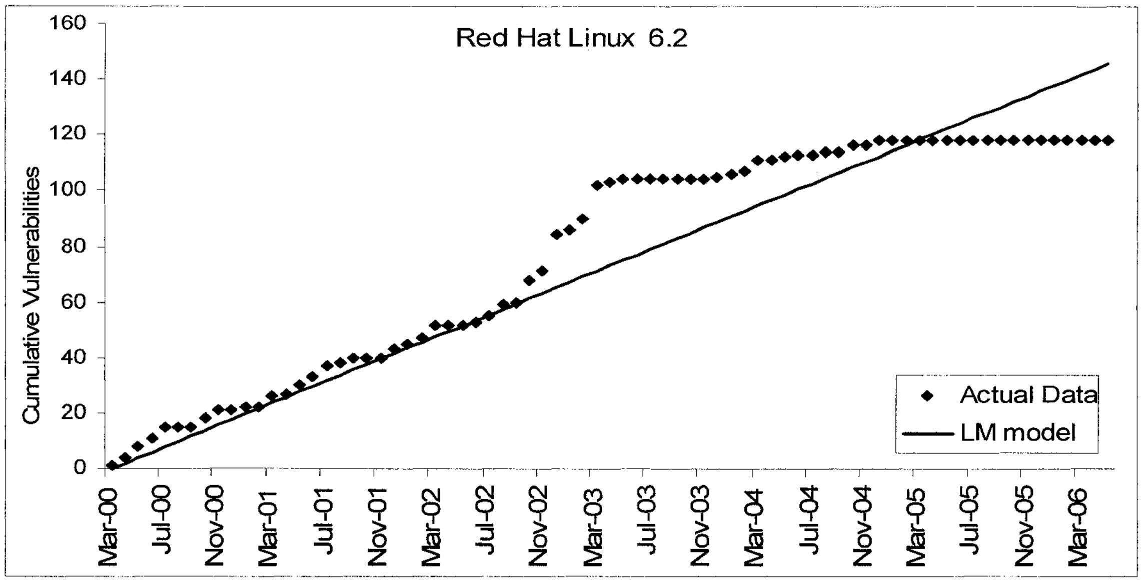

Figure 4-17. Fitting Windows Red Hat Linux 6.2 to the linear model (LM)

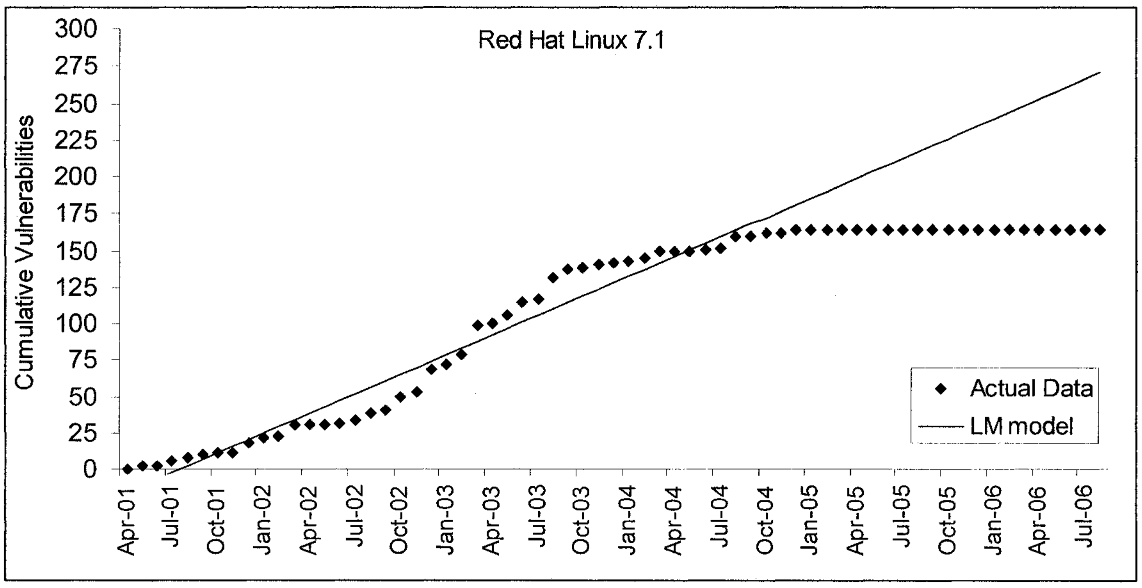

Figure 4-18. Fitting Windows Red Hat Linux 7.1 to the linear model (LM)

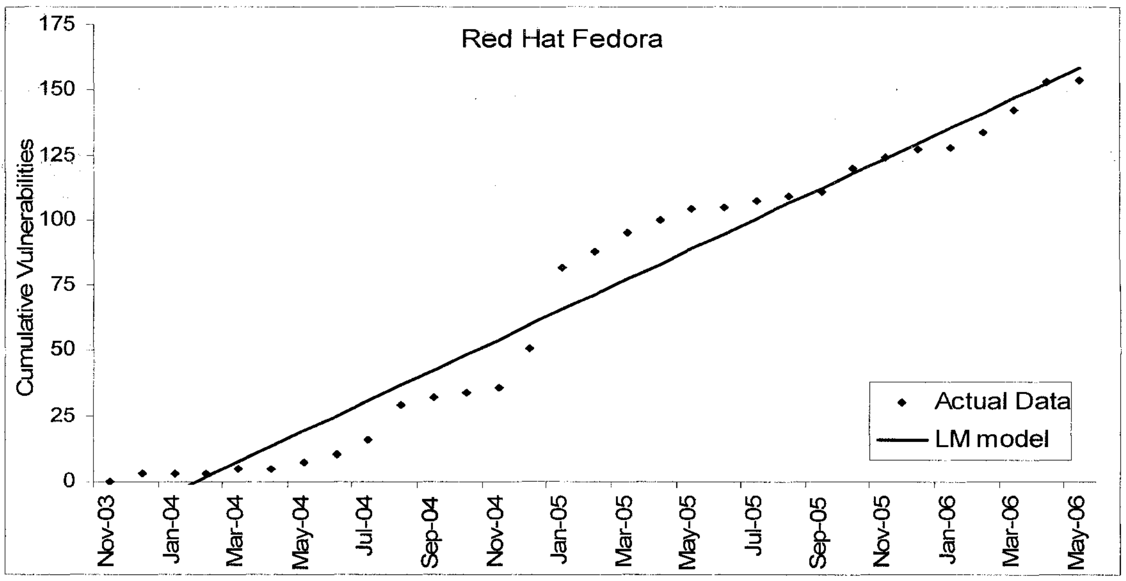

Figure 4-19. Fitting Windows Red Hat Fedora to the linear model (LM)

Red Hat Fedora was unable to fit the linear model as shown in Figure 4-19. It has a low P-value of 0.000022(see Table 4-2).

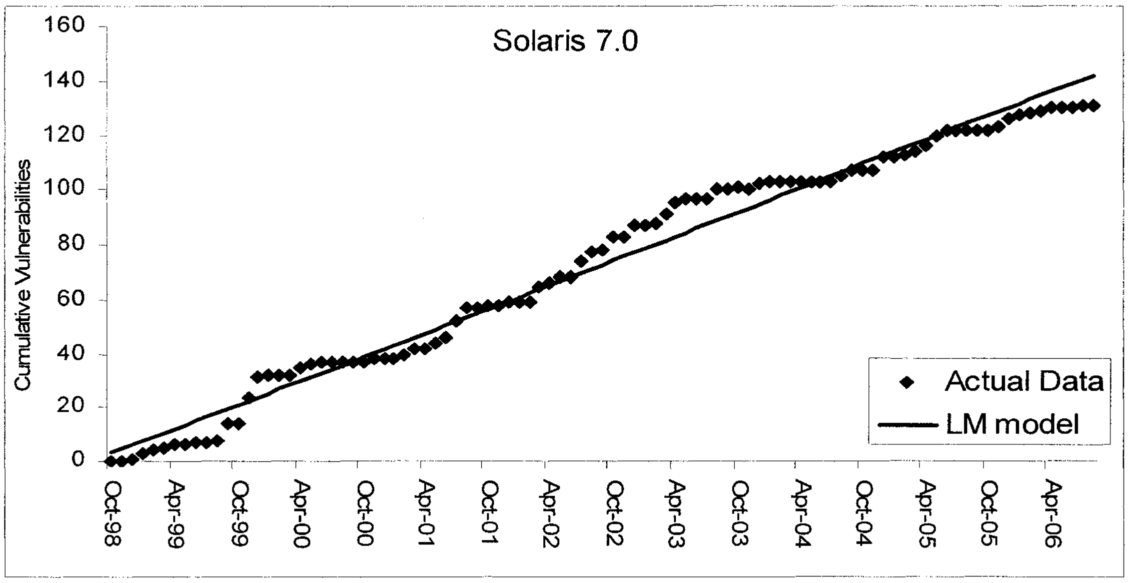

Figure 4-20. Fitting Solaris 7.0 cumulative Vulnerabilities to the LM model

Solaris 7.0, 8.0, and 9.0 have fitted the linear model (see Figure 4-20, Figure 4-21 and Figure 4-22). The plots indicate that there is a strong linear property of the trends with P-values of 0.79982, 0.95088, and 0.99882 respectively.

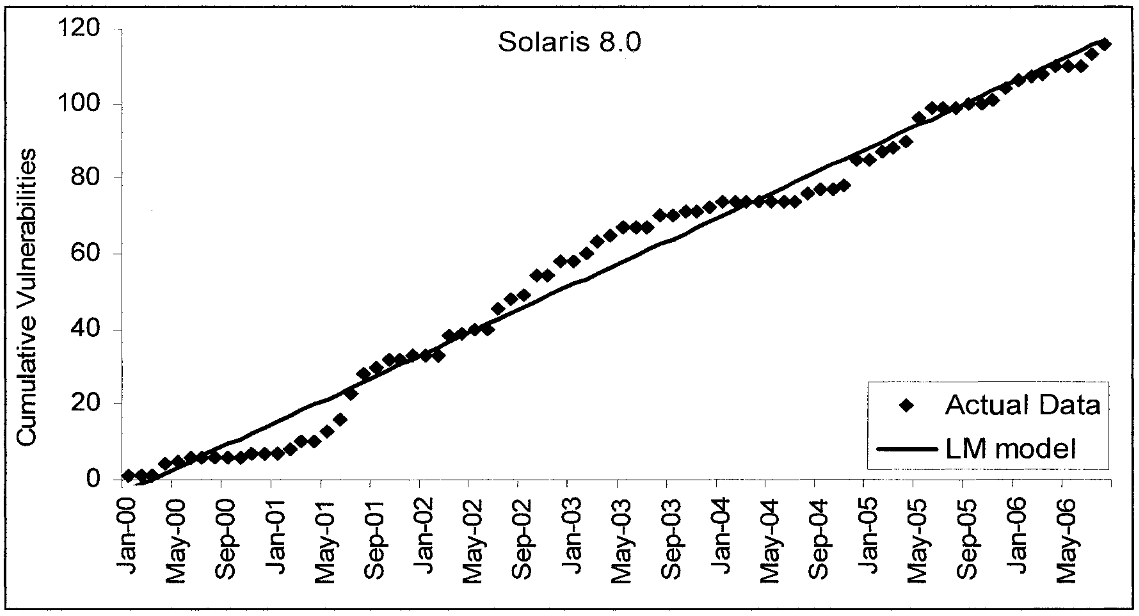

Figure 4-21. Fitting Solaris 8.0 cumulative Vulnerabilities to the LM model

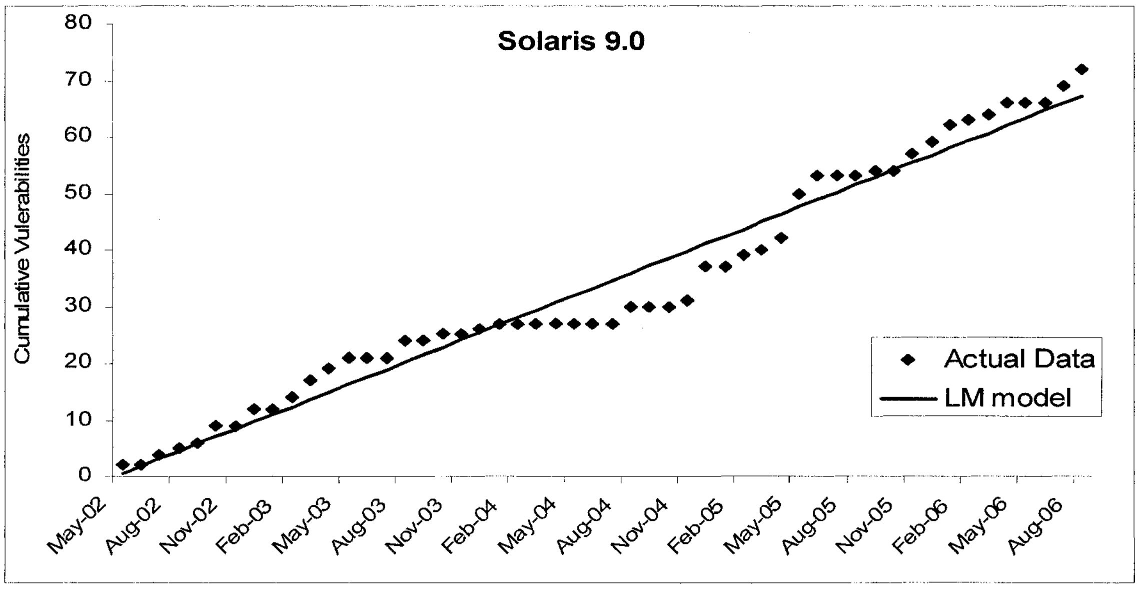

Figure 4-22. Fitting Solaris 9.0 cumulative Vulnerabilities to the LM model

Table 4-2: Goodness of fit test results for linear fit (LM), showing parameter values

System	Slope (s)	k	DF	χ^2	χ^2_{critial} (5%)	P-value	Fit Result
Win 95	0.49043	-2.2199	132	221.905	159.8135	0.000016	Insignificant
Win 98	0.94566	1.80963	92	47.7546	115.3898	0.999962	Significant
Win XP	2.94126	-18.3565	59	81.6803	77.93052	0.026955	Insignificant
Win NT 4.0	1.67923	-33.6284	140	416.205	168.613	0	Insignificant
Win 2000	3.13807	-7.11782	79	67.6315	100.7486	0.815443	Significant
Red Hat Linux 6.2	1.78128	9.591	75	145.462	96.21667	0.000002	Insignificant
Red Hat Linux 7.1	4.4672	-19.6343	65	16.4555	84.82065	1	Significant
Red Hat Fedora	5.8121	-21.477	27	67.979	40.11327	0.000022	Insignificant
Solaris 7.0	1.4777	1.72296	95	83.256	118.7516	0.79982	Significant
Solaris 8.0	1.5172	-4.330	80	60.301	101.8795	0.95088	Significant
Solaris 9.0	1.3069	-0.710	52	46.965	69.83216	0.99882	Significant

4.3.3 Discussion

We statistically examined the linear model on the same sets of operating systems and found that the fit was significant in six datasets and insignificant in the other five datasets. Table 4-2 details the Chi-square test results.

The slope results show that the value of the parameter Slope (s) varies, which indicates a difference in vulnerabilities discovery rates in some systems. For example, newer Windows systems like Windows XP and Windows 2000 show a slope of 2.94126 and 3.13807 vulnerabilities per month, while the older systems Windows 95, Windows 98, and Windows NT 4.0 have shown a slower rate of 0.49043, 0.94566, and 1.67923 vulnerabilities per month respectively.

We did not constrain the slope except when k was constrained to be positive, which was a reasonable assumption, since the number of vulnerabilities cannot be negative at any time. The question was when to use linear and when to use S-shaped. Below, Figures 7.1 and 7.2 show a plot of two datasets, Red Hat Linux 7.1 and Windows XP.

Red hat Linux 6.2 Fedora Systems has a slope of 1.78128, while Red Hat Linux 7.1 and Red Hat Fedora have shown steeper slopes of 4.4672 and 5.8121 respectively. Finally, the slopes of Solaris were very close to each other, ranging from 1.3069 to 1.5172 (see Table 4-2).

The linear model is simple and can provide valuable information about the vulnerability discovery rate directly. However, it only fit a limited number of datasets. Here too, we believe that some factors change the vulnerability discovery rates which make the datasets basically a number of consecutive linear models, these factors include shared vulnerabilities, changes in market share or releasing newer version of the software system.

4.4 MODELING VULNERABILITIES IN WEB SERVERS

HTTP Web servers carry the vital task of communicating with clients, a task which puts them under increasing security challenges. Therefore, it is very rewarding to discover vulnerabilities in this category of software systems; the reward factor is closely associated with the popularity and spread of a web server. Therefore, the market share is one of the most significant factors impacting the effort expended in exploring potential vulnerabilities. Higher market share offers a greater incentive to explore and exploit vulnerabilities because attackers will obviously find it more profitable or satisfying to spend their time focusing on a software system having a greater market share.

Table 4-3. Market share, Number of vulnerabilities and release dates of some web servers

	Apache		**IIS**		**SJSWS (SunOne)**	**Zeus**
First Release	1995		1995		2002	1995
Market Share	69.7%		20.92%		2.53%	0.78%
Version	**1.x**	**2.x**	**4.0**	**5.0**	Up to 6.1	Up to 4.3
Vulnerabilities	60	47	86	74	3	5

Table 4-3 above present's data obtained from NVD [61] and Netcraft [59], showing the current web server market share and total number of vulnerabilities found to date. As we can see from the table, for servers with a lower percentage of the market, such as Sun Java System Web Server (SJSWS) and Zeus, the total number of vulnerabilities found is low. This does not necessarily mean that these systems are vulnerability-free, but merely that only a limited effort has gone into detecting their vulnerabilities. A significant number of vulnerabilities have been found in both Apache and IIS, illustrating the impact of the market share on the motivation for exploring and finding undiscovered vulnerabilities.

The Apache HTTP server was first released in mid 1995. Since then it has gained wide popularity and is used by over 50 million web server systems. Apache dominates the market, probably because it is an open source system that is free. Apache may also have benefited from not having been exposed to serious security issues such as the Code

Red [52], which exploited a vulnerability in IIS (described in Microsoft Security Bulletin MS01-033, June 18, 2001). It appeared on July 13, 2001 and soon spread world-wide in unpatched systems and is an example of a very critical vulnerability. Another example is Nimda worms that were faced by IIS in 2001.

To assure an acceptable degree of security for a web server, the developers need to determine how much testing for security vulnerabilities is needed. Moreover, developers need to be able to project the post-release vulnerability discovery rate to plan the maintenance and patch development effort needed. The available data can be fitted to the vulnerability discovery models (VDMs) to project the trend that the vulnerability discovery process is likely to follow.

There has been considerable discussion of the security of web servers in recent years. However, investigations have focused on qualitative issues related to detection and prevention of individual vulnerabilities. Quantitative data is sometimes cited, but without any significant critical analysis. Methods need to be developed to allow security related risks to be evaluated quantitatively in a systematic manner. A study by Ford et al. [30] has compared several servers, the number of vulnerabilities, and the associated severity levels. This study identifies a need to develop tools for estimating the risks posed by vulnerabilities.

The two major software components of the Internet are an HTTP (Hyper Text Transfer Protocol) server, also termed a web server, and the browser, which serves as the client. Both of these were first introduced in 1991 by Tim Berners-Lee of CERN and they have now become indispensable parts of both organizational and personal interactions. The early web servers provided information using static HTML pages. The web server now provides dynamic and interactive services between the server and client using database queries, executable script, etc. The web server is able to support functions such as serving streaming media and mail. An HTTP server has thus emerged as a focal point for the Internet.

In this research, we examine the vulnerabilities in the two most widely-used HTTP servers, the Apache server, and the Microsoft IIS (Internet Information Service), both introduced in 1995. While Apache has a much larger overall market share, roughly 70%, IIS may have a higher share of the corporate websites. The market share for other servers is small and thus they are not examined here. IIS is the only HTTP server that is not open-source. Both Apache and IIS are generally comparable in features; however, IIS runs only under the Windows operating systems, whereas Apache supports all the major operating systems.

4.4.1 MODELING WEB SERVERS VULNERABILITIES USING THE LOGISTIC MODEL (AML)

In this section, we fit the vulnerability data for Apache to the Alhazmi-Malaiya Logistic model (AML) [4][5][10] Vulnerability data obtained from NVD are used to test the models [67].

Figure 4-23 shows Apache 1.x (which includes subversions 1.0 through 1.3) vulnerabilities fitted to the AML, clearly showing that the AML follows the data very closely. Moreover, in Figure 4-24 the Apache 2.x data set is shown to fit both models equally well.

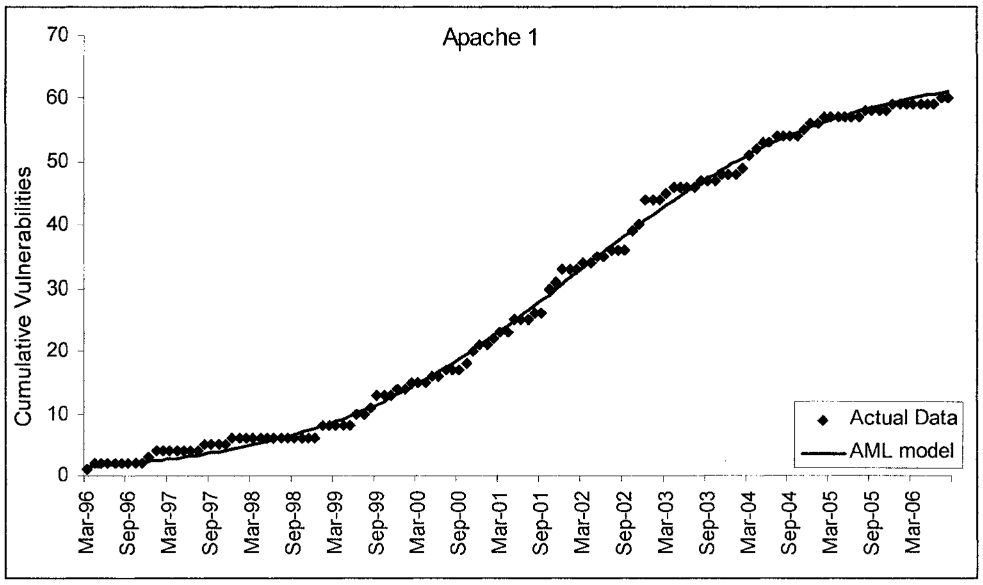

Figure 4-23. Fitting Apache 1 cumulative Vulnerabilities to the AML model

Apache 2 (including sub versions through 2.1) appears to be showing the beginnings of saturation with after a quick learning phase and a mainly linear trend.

Despite having been on the market for several years, Apache 2 and even Apache 1 have not yet reached a clear saturation phase, possibly because of their larger market share. Besides, the number of systems using the Apache web server is still increasing indicating that vulnerability discovery for Apache can be expected to continue at a significant pace in the near future.

On the other hand, the IIS 4.0 and 5.0 web servers appear to have reached a much clearer saturation phase. During the past several months, the vulnerability discovery rate for IIS has dropped to a very low point (see Figure 4-25 and Figure 4-26). A possible explanation for this can be that the number of IIS web servers installed appears to be stationary, unlike the Apache server which is still gaining in terms of new installations. Another possibility is that the number of remaining undiscovered vulnerabilities may actually have dropped significantly.

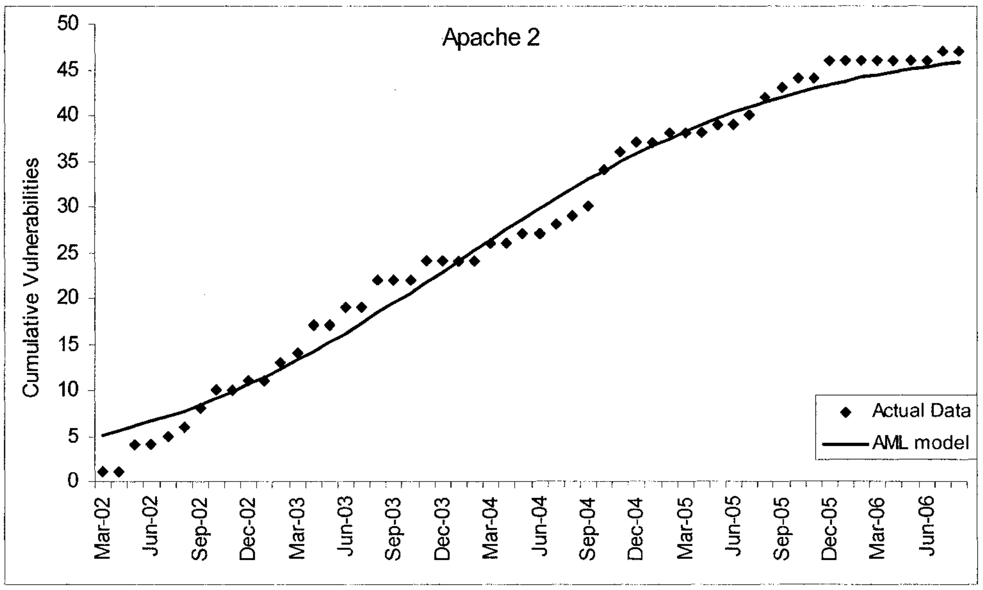

Figure 4-24. Fitting Apache 2 cumulative Vulnerabilities to the AML model

Table 4-4 shows the data fitted to the model using the least square fit. For Chi-squared goodness of fit test, we chose an alpha level of 5%; χ^2 test results in the table show that all data sets have fitted the AML model with P-value > 0.998. The values of the parameter A range from 0.0008 to 0.002, and for C they range from 0.18 to 0.70. The parameter B corresponds approximately to the number of vulnerabilities.

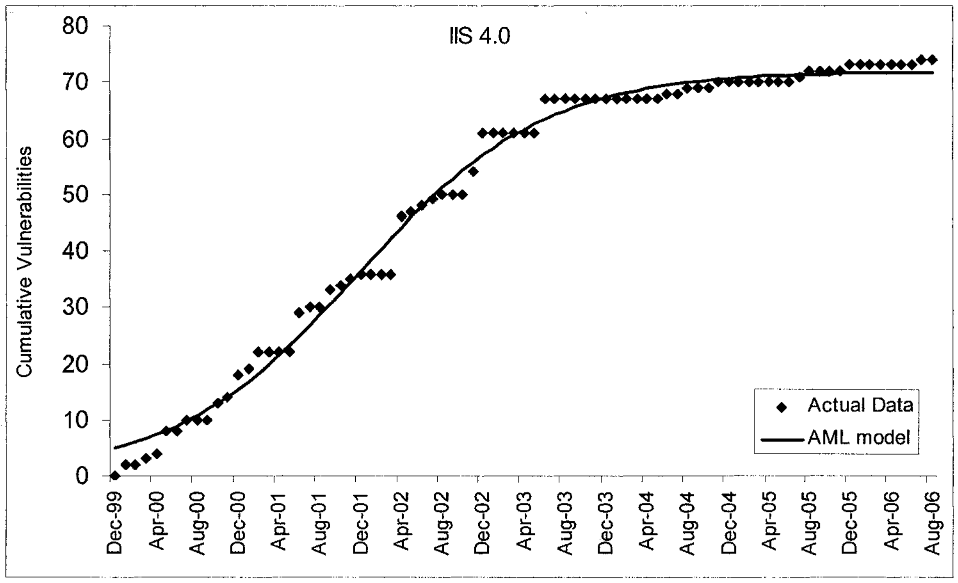

Figure 4-25 Fitting IIS 4.0 cumulative vulnerabilities to the logistic model (AML)

The AML model fitted with P-value > 0.991. The values of the parameter *A* range from 0.000848 to 0.001860, and for *C* they range from 0.18498 to 0.72709. The parameter *B* corresponds approximately to the number of vulnerabilities.

The goodness of fit for the LM model for the later versions Apache 2 and IIS 5 is significant, implying that the vulnerability discovery has not yet reached the saturation phase; whereas the data for the older versions Apache 1 and IIS 4 did not fit the LM model. By observing the *slope* parameter we can see that the rate of vulnerability discovery given by the LM model is higher for Apache 2 than Apache 1 and the IIS 5 rate is higher than IIS 4.

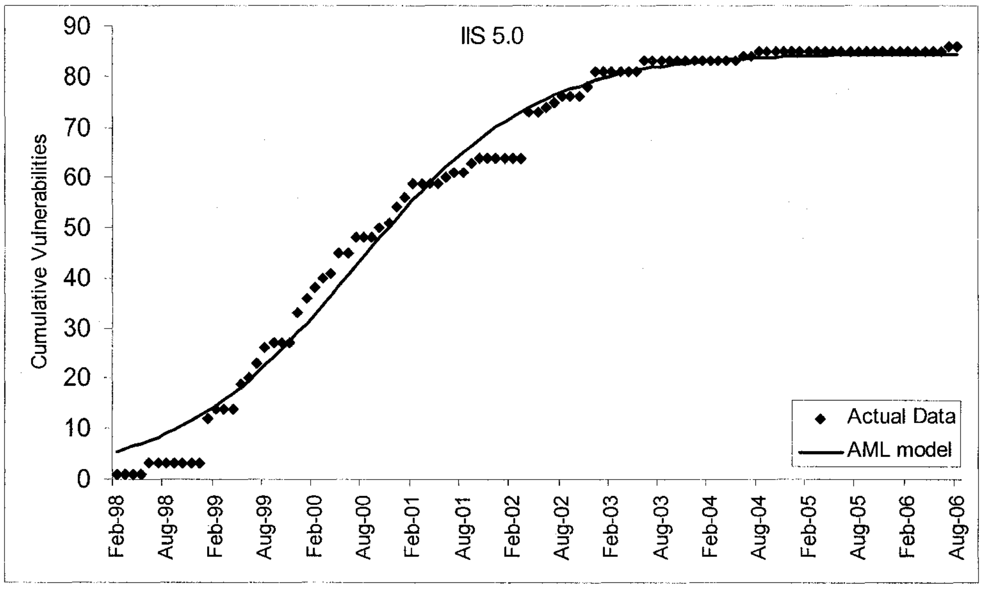

Figure 4-26. Fitting IIS 5.0 cumulative vulnerabilities to the logistic model (AML)

Table 4-4. Goodness of Fit Results for the examined web server's datasets, fitted to the logistic model (AML)

Software System	A	B	C	DF	χ^2	χ^2_{critical}	P-value	Significance
Apache 1.0	0.000848	63.556	0.72709	126	12.896	153.198	1	Significant
Apache 2.0	0.001860	49.886	0.18498	54	15.469	72.153	1	Significant
IIS 4.0	0.001094	84.573	0.18890	103	59.578	127.689	0.998	Significant
IIS 5.0	0.001525	72.002	0.21138	81	22.24	103.01	1	Significant

4.4.2 MODELING WEB SERVER VULNERABILITIES USING THE LINEAR MODEL (LM)

In this section, we fit the vulnerability data for Apache to the linear model (LM) where vulnerability data obtained from NVD are used to test the models.

Figure 4-27 shows Apache 1.x (which includes subversions 1.0 through 1.3) vulnerabilities fitted to the linear model (LM); also, Apache 2.x (see Figure 4-28) has fitted the linear model even better than Apache 1.x. Apache 2 (including subversions through 2.1) appears to be in the linear phase, since the number of vulnerabilities still appears to be growing linearly.

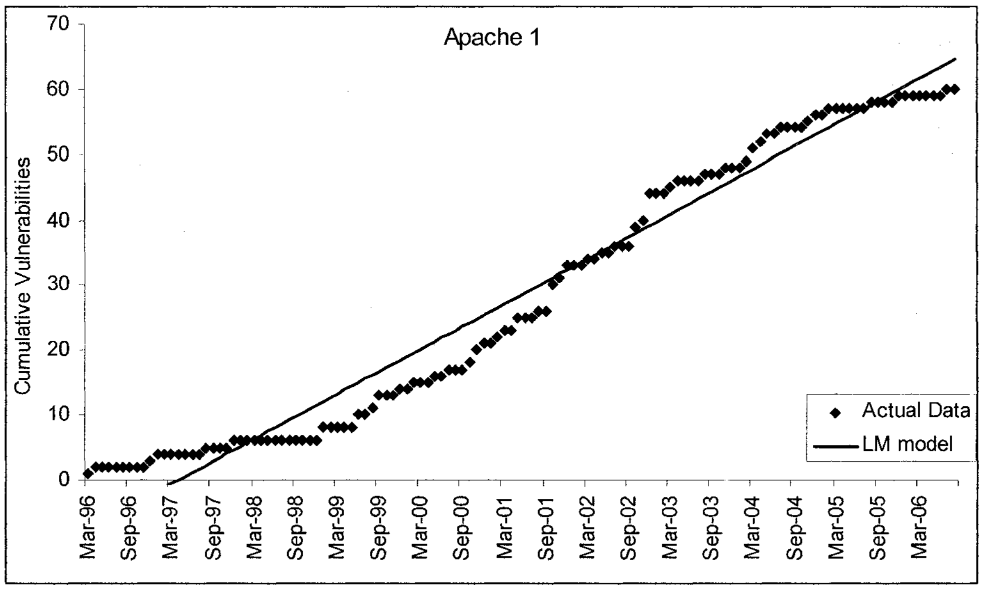

Figure 4-27. Fitting Apache 1 cumulative vulnerabilities to the linear model

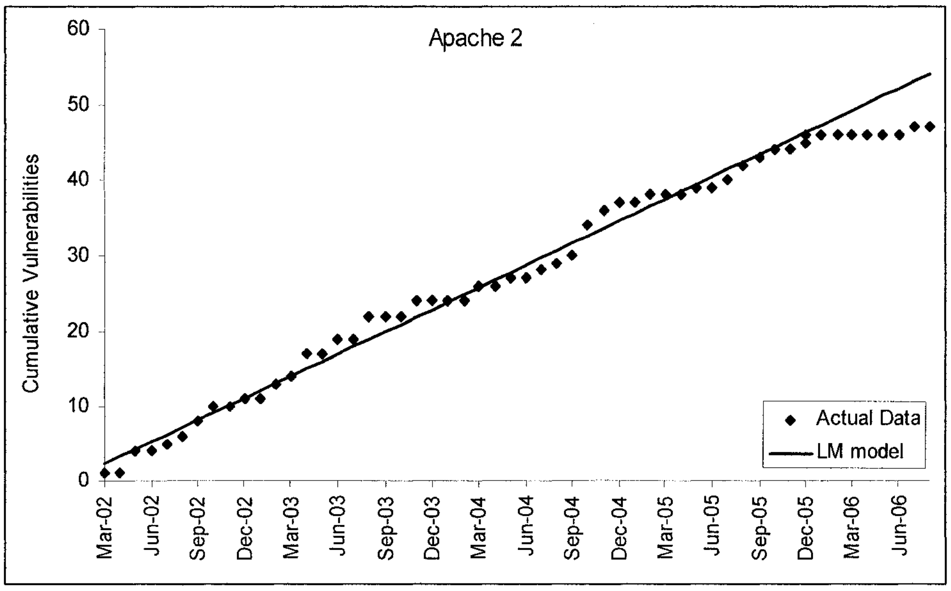

Figure 4-28. Fitting Apache 2 cumulative vulnerabilities to the linear model (LM)

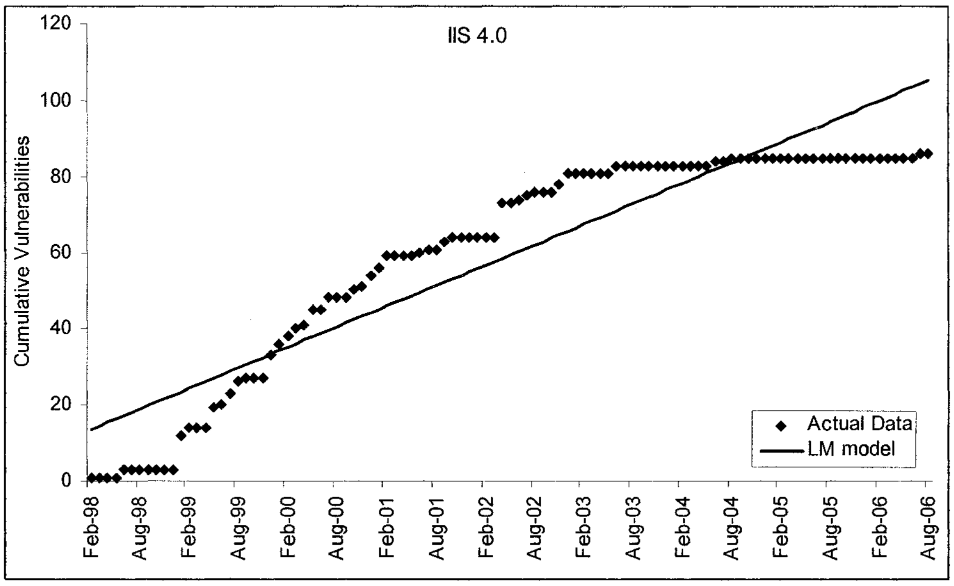

Figure 4-29. Fitting IIS 4.0 cumulative vulnerabilities to the linear model (LM)

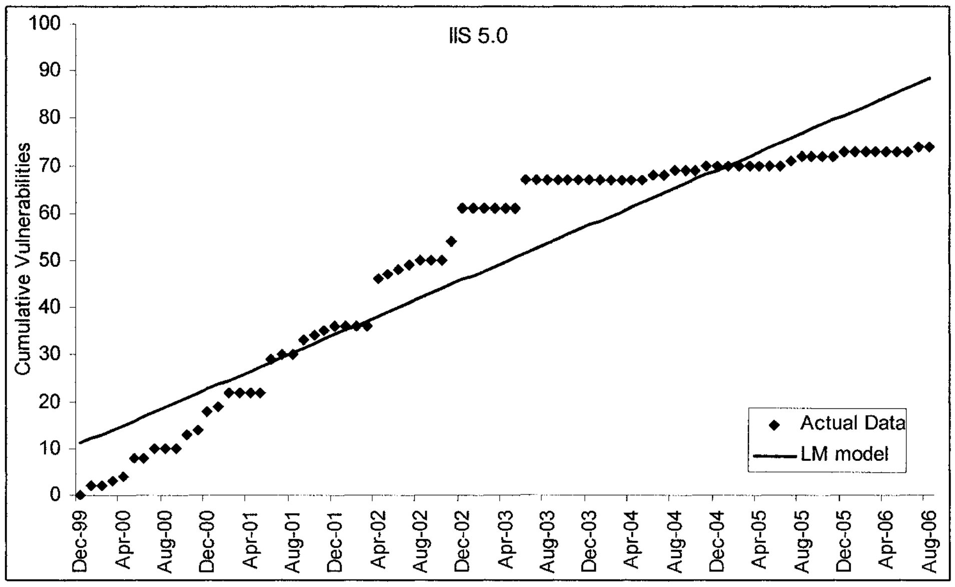

Figure 4-30. Fitting IIS 5.0 cumulative vulnerabilities to the linear model (LM)

Figure 4-29 and Figure 4-30 show fitting plots of IIS 4.0 and 5.0 respectively. Both, IIS versions appear to have been saturated for a while and therefore, they were unable to fit the linear model. Hence, we can link this to the fact that the number of IIS web servers installed appears to be stationary, unlike the Apache server which is still gaining in terms of new installations. Another possibility is that the number of remaining undiscovered vulnerabilities may actually have dropped significantly.

Table 4-5 below shows the goodness of fit results indicating that both versions of Apache have a significant fit with P-values of 0.999 and 1; nevertheless, both IIS versions failed to fit the model with P-value of 0 and 0.00002, which shows that the data is not linear.

Table 4-5. Goodness of Fit Results for the examined web server's datasets, fitted to the linear model (LM)

Software System	Slope	K	DF	χ^2	χ^2_{critical}	P-value	Significance
Apache 1.0	0.565807	-7.6272	126	63.7930	128.8039	0.999	Significant
Apache 2.0	0.91275	2.4920	54	7.58338	69.83216	1	Significant
IIS 4.0	0.9004414	12.6334	103	305.2259	127.6893	0	Insignificant
IIS 5.0	0.9649051	10.1796	81	156.2788	103.0095	.00002	Insignificant

4.4.3 Discussion

The results presented show that The AML model has fitted all four data sets, while the linear model has only fitted the two of Apache's datasets and none of the IIS's datasets. This clearly shows that the logistic model applies on all datasets because of its ability to model the three phases: learning, linear and saturation, even though if it is more visible in IIS 4.0 and 5.0 but less obvious in Apache 1.0 and 2.0. Consequently, the results imply that the logistic model previously examined on operating systems applies to web servers too. In some cases we can note that linear model is also able to fit.

4.5 MODELING VULNERABILITIES IN WEB BROWSERS

The introduction of browsers has created a powerful new medium for conducting business, commercial and personal activities. Potential discovery and exploitation of vulnerabilities has become a subject of great concern. Secure Science Corp. [37] reports that a single phishing group collected access information for 13,677 accounts by installing malicious code which exploited an unpatched vulnerability. These exploitation techniques and tools are no longer in the exclusive possession of experts; many of them are now widely available and can be relatively easy to use.

The vulnerabilities in the web browsers use a medium of spreading viruses and worms. For example, Nimda, which use the buffer overflow vulnerability, affects all Windows versions of Microsoft Internet Explorer. The vulnerability discovery rate trends provide a quantitative perspective of the problem and can be used to plan the effort needed to implement effective risk containment strategies. For example, the quantitative projections can be used to allocate resources needed for fast patch development.

In this chapter we use the data for the two major web browsers, IE and Firefox, and determine whether the vulnerability discovery trends are described by the logistic model (AML) or the linear model (LM).

IE has been the most popular web browser, utilized by approximately 85% of Internet users. This has made it a very attractive target for exploration and exploitation by malicious users. The problem has been exacerbated by the integration of IE into Windows, unlike Firefox or Mozilla. This integration provides more functions, such as immediately showing web pages or pictures as the desktop wallpaper. However, security analysts and experts consider the integration of IE to be a security disadvantage since IE connects with a variety of Windows core components. Another weakness of IE is the use of non-standard features which do not follow the W3C standard. For example, ActiveX, which supports interfaces to provide a variety of functions and is offered as an add-in only for IE, can be used for executing arbitrary code. Even though IE is known for its many security flaws, numerous Internet users still prefer to use IE because many web sites are optimized for IE and, moreover, Windows software is marketed with IE pre-installed.

Although Firefox was released in September 2002, it did not gain significant recognition until 2004. Its popularity has increased because of its perceived better security, intuitive design and multi-tap features. Currently Firefox is more common in schools or public computers and is expanding its market share. However, its popularity has led to a rising number of discovered vulnerabilities.

4.5.1 Modeling Web Browsers Vulnerabilities Using the Logistic Model (AML)

Figure 4-31 shows the cumulative vulnerabilities by month and the fitted AML model for IE 4.0. The AML model fits the data for IE 4.0 very well. At the beginning, the slope of the curve for IE rose linearly until mid-2002, after which the slope has saturated.

From the prospective of the three phases of the vulnerability discovery process, IE 4.0 appears to have entered the saturation phase. IE 5.0 shown in Figure 4-32 also shows some saturation, while the saturation is milder than IE 4.0; because, more of the later vulnerabilities found in IE 5.0 are actually shared with IE 6.0.

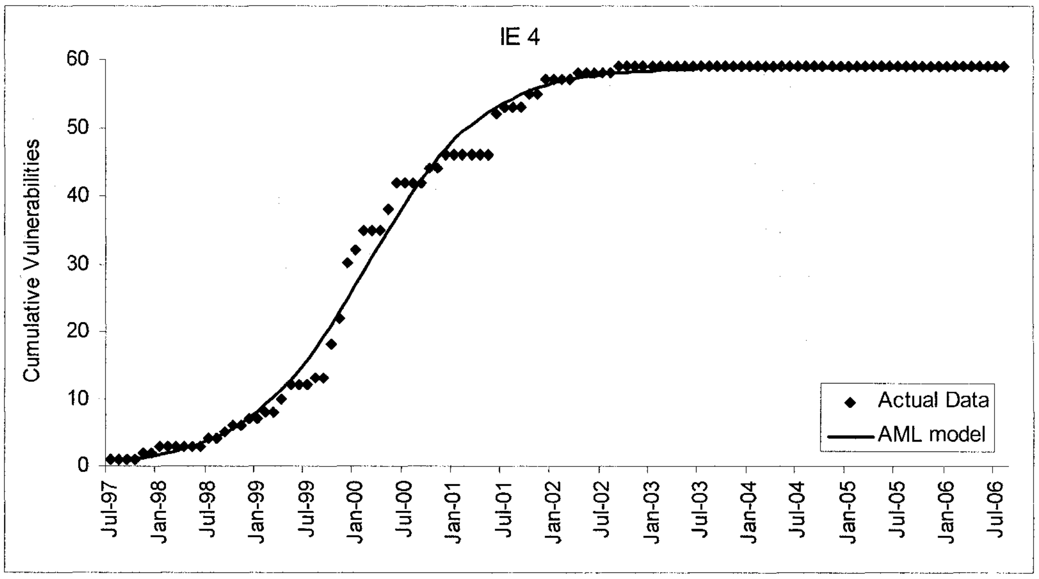

Figure 4-31. Fitting IE 4.0 cumulative Vulnerabilities to the AML model

IE 6.0 (see Figure 4-33) does not appear to have yet entered the saturation phase. Rather, IE 6.0 currently still appears to be in the linear phase, since the number of vulnerabilities is growing linearly in spite of the browsers having been on the market for several years. This may be because of its larger market share and possibly because it may have a higher number of potential vulnerabilities. Besides, IE 6.0 is being continually updated and therefore there are always new modules being introduced. Moreover, IE 6.0 is still the current web browser without a successor, so this suggests that vulnerability discovery for IE 6.0 may continue at a significant pace in the near future. It is expected that the next release of IE (i.e. IE 7.0) will have much better security.

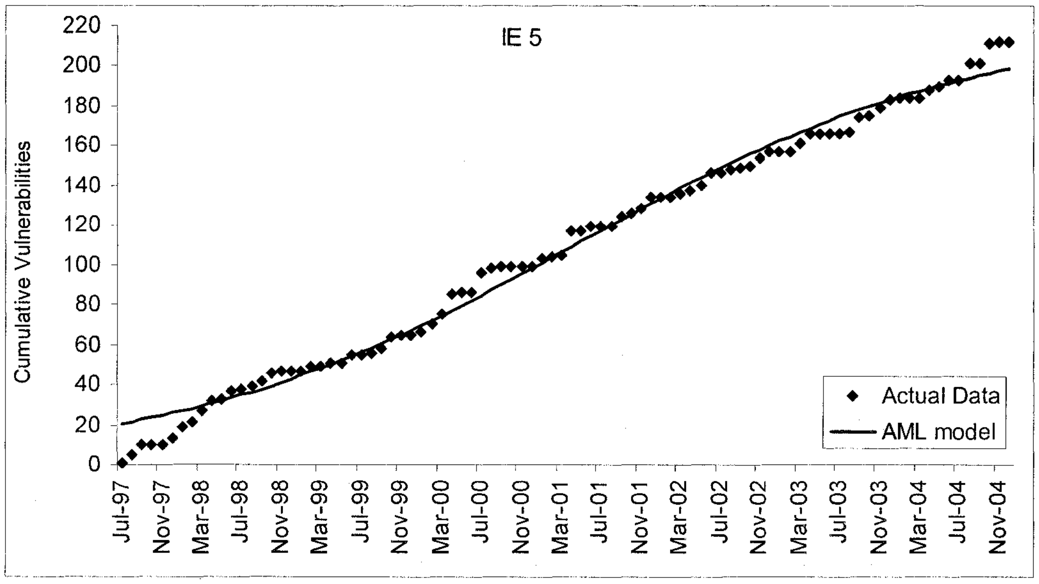

Figure 4-32. Fitting IE 5.0 cumulative Vulnerabilities to the AML model

Firefox is the second most popular web browser. While there is still a considerable market share gap between IE and Firefox, this gap is shrinking over time. Although Firefox is just four years old, and its market share is just one-eighth of the market share, the fitted model suggests that Firefox 1.5 is still in the linear phase. Consequently, we can expect that more vulnerabilities will be found in the near future, and the saturation phase is not likely to be reached soon.

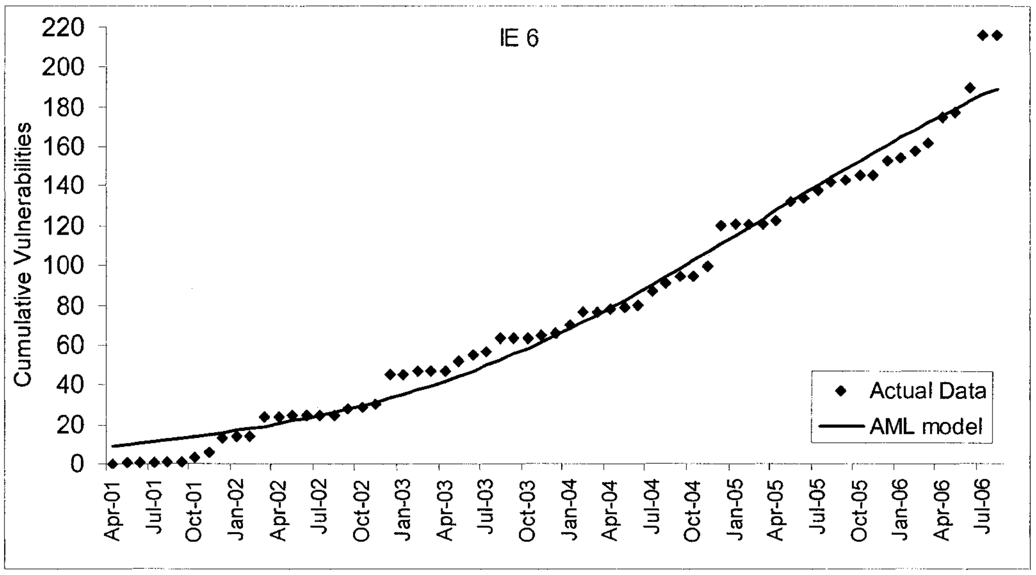

Figure 4-33. Fitting IE 6.0 cumulative Vulnerabilities to the AML model

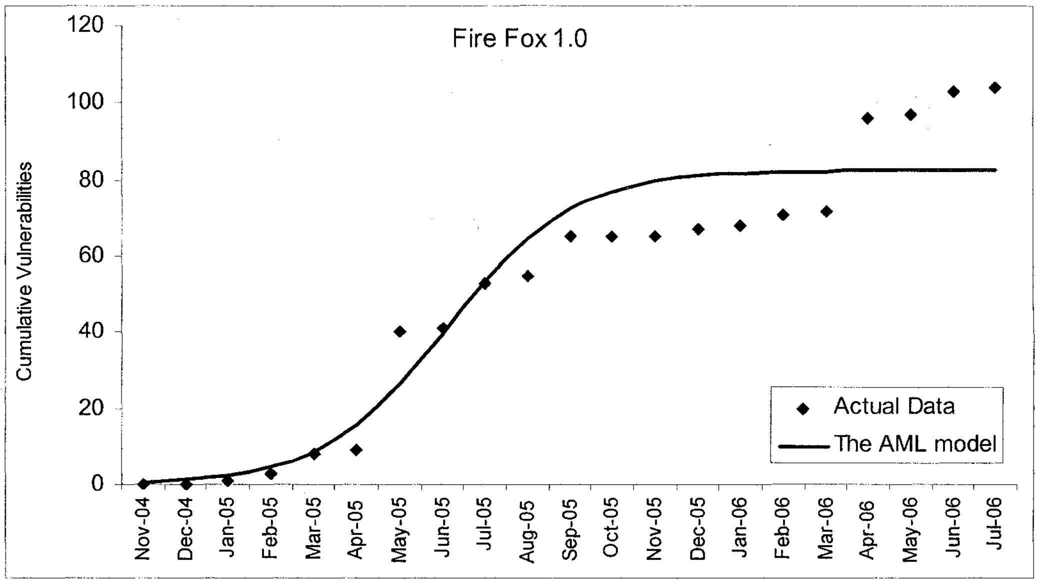

Figure 4-34. Fitting Firefox 1.0 cumulative Vulnerabilities to the AML model

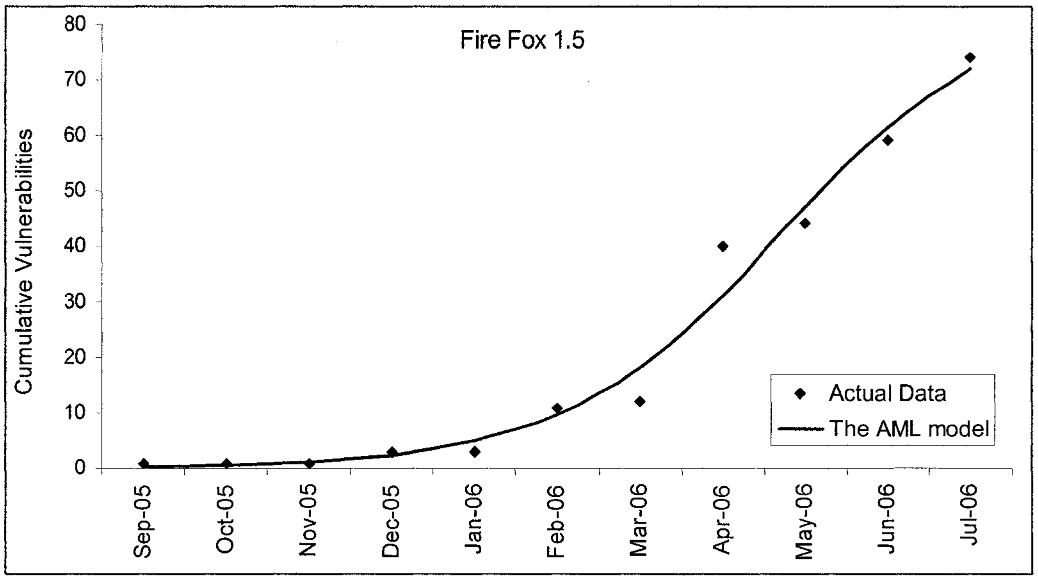

Figure 4-35. Fitting Firefox 1.5 cumulative Vulnerabilities to the AML model

Table 4-6. Goodness of Fit Results for the examined web browsers datasets on the logistic model (AML)

Software System	**A**	**B**	**C**	**DF**	χ^2	χ^2 **critical**	**P-value**	**Significant**
IE 4.0	0.00237	58.887	1.5088	110	42.853	135.48	1	Significant
IE 5.0	0.000225	221.44	0.0462	90	67.693	113.15	0.962	Significant
IE 6.0	0.000274	249.78	0.1092	65	112.44	84.821	.0002	Insignificant
FFox1.0	0.00831	82.573	49.312	25	50.565	37.652	0.0018	Insignificant
FFox1.5	0.008792	85.284	8.159	11	9.488	16.919	0.577	Significant

Another browser, Mozilla, was first released at the end of 1998. However, since Mozilla never became very popular among Internet users, very few vulnerabilities have been found in Mozilla even though it was developed long before Firefox. Only 11 vulnerabilities were found through June of 2004. A large number of vulnerabilities were first found in Firefox, followed by a similar rise in the discovery of Mozilla vulnerabilities. This is likely to be due to the fact that significant parts of code are shared between Firefox and Mozilla, demonstrating that market share can be a more important contributing factor than software age.

4.5.2 Modeling Web Browsers Vulnerabilities Using the Linear Model (LM)

The logistic model was able to fit most of the datasets examined. However, some of the dataset have some linearity. This suggests using a linear model in these cases. IE 4.0 given by Figure 4-36 clearly contradicts the linear trend, while IE 5.0 fits the linear trend closely (see Figure 4-37).

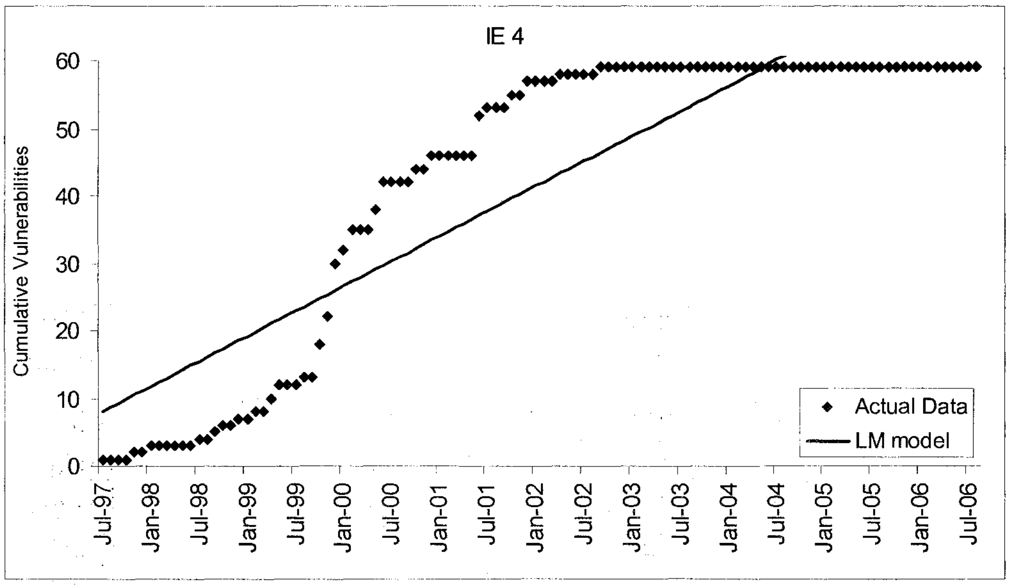

Figure 4-36. Fitting IE 4 cumulative vulnerabilities to the linear model

In Figure 4-37 the linear model was able to capture the trend until a later jump in 2006. This shows that the IE 6 is still showing an increasing rate of vulnerability discovery.

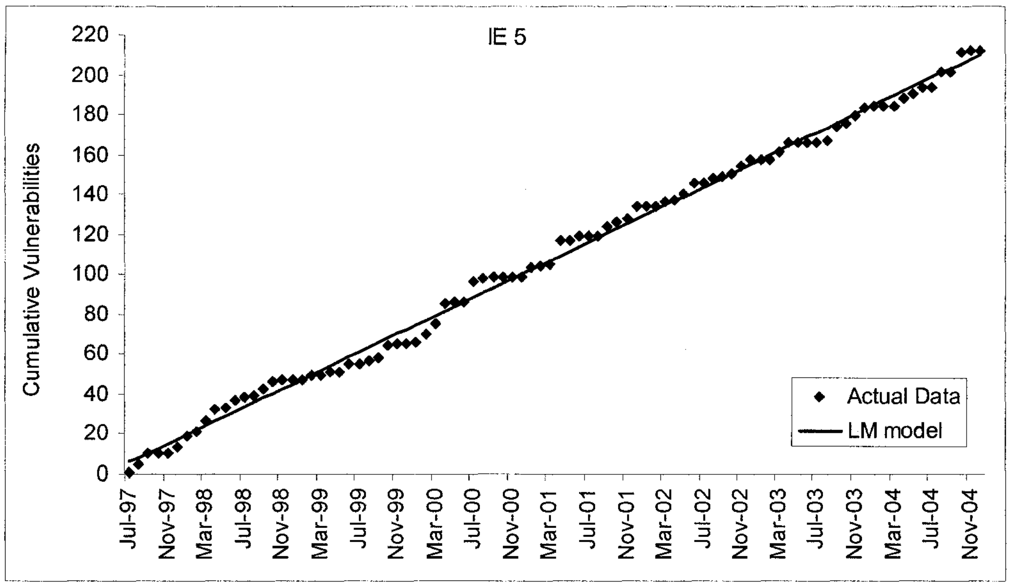

Figure 4-37. Fitting IE 5 cumulative vulnerabilities to the linear model

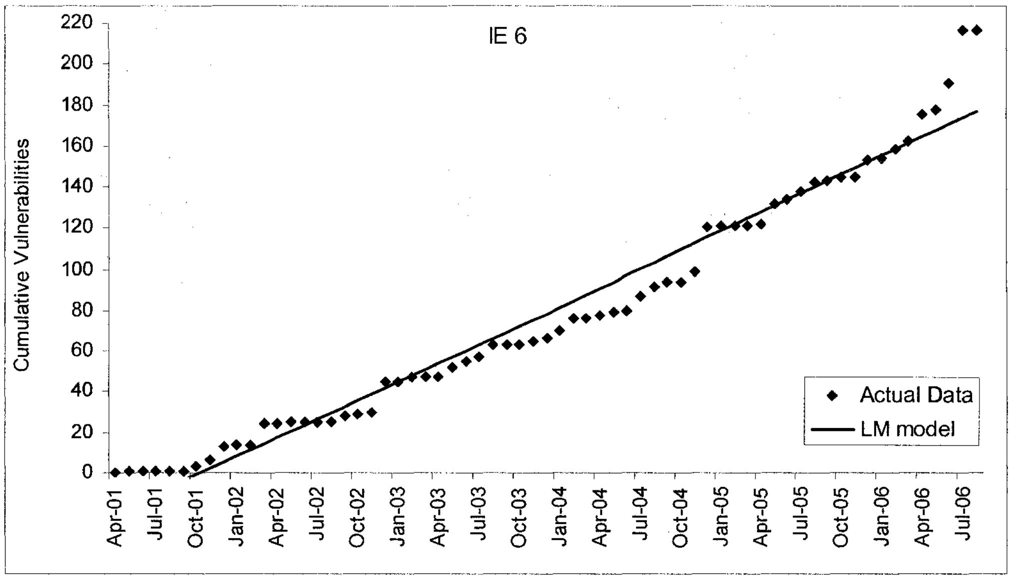

Figure 4-38 Fitting IE 6 cumulative vulnerabilities to the linear model

The linear model was unable to fit the Firefox 1.0 (see Figure 4-39), with P-value of 0.00201. It seems that Firefox 1.0 has shown a saturation followed by a jump after the release of Firefox 1.5, due to the impact of shared vulnerabilities. On the other hand, the linear model has fitted the Firefox 1.5 dataset (see Figure 4-40) with a P-value of 0.47628 as shown in Table 4-7 (Note: when calculating the chi-square value, only the positive part of the curve is considered, because chi-square only considers positive values).

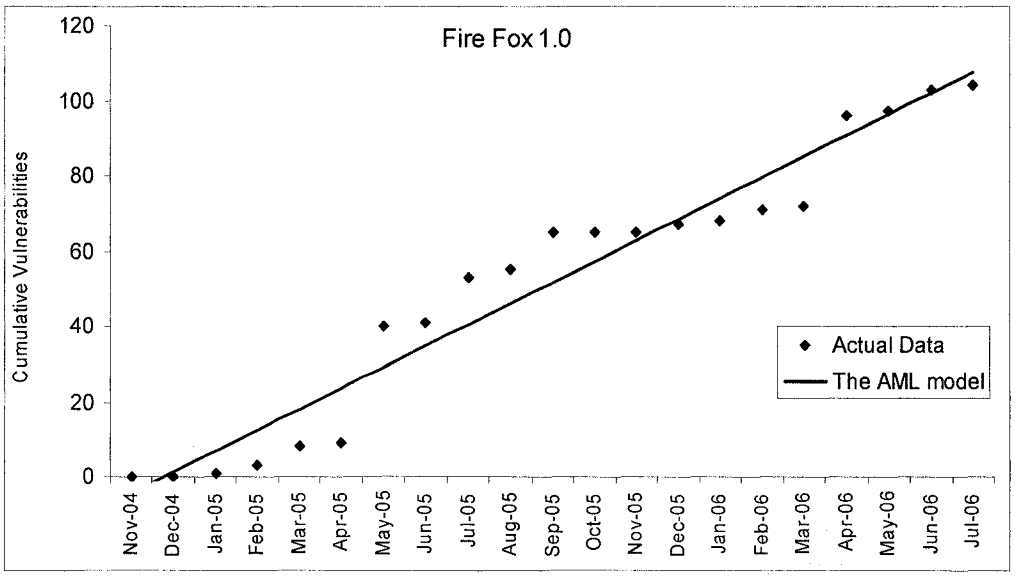

Figure 4-39. Fitting Firefox 1.0 cumulative vulnerabilities to the linear model

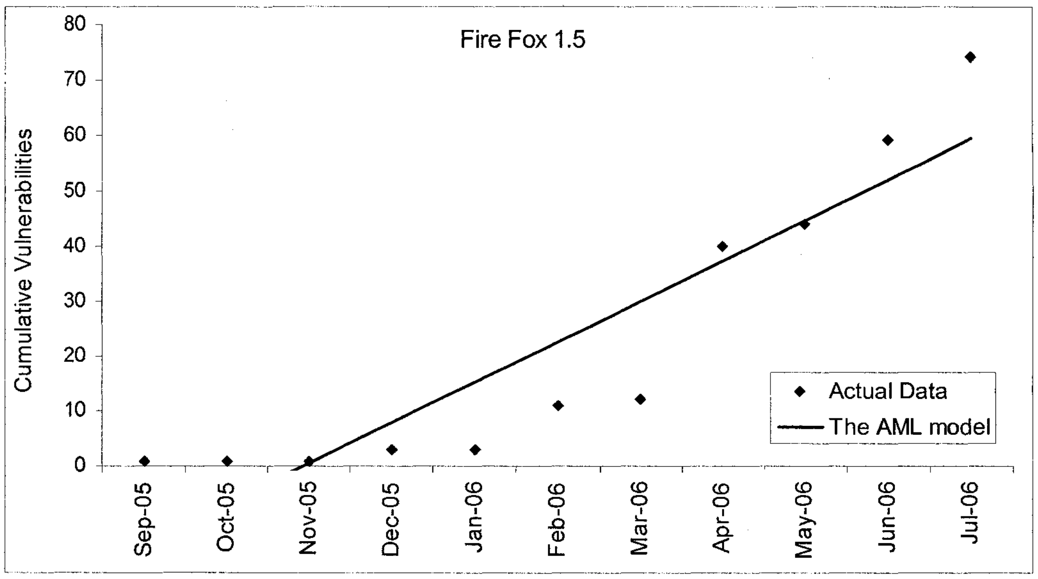

Figure 4-40. Fitting Firefox 1.5 cumulative vulnerabilities to the linear model

Table 4-7. Goodness of Fit Results for the examined web browsers datasets on the linear model (LM)

Software System	Slope	C	DF	χ^2	χ^2_{critical}	P-value	Significance
IE 4	0.620395	7.395333	110	385.0309	134.48	0.00000	Insignificant
IE 5	2.293205	3.614732	90	31.16308	113.1453	1.00000	Significant
IE 6	3.065297	-22.4779	65	94.55107	84.82065	0.00977	Insignificant
FFox 1.0	-32.27	5.5896	25	50.21824	37.65249	0.00201	Insignificant
FFox 1.5	-21.49	7.3545	11	10.61238	18.30703	0.47628	Significant

4.5.3 Discussion

We have examined the data sets to see if the discovery process tends to follow some specific patterns and if these patterns can be modeled. The results show that when the aggregate number of vulnerabilities is examined, the AML model fits the datasets well, as shown in Table 4-6.

The fitting was done for the classes of vulnerabilities for which the available data is statistically significant. It would be difficult to use such models for the types of vulnerabilities that occur less frequently because the data may not be sufficiently statistically significant to make meaningful projections.

Examining the current vulnerability discovery trends for the three web browsers, reveals that none all of them has reached the saturation phase except for IE 4.0 which show obvious saturation. Thus, the other four systems (IE 5.0, IE 6.0, Firefox 1.0, and Firefox 1.5) are expected to show a high rate of vulnerability discovery in the near future until they show some signs of saturation.

When comparing browser security, we need to keep in mind that the vulnerability discovery rate in near future may be more important than the vulnerabilities already discovered in the past. Other factors to consider include severity levels and quick availability of patch releases. Currently, some experts regard IE to be less secure, pointing to integration of IE into Windows and Active X. Secunia [74] reports that IE versions have more unpatched vulnerabilities than Firefox. However, if Firefox popularity keeps rising, it will attract more people trying to discover its vulnerabilities. The new version of IE 7.0, currently in beta version, is claimed to be more secure because of the incorporation of new security features.

In this chapter we have examined trends in vulnerability discovery in web browsers and explored the applicability of quantitative models for the number of vulnerabilities. These models gave us an insight into the vulnerability discovery process. The results showed that the AML vulnerability discovery model generally tracks the available data well.

The results indicate that the models originally proposed and validated for operating systems and HTTP web servers are also applicable to web browsers. These models can be used to estimate vulnerability discovery rates, which can be integrated with risk assessment models in the future.

4.6 MODELING VULNERABILITIES USING THE EFFORT-BASED MODEL

The effort based model presented in section 3.3.3 needs monthly market share data available in order to compute equivalent effort. The model uses equivalent effort instead of time as an independent variable. This model needs to be validated when the required data is available.

Here, we fit the data of Windows 98, Windows NT 4.0, IIS, and Apache data to the Effort-Based model; also, the goodness of fit is measured to determine if the fit is significant. The fit is done using the least square-fit method. For the Chi-square goodness of fit test; we chose alpha level as 5%. So, if P-value is greater than 0.05 it means that the fit is significant at this alpha level.

4.6.1 Validating the Effort-Based model on Operating Systems

To examine the effort-based model given by Equation (10), we used the same vulnerability data of Windows 98, together with the corresponding data on the share of installed base and the number of internet users. The data fit the model, as is shown in Figure 4-41 and Figure 4-42. Table 4-8 gives the parameter values obtained. The Chi-square values are given in Table 4-8. The Chi-square values are less than the critical values (this also implies that P-values are greater than 0.05) and therefore the fit is acceptable.

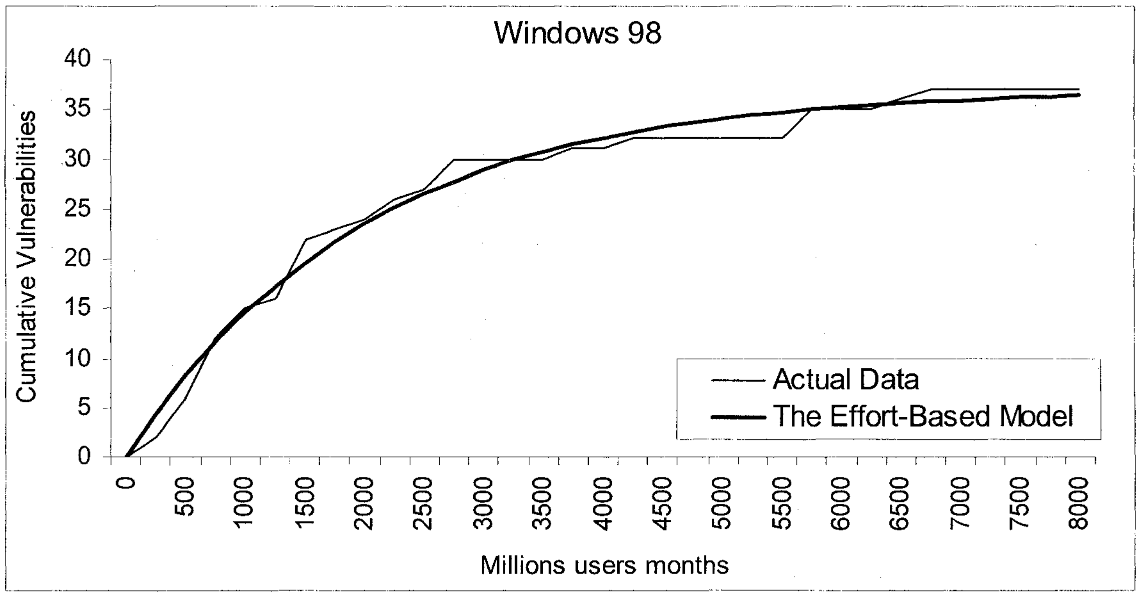

Figure 4-41. Fitting Windows 98 data (Effort-Based Model)

For the two operating systems, both models fit well. The time-based model is easier to use because it does not require evaluation of E. However the effort-based model removes the variability due to change of effort during the life-time of the OS.

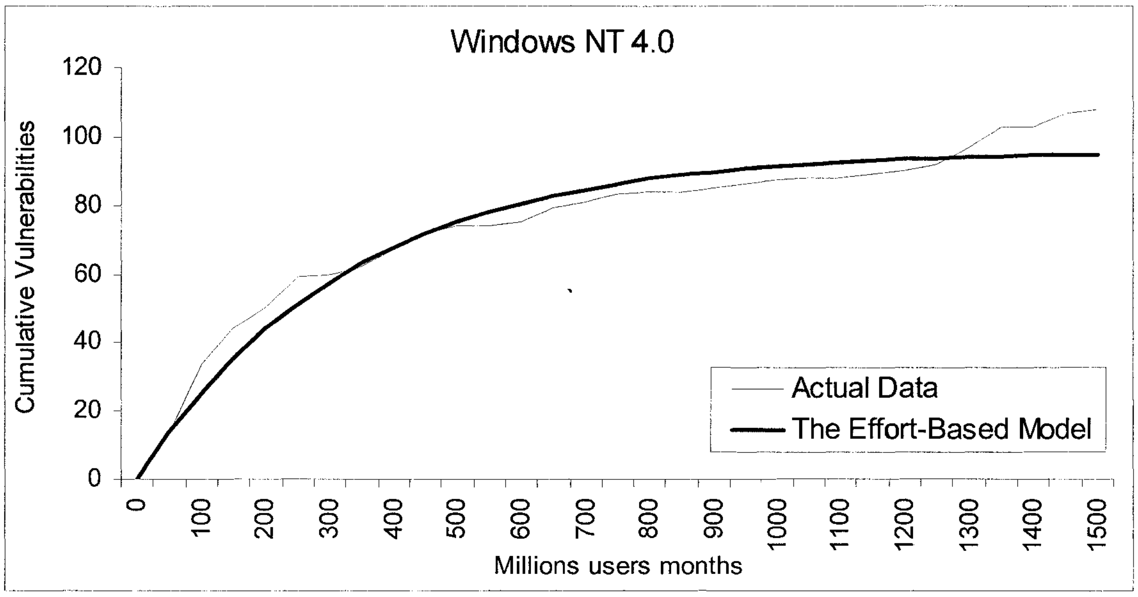

Figure 4-42. Fitting Windows NT 4.0 (Effort-Based Model)

Table 4-8. χ^2 Goodness of fit test results for fitting the Effort-Based Model to vulnerabilities of some versions of Windows

	B	λ_{VU}	χ^2	$\chi^2_{critical}$	P-value	Significance
Windows 98	37	0.000505	3.510	44.9853	1	Significant
Windows NT 4.0	108	0.00306	15.05	42.5569	0.985	Significant

4.6.2 Validating the Effort-Based model on HTTP Servers

Figure 4-43 shows cumulative vulnerabilities by number of Apache installations in terms of million system-months and the fitted effort-based model. This effort-based model shows that Apache has not yet entered the saturation phase, since the number of vulnerabilities continues to increase approximately linearly as the number of Apache servers increases [77].

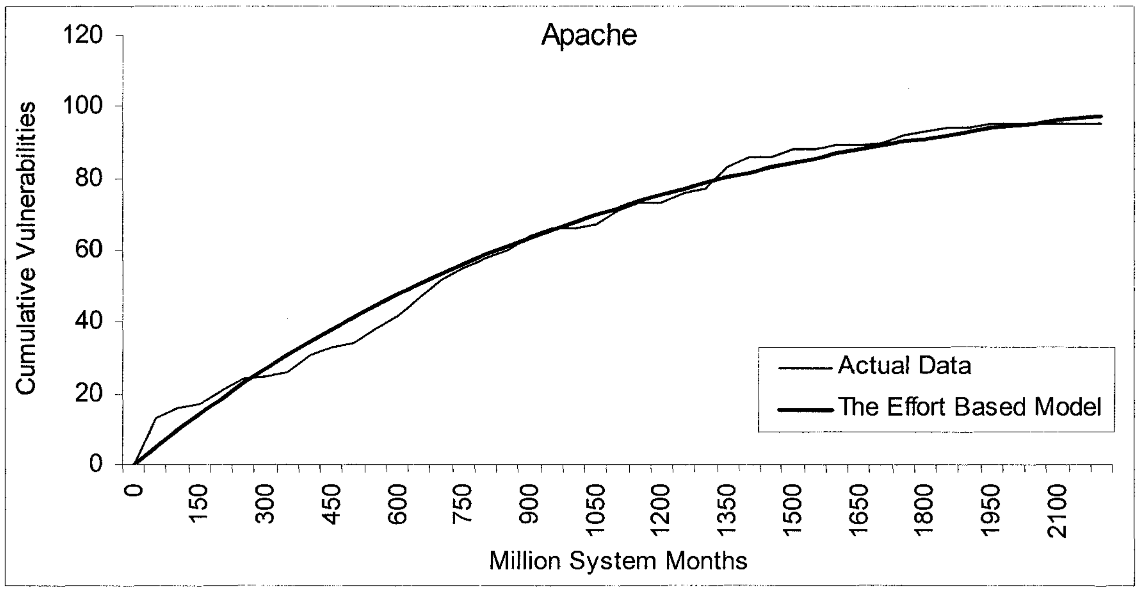

Figure 4-43. Fitting Apache vulnerability data to the Effort Based Model

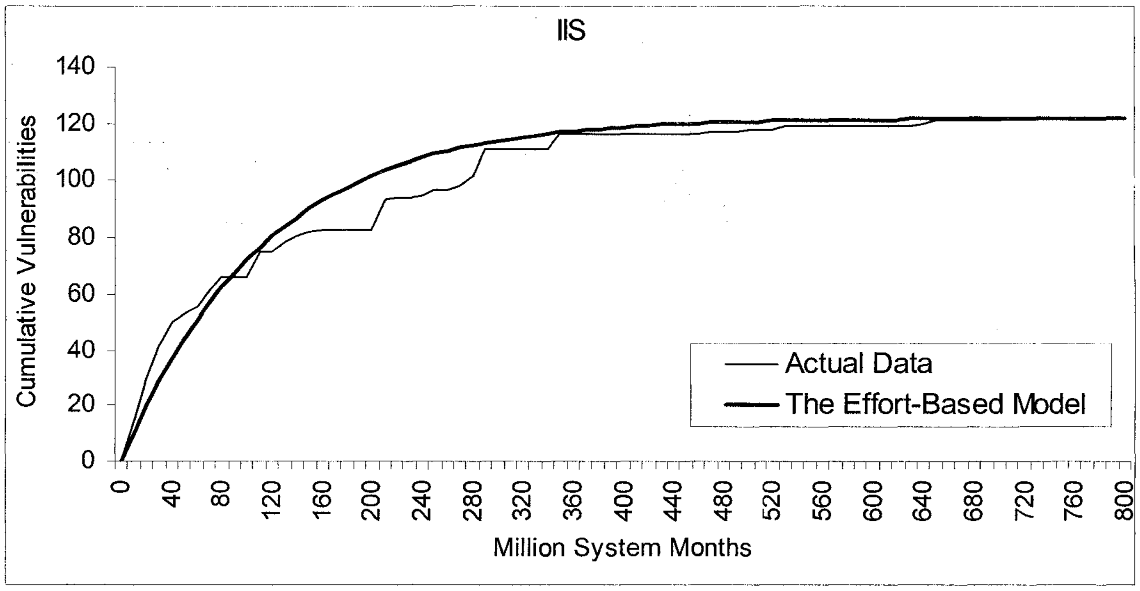

Figure 4-44. Fitting IIS vulnerability data to the Effort Based Model

IIS was released in the early part of 1996. IIS is a popular commercial web server with about 15 million installations currently. We have used the vulnerability data from January 1997 to May 2006.

Figure 4-44 shows cumulative vulnerabilities for the IIS server and the effort-based model by million system-months [77].

Table 4-9. χ^2 Goodness of fit test results for fitting the Effort-Based Model to some HTTP servers

	B	λ_{VU}	χ^2	χ^2_{critical}	**P-value**	**Significance**
Apache	112.5	.00092	23.726	61.66	.992	Significant
IIS	122	.009	46.6	103	.998	Significant

We examine the fit of the models to the data as shown in Figure 4-43 and Figure 4-44. For χ^2 goodness of fit test, we chose an alpha level of 5%. Table 4-9 gives the chi-square values and parameter values for both the time-based and effort-based models. The table shows that the chi-square values are less than the critical values; this demonstrates that the fit for Apache and IIS is significant. Both data sets fit both models with χ^2, with high P-values ranging from 0.992 to 0.998, indicative of the fit significance.

4.6.3 Discussion

The data of two operating systems, Windows 2000 and Windows 98, was fitted to the effort based model. This was possible because market share data for those two software systems is available; therefore, the equivalent effort is computable. The result shows a significant fit to the Effort-based model.

Moreover, the data of two of the most widely used web servers were fitted to the effort based model; this was also possible because market share data for those two software systems is available; also here, the result shows a significant fit to the Effort-based model.

The result suggests that when future market share data projection is available and accurate, we can apply the model to predict vulnerabilities more accurately because the model considers changes in market share and hence in the rewards associated with the attacks.

4.7 SHARED VULNERABILITIES IN SUCCESSIVE VERSIONS OF SOFTWARE SYSTEMS

Here we take a closer look at shared vulnerabilities. Shared vulnerabilities often occur between consecutive versions of software systems; usually the amount of shared vulnerabilities is strongly correlated with the size of reused code. Consequently, this can differently affect the goodness of fit of the AML logistic model, because they may cause what appears like another AML model forming just after it.

Figure 4-45 below shows that the new vulnerabilities of Windows NT 4.0 discovered after June 2004 are vulnerabilities shared with Windows 2000 since those two curves are parallel.

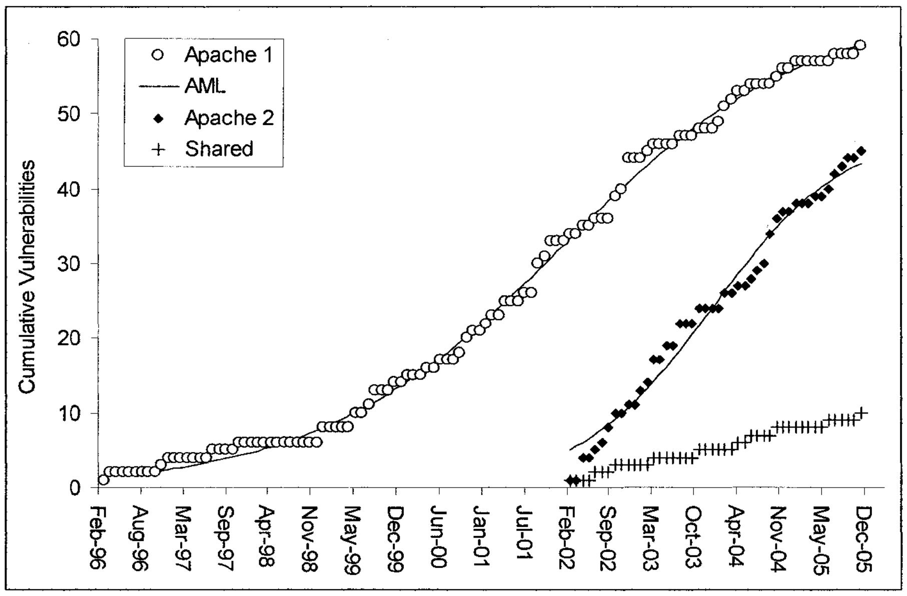

Figure 4-45. Plot of the Vulnerabilities of Windows NT 4.0 and Windows 2000 showing shared vulnerabilities

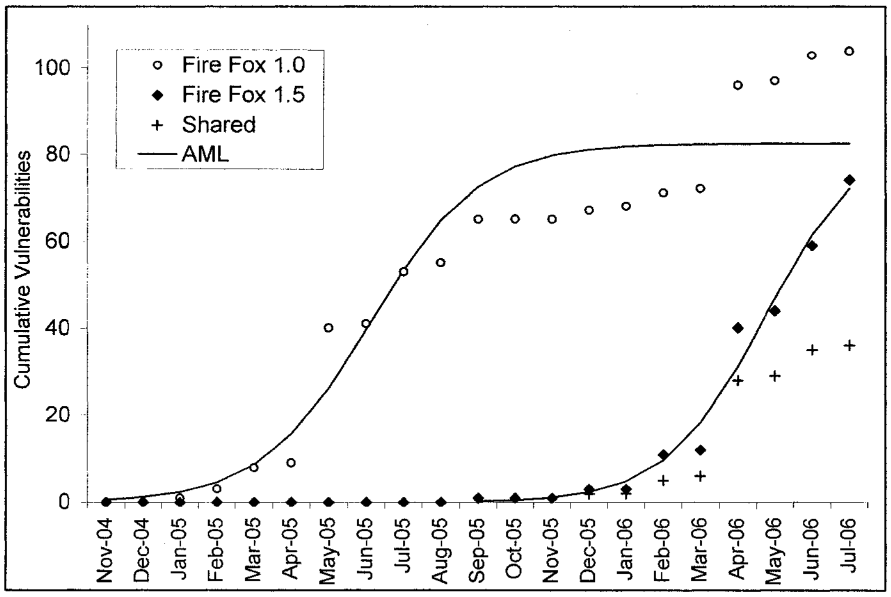

Figure 4-46. Plot of the Vulnerabilities of Fire Fox web browser versions 1.0 and 1.5 showing shared vulnerabilities

Figure 4-46 above shows the shared vulnerabilities between Fire Fox browsers version 1.0 and 1.5. We can notice that for Fire Fox 1.0 the data started to saturate for a few months, then after version 1.5 became available, a jump in the number of vulnerabilities occurred. This jump consists of shared vulnerabilities as the figure shows. The figure shows that both shared vulnerabilities and vulnerabilities of Fire Fox 1.0 are parallel for the latter part.

Figure 4-47 below shows the vulnerabilities shared between Apache 1.x and 2.x. It shows the AML fitted to both datasets. In this example, the number of shared vulnerabilities is less significant than observed in earlier plots. The reason behind this is that Apache 1.3 and Apache 2.0 are being developed independently as they took different tracks; therefore, they do not fit the definition of the typical successive versions. However, they still share some code and shared vulnerabilities are found but not in large numbers.

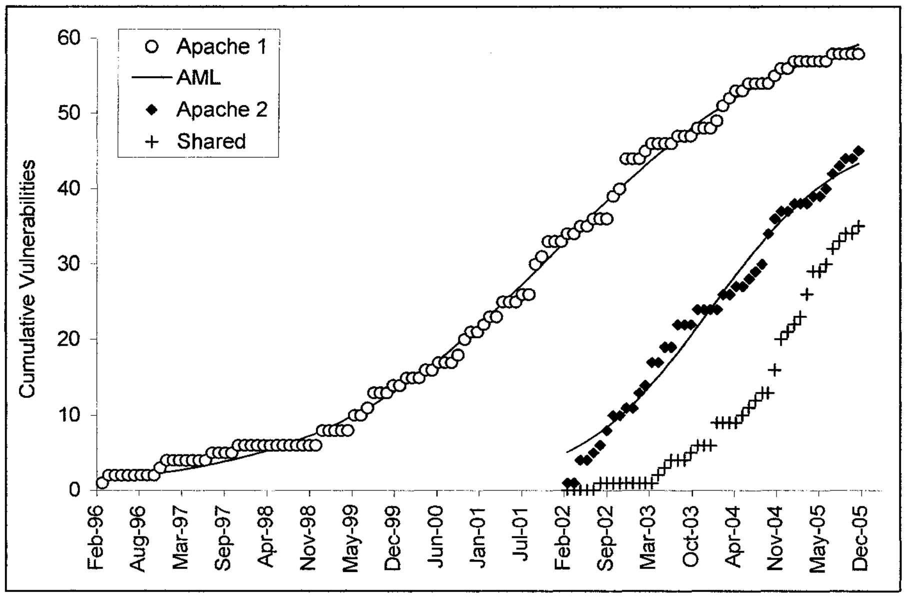

Figure 4-47. Plot of the Vulnerabilities of Fire Fox web browser versions 1.0 and 1.5 showing shared vulnerabilities

4.8 CONCLUSION

This chapter has examined the goodness of fit of several datasets to the three models. Generally, for practical reasons, Vulnerability discovery models assume that each specific release of an operating system is independent and can be separately modeled. However, in practice, a significant sharing of the code occurs between successive releases.

The results have shown that the Alhazmi-Malaiya Logistic model has fitted the vast majority of the datasets; however, it did not fit in some exceptional cases. The Alhazmi-Malaiya Logistic model did not fit when some factors are presents; here are two examples where the AML model show insignificant fit:

1. There is a case of super imposition of more than one logistic model in the data. For example, when there are consistent changes applied to the software system; consequently, when the new modules are introduced, they themselves are not

thoroughly tested; therefore, they will keep the so trend from saturation artificially. Here, if data were to be separated for smaller periods of time, more than one model can be fitted, and it is expected that these smaller models applied to those datasets would fit significantly

2. There are a significant number of vulnerabilities which are shared with a successive version. So, for example, a system that has a declining small market share, and still showing high vulnerability discovery rate, it is most likely that these new vulnerabilities are actually being discovered in another version that has a larger market share and shares some source code with it.

The linear model has shown to fit in a number of cases to some of the datasets. Because the linear model was able-in some cases- to neutralize some factors affecting the data, it was found especially applicable in the following cases:

1. The software is still young or in mid-life, so it is in the linear phase of its life.
2. The software is growing, by adding new modules and patches that themselves keep the dataset trend go linear. So, the linear model will take advantage of the linear behavior that occurs when there is superimposition of more than one AML model.
3. There is a significant proportion of shared code with a successive version.

The effort based model has shown to fit all examined datasets; however, it requires some market share data, and therefore, there is limitations on using the model on software systems market share is unavailable or not properly filtered.

Overall, the AML model and the Effort-based model were the most successful, followed by the linear model.

CHAPTER 5

COMPARING VULNERABILITY DISCOVERY MODELS

5.1 INTRODUCTION

Software reliability growth models [44][55] have been used for characterizing the defect-finding process. Such models are used to assess the test resources needed to achieve the desired reliability level by the target date and are needed for evaluating the level of reliability achieved. They can also be used to estimate the number of residual defects that are likely to be present. With the presence of a more than one vulnerability discovery model there is a need to compare existing models.

In the security field, *vulnerability exploitation models* (VEMs) were the first to be considered. Browne et al. [20] have examined the exploitation of some specific vulnerabilities and have presented a modeling scheme. Some researchers have recently examined investigations on the modeling of the vulnerability finding process [4][13][14][64] [69]. We examine and evaluate these models, termed *vulnerability discovery models* (VDMs), in this chapter. An evaluation of the risk would involve both characterizations of vulnerability discovery process as well as its potential exploitation. This evaluation of the risk involves both VEMs and VDMs.

The first VDM model proposed by Anderson [14] is here termed the Anderson Thermodynamic (AT) model. The second, termed the AML model, is a logistic model proposed by Alhazmi and Malaiya in [4] and investigated in [5]; its prediction capabilities were examined in [7]. Two trend models were examined by Rescola in [69] , a quadratic model and an exponential model; these are termed RQ and RE respectively.

In addition, we examine an LP model which is an application of a traditional Logarithmic Poisson reliability growth model [54] proposed by Musa and Okumoto. Additionally, we test a simple linear model as an approximation of the AML model. All of these can be termed time-based models since they consider calendar time as the main factor. An effort-based model has also been proposed by Alhazmi and Malaiya in [4]. It will not be considered in this context since it utilizes a different approach that attempts to use test-effort as the main factor rather than calendar time. In this research, we briefly examine the basis of the proposed models and evaluate the applicability of these models using actual vulnerability data.

Just as static models for defect density and fault exposure ratio can assist in the use of SRGMs, the metric vulnerability density and vulnerability/defect ratio can be applied to complement and support VDMs. Alhazmi and Malaiya [4] have shown that for similar systems, the values of these attributes tend to fall within a range. Static metrics can be used to constrain parameter estimation during fitting [5].

This chapter previews some of the existing models and presents a comparison of the models' adequacy using actual data for vulnerabilities in four major operating systems. The statistical goodness of fit test is used to examine how well models track the actual discovery process. The Akaiki information criterion is used to measure the adequacy of the models for the purpose of comparison. In the next section, we discuss the proposed models. Then, we explain the comparison methodology, and finally, we fit the datasets to the models and present the comparison results in plots and in tables. Finally, the results are discussed and we conclude.

5.2 PREVIEW OF VULNERABILITY DISCOVERY MODELS

In this section all models are previewed; however, the logistic model will be previewed briefly, because it was already previewed in section 3.3.1 (The logistic model will be referred to as Alhazmi-Malaiya Logistic model (AML)).

5.2.1 ANDERSON THERMODYNAMIC MODEL (AT)

This model was originally proposed for software reliability in [21]. Later, in [14], Anderson applied it to vulnerabilities. Figure 5-1 shows a hypothetical plot of the AT model for different values of k/γ and C. Let us suppose that there are $N(t)$ vulnerabilities left after t tests, and let the probability that a test fails be $\omega(t)$. The model assumes that encountering a vulnerability causes it to be removed and also that no bugs are re-introduced.

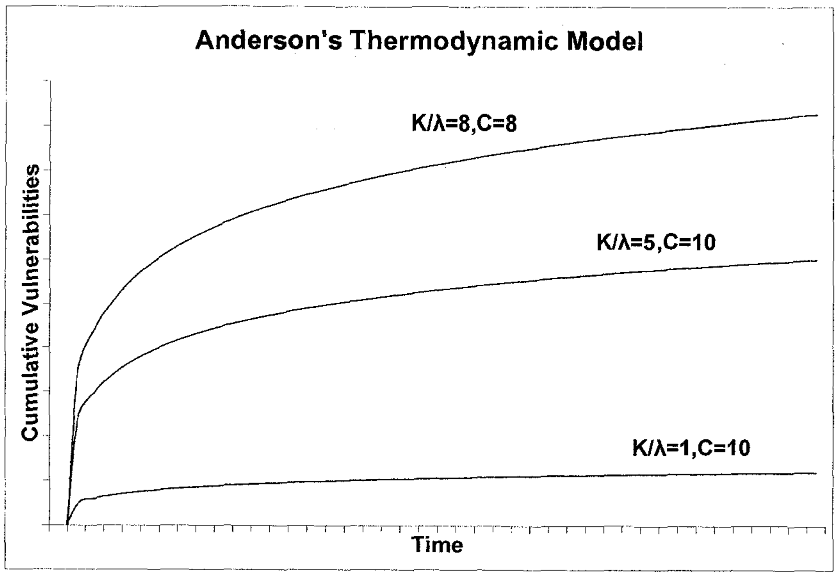

Figure 5-1. Hypothetical plots of the Anderson Thermodynamic (AT) model

Using an analogy from thermodynamics, Anderson argues that $\omega(t) \leq k / t$, where k is a constant. Arguing that equality should be a reasonable approximation, he finally arrives at the model

$$\omega(t) = \frac{k}{\gamma t}, \tag{13}$$

where γ is a factor that takes into account the lower failure rate during beta testing by the users compared to alpha testing. Because we want to compare cumulative models, we integrate Equation (13) to obtain the model in terms of the cumulative number of vulnerabilities given by the function *Ω(t)* as follows:

$$\begin{aligned}\Omega(t) &= \int\left(\frac{k}{\gamma t}\right)dt \\ &= \frac{k}{\gamma}\ln(t) + \left(\frac{k}{\gamma}\ln(C)\right)\end{aligned}$$

where $\left(\frac{k}{\gamma}\ln(C)\right)$ represents the integration constant. By simplifying we obtain,

$$\Omega(t) = \frac{k}{\gamma}\ln(Ct), \tag{14}$$

where C is a constant introduced by the integration. Note that this Ω(t) in this model is not defined when t=0; hence, we will only consider its applicability when $t \geq 1$. As t grows, Ω(t) grows with logarithmic increase. It should be noted that this model has a relationship to the well-known Logarithmic Poisson SRGM and the failure rate bound proposed Bishop and Bloomfield [19].

5.2.2 Alhazmi-Malaiya Logistic Model (AML)

This model was proposed by Alhazmi and Malaiya in[4]. The AML model is based on the observation that the attention given to an operating system increases after its introduction, peaks at some time and then drops because of the introduction of a newer competing version. Thus, the vulnerability discovery rate increases at the beginning, reaches a steady rate and then starts declining. The cumulative number of vulnerabilities thus shows an increasing rate at the beginning as the system starts attracting an increasing share of the installed base. After some time, a steady rate of vulnerability finding yields a linear curve. Eventually, as the vulnerability discovery rate starts dropping, there is saturation due both to reduced attention and a smaller pool of remaining vulnerabilities.

$$\Omega(t) = \frac{B}{BCe^{-ABt} + 1} \tag{15}$$

It is a three-parameter model given by the logistic function (15). The equation shows that as t approaches infinity, y approaches B. Thus, the parameter B represents the total number of accumulated vulnerabilities that will eventually be found (see Figure 5-2). (For more details about the AML model see section 3.3.1).

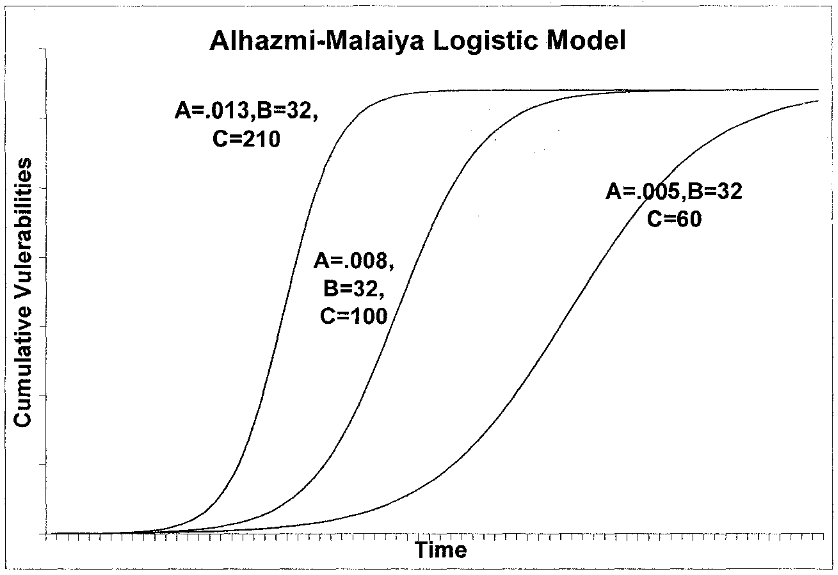

Figure 5-2. Hypothetical plots of the Alhazmi-Malaiya Logistic (AML) model

5.2.3 Rescola Quadratic Model (RQ)

Rescola has attempted to identify trends in the vulnerability discovery data by applying some statistical tests. We here refer to these tests as the linear model and exponential model [69] . Figure 5-3 shows a hypothetical plot of the RQ model for various values of *B* and *K*.

First, rather than using the cumulative data, Rescorla has fitted a basic linear model to the vulnerability finding rate. In his approach, vulnerabilities were grouped in three-month time periods. The linear fit for the failure rate ω(t) implies the model:

$$\omega(t) = Bt + K, \tag{16}$$

where B is the slope and K is a constant, while both are regression coefficients. The cumulative vulnerability discovery model can be derived by integrating Equation (16):

$$\Omega(t) = \int (Bt + K)dt$$

$$\Omega(t) = \frac{Bt^2}{2} + Kt \tag{17}$$

where the integration constant is taken to be zero to allow Ω(t) to be zero at t =0. In this model, as t grows, Ω grows polynomially, as given by the squared term.

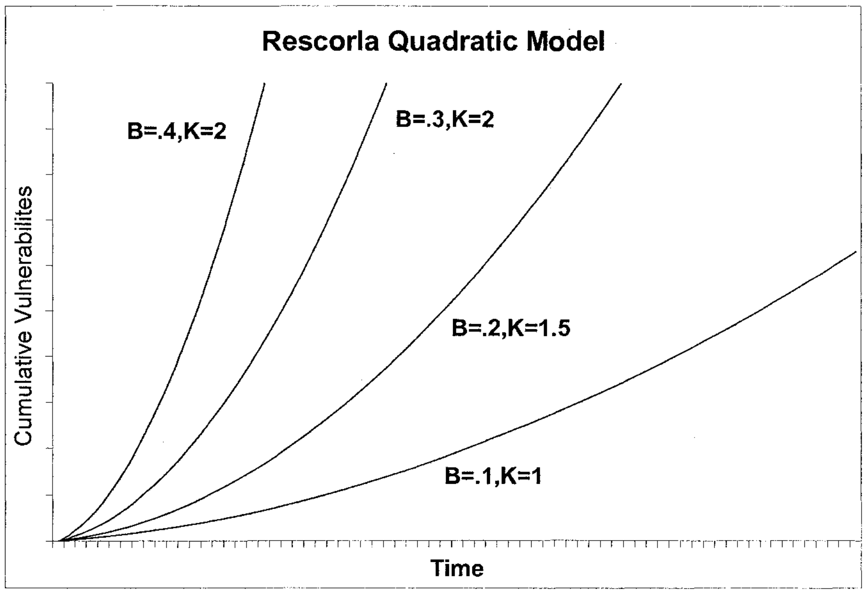

Figure 5-3. Hypothetical plot of the Rescorla Quadratic (RQ) model

5.2.4 RESCOLA EXPONENTIAL MODEL (RE)

Rescorla [69] has also used Goel-Okumoto SRGM [28] to fit the data. This exponential model can be given as:

$$\omega(t) = N\lambda e^{-\lambda t}, \tag{18}$$

where N is the total number of vulnerabilities in the system and λ is a rate constant. Again, we here integrate (18) to get the cumulative number of vulnerabilities. Figure 5-4 shows some plots of RE model for different values of λ, and N.

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

$$\Omega(t) = N(1 - e^{-\lambda t}), \tag{19}$$

where the integration constant has been equated to N to allow the initial value of Ω to be zero. As time grows, Ω approaches N.

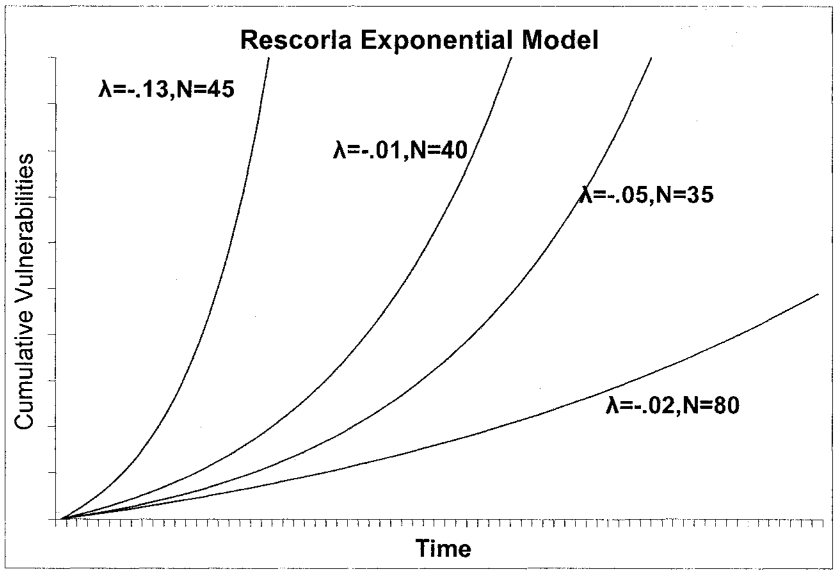

Figure 5-4. Hypothetical of the Rescorla exponential (RE) model

5.2.5 LOGARITHMIC POISSON MODEL (LP)

This model is also known as the Musa-Okomoto model [54]. A physical interpretation of the model and its parameters is complex. An interpretation in terms of the variability of the fault exposure ratio is given in [48]. Figure 5-5 shows various plots of the LP model for different values of β_0 and β_1.

$$\Omega(t) = \beta_0 \ln(1 + \beta_1 t), \tag{20}$$

where β_0 and β_1 are regression coefficients. At t=0, $\Omega(t) = 0$; Ω (t) grows indefinitely as the system ages with logarithmic growth. In spite of the fact that the parameters have a complex interpretation, in many cases the model has been found to be among the better fitting SRGMs.

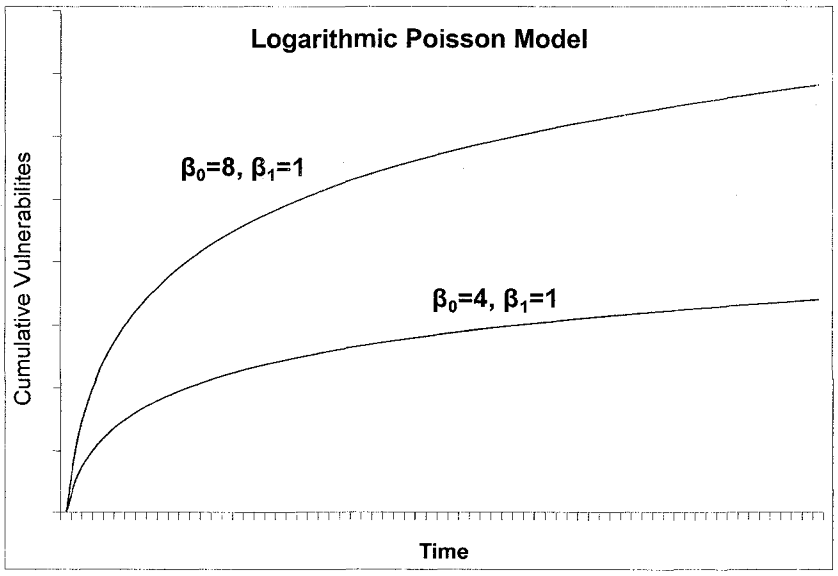

Figure 5-5. Hypothetical plots of the Logarithmic Poisson (LP) model

Some of the models are somewhat related. The AT and LP models are given by rather similar expressions, with the significant difference that AT is undefined at t = 0. It can be shown that RE and LP may yield similar short term projections, but they differ significantly for very large values of *t*.

5.2.6 The Linear Model (LM)

This model is a basic linear model; it is suggested as an approximation to the logistic model. It has shown good performance when the learning phase is short and when there is a number of shared vulnerabilities with a successor version. The equation below shows the linear model:

$$\Omega(t) = vt + u\,, \tag{21}$$

where v and u are regression coefficients. At *t=0, Ω(t)=* 0; *Ω(t)* grows indefinitely as the system ages with a linear growth in spite of the fact that the parameters have a complex interpretation.

5.3 COMPARING THE MODELS

Here, we discuss how the data was collected and prepared for fitting, and then we describe how the goodness of fit was evaluated.

5.3.1 The data sources

Compared to data that has been used for SRGMS in the past, the vulnerability data available has some different characteristics. One of the main differences is that generally no information is available concerning the faults in SRGM data, whereas for vulnerabilities the databases identify the specific vulnerability. The vulnerability data comes from several well-known products, since the data for every operating system and server, both commercial and open-source, are available. The SRGM data comes only from some selected projects where the management has permitted disclosure of the data. On the other hand, vulnerability data has some limitations—namely, that it comes from a limited number of sources, and the number of vulnerabilities typically represents only a small fraction of the total number of defects.

In our analysis, we have used the data sets for four different operating systems [5][12] [61][51][56] [59] shown in Table 5-1. The vulnerability data of Windows 95 is for a client operating system that has existed for several years. It has gone through nearly a complete life cycle and its remaining installed base is now very small. The data set for Windows XP represents a relatively new operating system, which may be near the peak of its popularity. We have also included data for Linux Red Hat 6.2, which represents an open-source operating system. Some of the key attributes of the four systems are given in Table 5-1. The vulnerability density is the number of discovered vulnerabilities per thousand source lines of code (Ksloc). Windows XP is much larger than Windows 95; however, its vulnerability density is comparable. It is likely that a significant number of thus far undiscovered vulnerabilities are present in Windows XP. The higher number of vulnerabilities in Red Hat Linux 6.2 may be because a larger fraction of its code is devoted to access control. Fedora is a relatively new version of Linux.

Table 5-1. Some attributes of the examined operating systems

Operating Systems	**Millions of Lines of source code**	**OS Type**	**Known Vulnerabilities**	**Vulnerability Density (per Ksloc)**	**Release Date**
Windows 95	15	Commercial client	51	0.0034	Aug 1995
Windows XP	40	Commercial client	173	0.0043	Oct 2001
R H Linux 6.2	17	Open-source server	118	0.00694	Mar 2000
R H Fedora	76	Open-source server	154	0.00203	Nov 2003

Vulnerability data needs to be manually extracted from databases. Our major sources of data are the Mitre Corporation [56] website and the National Vulnerabilities Database

(NVD) [61] metabase. NVD is an easily searchable database with the option of downloading an ACCESS database.

5.3.2 Evaluation of goodness of fit

To evaluate goodness we will apply the chi-square goodness of fit test. The chi-square (χ^2) statistic is calculated as follows:

$$\chi^2 = \sum_{i=1}^{n} \frac{(o_i - e_i)^2}{e_i} \tag{22}$$

For fit to be acceptable, the chi-square statistic should be less than the critical value for a given alpha level and degrees of freedom. The P-value represents the probability that a value of the χ^2 statistic at least as high as the value calculated by the above formula could have occurred by chance. We use an alpha level of 5%; i.e., if the P-value of the chi-square test is below 0.05, then the fit will be rejected. A P-value closer to 1 indicates a better fit. The P-value is calculated by using the number of degrees of freedom of the data set and chi-square distribution.

5.3.3 Models Adequacy Criteria

For model adequacy testing, *Akaike Information Criteria (AIC)* [2] is also frequently used to make a fair comparison between models. AIC is formally defined as

$$AIC = -2 \log liklihood + 2M$$

An equivalent way to compute AIC is:

$$AIC = T.\ln(RSS) + 2M \tag{23}$$

where M is the number of free parameters of the examined model, T is the number of observations, and RSS is the residual sum of squares. In Equation (23), we use the formulation of AIC given by Akaike.

Other variation of AIC is AIC$_C$ is better than AIC when sample size, n, is small. The formula for AIC$_C$ is:

$$AICc = AIC + \frac{2k(k+1)}{n-k-1} \qquad (24)$$

5.4 FITTING THE DATA SETS TO THE MODELS

The results of fitting the models to the data is presented graphically in the plots given in Figures 6-9, which show the fitted plots together with actual cumulative data. For convenience in viewing, the plots for each operating system are shown in two separate figures given side by side. In addition, the parameter values obtained during the fit, and the corresponding measures of goodness of fit for the four operating systems' sets, are given in Tables 2-5. First, we discuss the results for each dataset; we then present observations for each of the models.

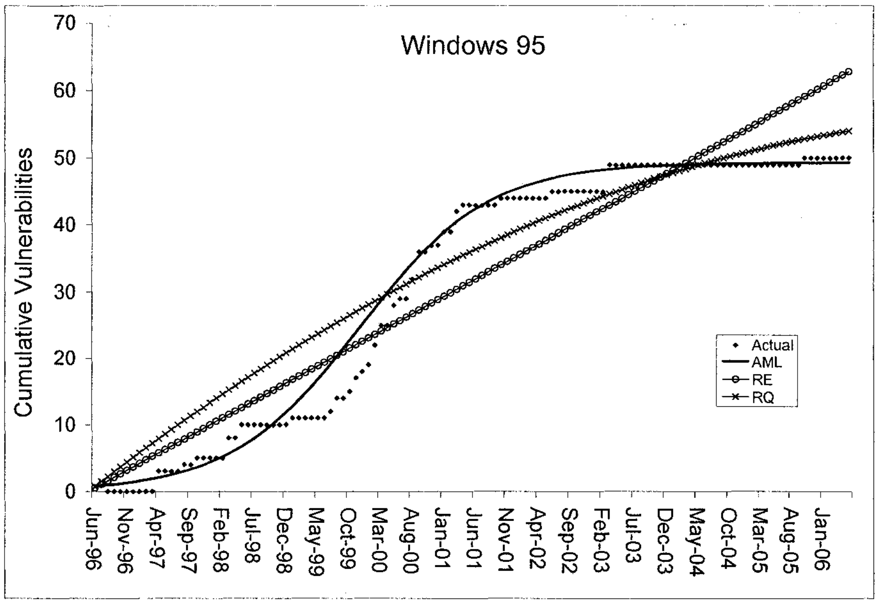

Figure 5-6. Fitted VDMs with actual Windows 95 data

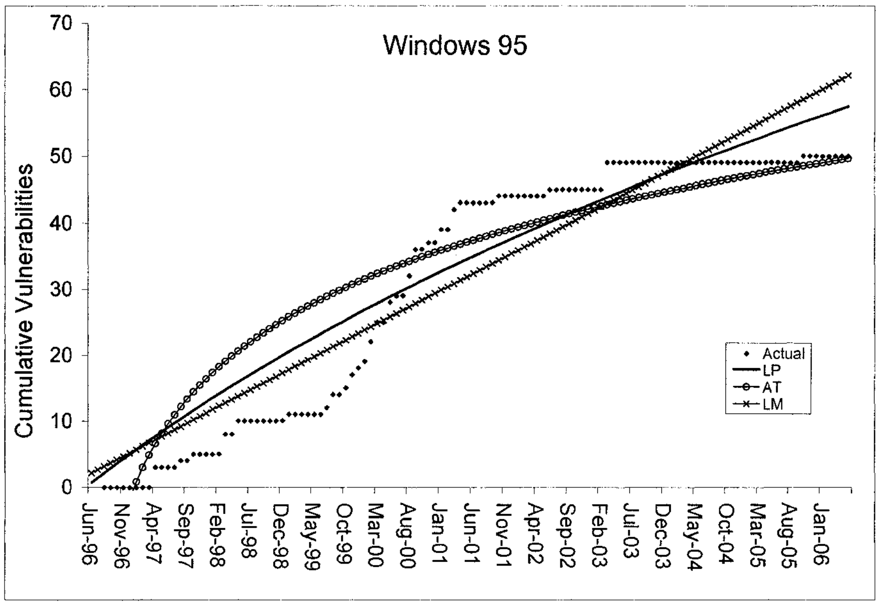

Figure 5-7.Fitted VDMs with actual Windows 95 data

Table 5-2. Goodness of Fit and AIC results for Windows 95 goodness of fit

Model	Parameters			χ^2–test		RSS	AIC
				χ^2	P-Value		
*AT	K/ γ	C		226.46	0	7992.3	1082.35
	18	2.36					
LP	β_0	β_1		178.18	.00036	3974.4	998.52
	69.00863716	1.08E-02					
*LM	u	v		202.44	.000003	5313.9	1033.37
	1.690	0.5032016					
RE	λ	N		181.47	.0002	5359.95	1034.41
	.000094	5629.743					
RQ	S	K		169.96	.0015	3429.55	980.82
	-0.00249	0.749276					
AML	A	B	C	46.93	1	422.9	731.65
	0.002	49.37396	1.358819937				

*Chi-square test was applied to the positive values of the AT and LM models

The Windows 95 data (Figure 5-6 and Figure 5-13) has a distinct s-shape because vulnerability detection reached saturation in 2003. As we would expect, the AML model fits quite well. The fitted RE, RQ, LP and LM models give plots that look linear and thus show considerable divergence at the end. The fitted AT model gives negative values at the beginning and significantly diverges from the actual data except near the end, which is why its chi-square value is significantly higher than the other models. The AML model has the lowest AIC, namely, 731.65, followed by RQ with 980.82, and then by other models ranging from 998.52 to 1082.35.

The Linux Red Hat 6.2 data (see Figure 5-8 and Figure 5-9) shows a milder s-shape. This allows two models, AML and RQ, to fit the data. The AML model shows a significant fit with P-value=1. The RQ model was able to fit the data with a P-value of 0.261 (see Table 5-3) because the data has just entered the saturation phase and thus, unlike other models, RQ was able to bend with the saturation. On the AIC scale, AML gave the best AIC test score, 558.82, followed by RQ, which gives 648.42, while the other models yielded higher numbers.

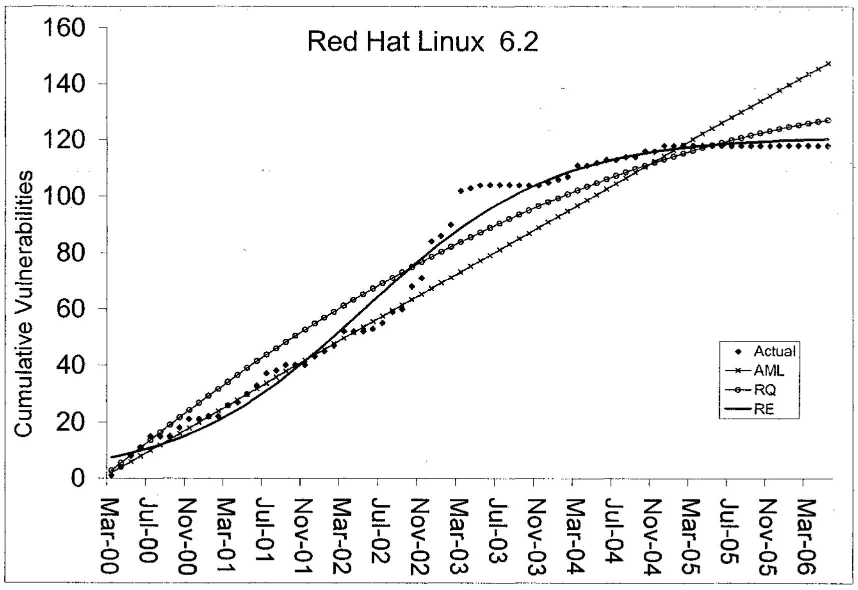

Figure 5-8. Fitted VDMs with actual Red Hat Linux 6.2 data

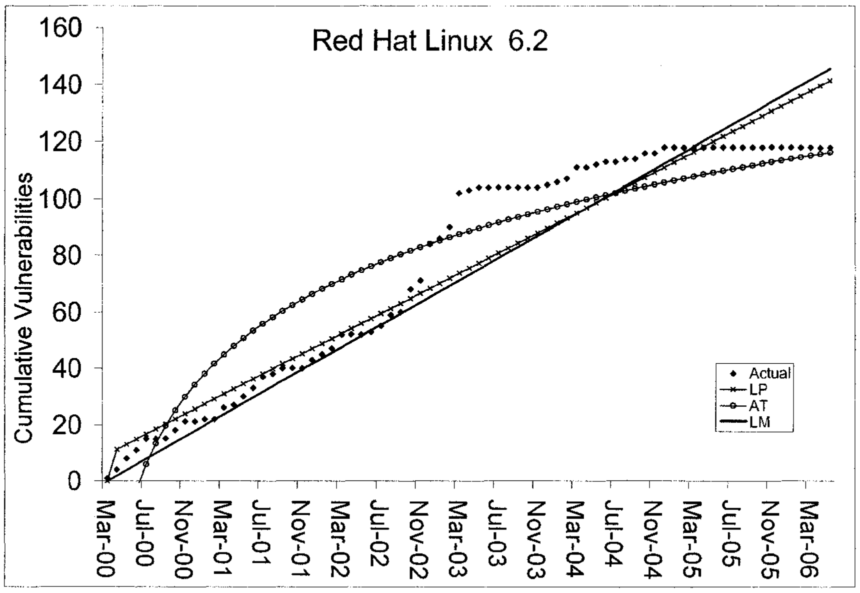

Figure 5-9. Fitted VDMs with actual Red Hat Linux 6.2 data

The data for Windows XP (shown in Figure 5-10 and Figure 5-11) shows a very linear trend, allowing RQ to fit quite well with the P-value of 0.971, leaving AML with a P-value of only 0.147 (see Table 5-4). This shows a less significant fit than RQ because the data for Windows XP does not yet have a strong S-shape. Windows XP is a relatively new operating system and the saturation phase had not been reached yet; consequently, the AML model does not yield the best fit. Other models were unable to provide a significant fit. On the AIC scale, RQ has shown a better AIC of only 492.10 compared to 511.47 of AML; other models have yielded higher AIC values ranging from 661.80 to 745.20. By this measure, also, the other models provided a poor fit.

In Red Hat Fedora (see Figure 5-12 and Figure 5-13), AML fit with 0.426 (see Table 5-3), while other models were unable to fit the data. This shows that Red Hat Fedora has begun to assume the s-shape but it is still in an early stage, allowing AML model to fit; furthermore, it can be expected as Red Hat Fedora data starts to mature further, the P-value will eventually go up. Additionally, AML has AIC of 347.68, while other models' AIC ranged from 388.08 to 449.90.

Table 5-3. Goodness of Fit and AIC results for Red Hat Linux 6.2

Model	Parameters			χ^2–test		RSS	AIC
				χ^2	P-Value		
*AT	K/γ	C		199.35	0	19469.39	744.75
	40.7745502	9.438709877					
*LM	u	v		144.087	0	10635.6	699.397
	9.591	1.7812802					
LP	β_0	β_1		131.02	0	12073.63	708.91
	6999.939	2.84E-04					
RE	λ	N		130.24	0	12009.071	708.51
	0.000354	5628.812					
RQ	S	K		81.35	0.261	5389.545	648.42
	-0.01504	2.823606					
AML	A	B	C	35.52	1	1589.1	558.82
	0.000855	121.235	0.139727427				

*Chi-square test was applied to the positive values of the AT and LM models

The Anderson Thermodynamic (AT) model did not fit any of the data; it exhibited the highest AIC scores and lowest P-values. For the Windows 95 data, its AIC of 1082.35 is 48 points away from the nearest model. For Linux 6.2, it yields an AIC of 744.75, which is 36 points higher than the nearest model. For Windows XP and Red Hat Fedora, AIC scores were also significantly higher than the nearest model.

The Rescorla Quadratic (RQ) model was able to fit two out of four of the datasets under consideration. However, the RQ was not able to fit either the Windows 95 or Red Hat Fedora datasets (see Figure 5-12 and Figure 5-13 below) due to the strong S-shaped trend of the data; although the model yielded an arc-shaped curve with the ability to show saturation at the end. Thus, for both datasets it scored poorly in the chi-square test with a P-value of close to zero, whereas it should have been at least 0.05 to be acceptable. For

Windows XP data, it achieved a P-value of 0.971, which is a very good fit. It fit the Linux 6.2 data with a P-value of 0.261.

The fit was judged to be poor for the chi-square test with P-values within 0 to 0.0002, which is significantly less than the acceptable P-value of 0.05. Moreover, RE scores were generally among the highest except if compared with AT.

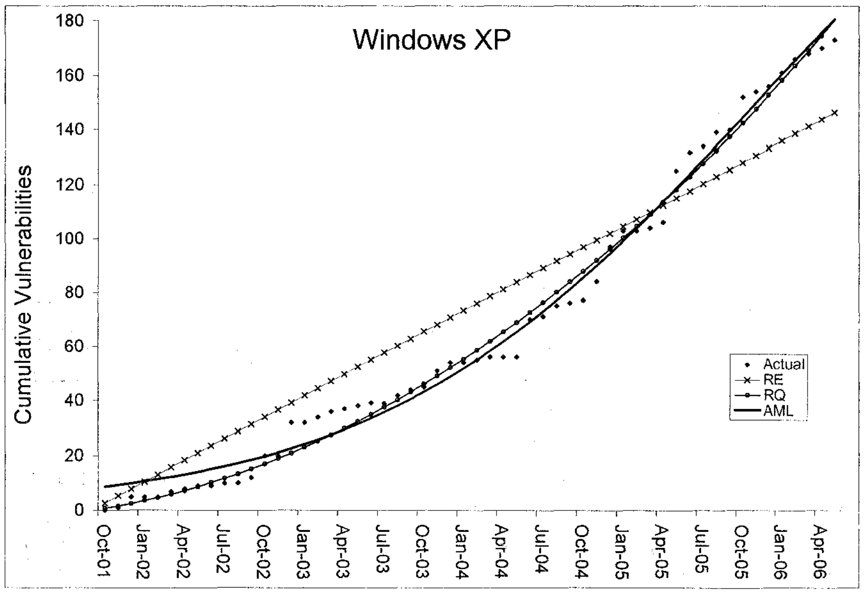

Figure 5-10. Fitted VDMs with actual Windows XP data

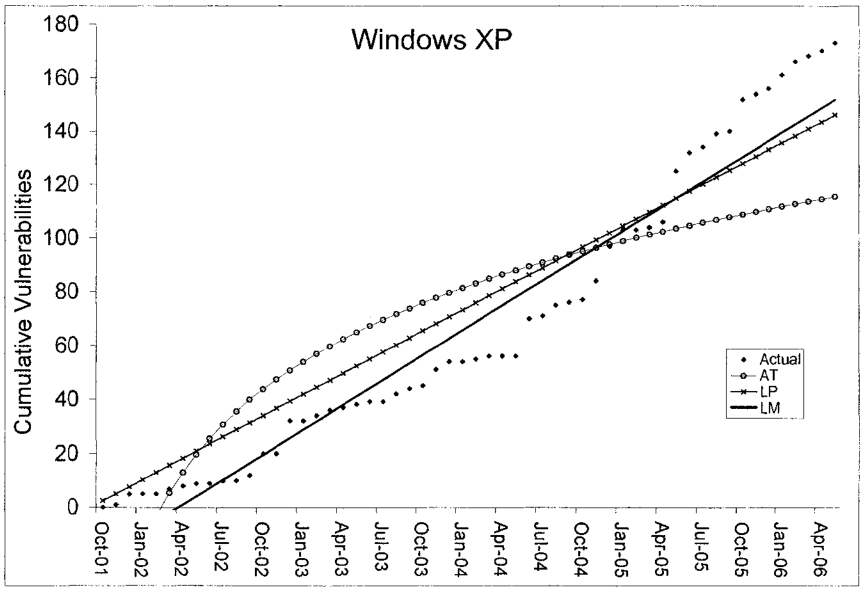

Figure 5-11. Fitted VDMs with actual Windows XP data

The Linear Model (LM) was unable to fit any of the datasets. However, its AIC score was better than RE and AT most of the time. Nonetheless, the Alhazmi-Malaiya Logistic Model (AML) was the only model that fit all four datasets. The fit was especially superior for the Windows 95 and Linux Red Hat 6.2 datasets. However, the fit was not as good for Red Hat Fedora and Windows XP where the P-values were 0.426 and 0.147, respectively. Moreover, the AML model showed significantly better AIC scores for all datasets, except for Windows XP where it was the second only to RQ.

The other four models failed the goodness of fit test for Windows 95, since they generated unacceptably high chi-square values and consequently low P-values below 0.02, although the Windows XP data is expected to saturate eventually. The results show that AML is the most consistent model for the four data sets.

Table 5-4. Goodness of Fit and AIC results for Windows XP

Model	**Parameters**			χ^2**–test**		**RSS**	**AIC**
				χ^2	**P-Value**		
***AT**	K/γ	C		524.59	0	54178.7	745.20
	49.251	9.186					
***LM**	u	v		127.18	0.006916	8928.76	622.67
	-20.412	3.0779474					
LP	β_0	B_1		243.04	0	16018.76	662.34
	11757.872	0.0002234					
RE	λ	N		241.42	0	15890.74	661.80
	6.9301E-05	37790.660					
RQ	S	K		36.86	0.971	1655.98	492.10
	0.044	0.734					
AML	A	B	C	65.99	0.147	1691.58	511.47
	0.0003	288.7025	0.1206				

*Chi-square test was applied to the positive values of the AT and LM models

The Logarithmic Poisson (LP) model was unable to fit all four examined datasets, with p-values ranging from 0 to .00036.

We note here that LP, RE and even RQ can perform better if only partial data is used before onset of saturation, when we expect the data to be somewhat linear, and hence models such as LP, RE, RQ and LM can easily fit the data [6].

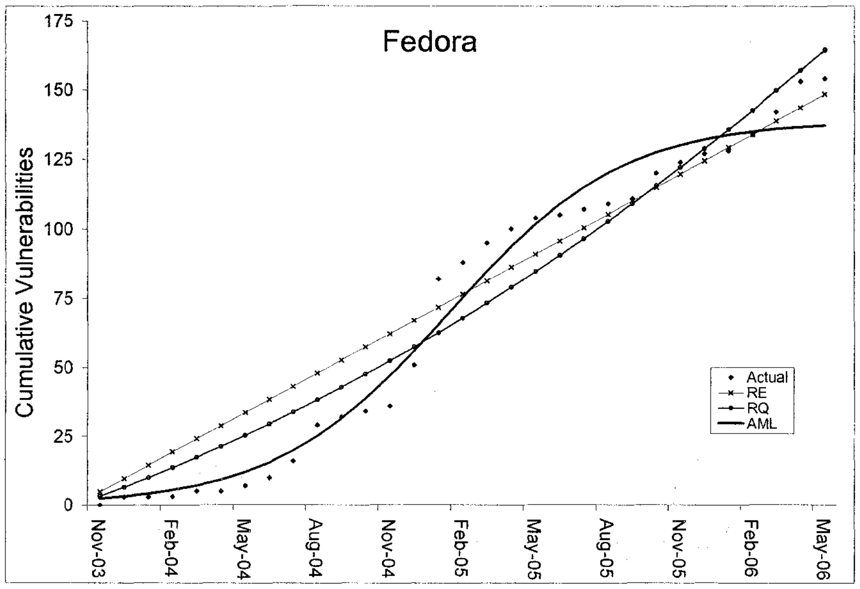

Figure 5-12. . Fitted VDMs with actual Fedora data

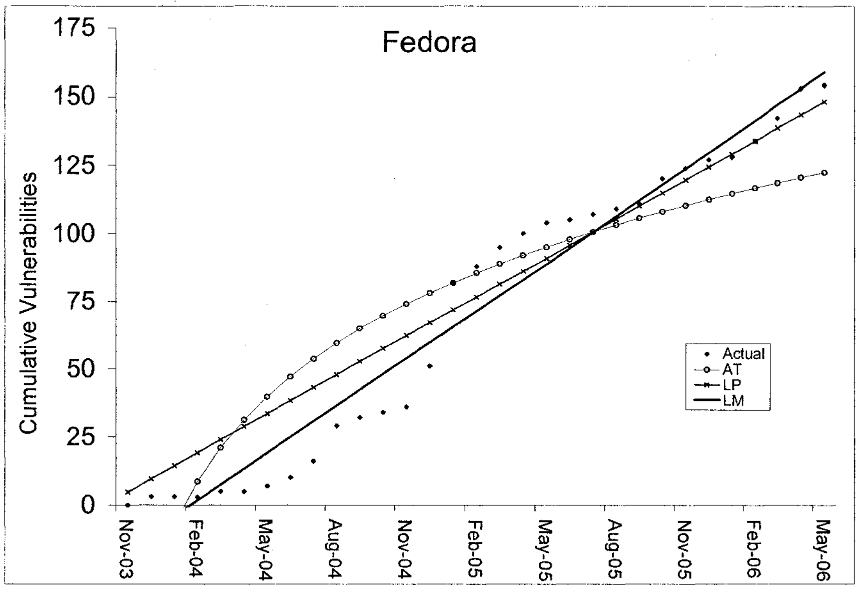

Figure 5-13. Fitted VDMs with actual Fedora data

Table 5-5. Goodness of Fit and AIC results for Red Hat Fedora

Model	Parameters			χ^2–test		RSS	AIC
				χ^2	P-Value		
*AT	K/γ		C	234.036	0	20108.948	449.90
	55.496		16.214				
*LM	u		v	66.423	0.00002	3592.179	372.39
	-21.477		5.8120967				
LP	β_0		β_1	178.986	0	7065.290	402.83
	9188.010		0.0005244				
RE	λ		N	178.051	0	7013.404	402.50
	0.000127		37790.59				
RQ	S		K	96.84	0	5090.857	388.08
	0.070		3.131				
AML	A	B	C	32.195	0.426	1984.343	347.69
	0.00201	139.045	0.535				

*Chi-square test was applied to the positive values of the AT and LM models

5.5 CONCLUSIONS

Several vulnerability discovery models were examined using Akaike Information Criteria (AIC) and chi-square (χ^2) tests. The evaluation found that the AML model is generally best for the longer term, performing better for systems such as Windows 95, Red Hat Linux 6.2, and Red Hat Fedora. Since it captures the S-shape pattern in the data, it has better fit as determined by using AIC and the chi-square test. RQ has shown very good fit for Windows XP, a system that has not yet shown signs of saturation. This can be attributed to the fact that the model can fit trends that are largely linear with slight saturation. LM, LP, and RE were not able to fit any of the datasets because of the lack of a semi-linear trend. The AT model considered did not perform well in general; it has shown significant diversion from the datasets. Three of the models, RQ, RE and LP, appear to do a good job of following the shorter-term trends when the cumulative vulnerabilities show a linear trend. The AML model often provides the best fit since it can follow the S-shaped trend that is often observed.

We also note that the AML model uses three parameters, while the other five models are two-parameter models. However, AIC takes the number of parameters into account and thus provides a fair basis for comparison.

Among the five models, the parameters of the RE and AML models have some simple interpretations. One of the parameters in both is related to the total number of vulnerabilities present in the software. If the expected range of vulnerability density values can be estimated based on past experience, a preliminary estimate of the total number of vulnerabilities may be empirically obtained [5][6] However, empirical estimation of other parameters requires further investigation.

Finding vulnerabilities in a widely installed system should be more rewarding to internal testers, as well as to external experts and hackers. Thus, the testing effort changes with changes in the market share of software systems. This variability of effort has been addressed in [4] by developing an equivalent effort model. The model addresses changes in the usage environment, which affects the discovery process by considering the cumulative time spent by the workstations using the software systems. Although the equivalent effort model fit very well, the model requires data that can be very hard to collect. The AML model attempts to implicitly model the effort variation.

Vulnerability discovery models assume that each specific release of an operating system is independent and can be separately modeled. In practice, a significant sharing of the code occurs between successive releases. Thus, a vulnerability detected in a particular version may also exist in previous versions. Further research is needed to model the impact of such shared codes.

In a recent work [7], the prediction capabilities of two of the VDM models were examined, and the results showed a good prediction capability that improves as more data becomes available. The prediction capability testing showed that putting some constraints on the model's parameters based on previous observations significantly improves the prediction capability. Metrics such as vulnerability density and vulnerability/defect ratios [4] can be used to check the projections made using VDMs during early phases when the available data is insufficient, or when a VDM is known to have some specific limitations. Examination of the prediction capabilities of the other models is still needed to give a broader comparison with the other vulnerability discovery models.

Both the developers and the user community can use vulnerability discovery models. Developers can assess the product readiness by projecting upcoming vulnerability discovery trends. Developers need to allocate security maintenance resources to detect vulnerabilities, preferably before others do, and to release security patches as soon as possible. The users also need to assess the risk due to vulnerabilities before patches are applied. A patch may need to be tested for stability before it is applied, as discussed by Brykczynski et al. [24] and Beattie et al. [18] An organization's effective security policy requires both time and resources, and vulnerability discovery models can be used to quantitatively guide such policies.

CHAPTER 6

ESTIMATING THE NUMBER OF VULNERABILITIES

6.1 INTRODUCTION

One way to evaluate the soundness of any model would be to see how well it fits the available data [5]. Vulnerability Discovery Models have been proposed and examined using vulnerability data and goodness of fit tests. For example, the logistic model (AML) has been shown to fit the data; however, the models' ability to predict the number of vulnerabilities has not yet been investigated. Nevertheless, a manager may often be interested in making projections when only part of the data is available. To evaluate prediction capability, we need to examine the accuracy of prediction using early partial data compared with later data.

In the software reliability engineering field, several researchers have investigated the predictive capability of a number of reliability growth models [47][54]. Generally, two of the measures of interest are average magnitude of error, and the average bias which measures the tendency of the model to overestimate or underestimate the number of vulnerabilities [47]. We will examine the prediction capability of vulnerability discovery models using a similar two-component predictability measure. The prediction capability of the VDMs will be examined by using actual data from major software systems and plotting the error in estimated values. This will allow us to compare different schemes, identify procedures that are likely to keep the error small, and assess the accuracy of projections [8][9].

The approach will consider applying the model under consideration for *n* number of times ($t_1, t_2, t_3 \ldots t_n$) with equal calendar time intervals between estimations. For each t_i, the partial data at t_i is fitted to the model at the best fit, using regression analysis to determine the best values for the parameters. At each test t_i, each model's parameters are used to plot the complete prediction curve to obtain the estimated number of vulnerabilities (Ω_i). These estimated numbers of vulnerabilities ($\Omega_1, \Omega_2, \Omega_3 \ldots \Omega_n$) are compared to the in-hand actual number of vulnerabilities (Ω) to determine the normalized estimation error $(\Omega_i-\Omega)/\Omega$. Then the average of the normalized error magnitude values is computed to obtain the measure of average error (AE). Average bias (AB) is similarly obtained when the sign of the error is also considered [47]. The average error (*AE*) and average bias (*AB*) are defined as:

$$AE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{\Omega_i - \Omega}{\Omega}\right|, \qquad (25)$$

$$AB = \frac{1}{n}\sum_{i=1}^{n}\frac{\Omega_i - \Omega}{\Omega}, \qquad (26)$$

where Ω is the actual number of vulnerabilities, while Ω_i is the number of vulnerabilities predicted at time t_i.

An alternative approach in evaluating the next step prediction is that the data up to t_i is used to estimate the value at t_{i+1} [1][22]. However, in cases where models are needed to make longer term projections, predictive capability for a variable period are also needed, and AE and AB are more appropriate measures.

Next, the estimation approaches will be previewed and the predictive capability of these approaches will be evaluated for some operating systems and HTTP web servers. The models' ability to predict will be examined to identify their estimation accuracy. The approach used for testing prediction capability is to have a complete vulnerability dataset. A subset of the data (i.e., the first 12 months) is used, and then the subset of data is fitted to the model with the best fit, and the rest of the curve will be plotted to make an estimation. The dataset is then extended gradually and the results are recorded for comparison with the real result. Thus, the estimations' deviation from the real result is measured.

We also gather data related to other attributes of the software systems (size, number of defects, etc.) to suggest some metrics such as the vulnerability density metric. We then compare vulnerability density to defect density to determine a relationship.

6.2 EVALUATING THE PREDICTION CAPABILITY OF THE VULNERABILITY DISCOVERY MODELS

We examine three approaches for estimating the number of vulnerabilities. These approaches are the static approach, which uses the program size as the static metric. The dynamic approach uses the vulnerability discovery models (VDMs). The third approach combines the dynamic capabilities with the static estimation to filter out the extreme estimations.

6.2.1 THE STATIC APPROACH

This approach requires the knowledge of the vulnerability density of comparable software systems. Vulnerability density [5] is defined as the number of vulnerabilities per thousand lines of code. The knowledge of vulnerability density of at least one previous software system is a requirement of the static approach. Such previous software systems should preferably have been developed in a comparable manner—i.e., developed by a similar team in the same organization. Then, assuming that the vulnerability density is close to the older system, we will have an error range that can be predetermined for addressing factors such as better development practices, more thorough testing before release, fraction of code that is access-related, etc.

The static approach has a major advantage, which is that it is usable on the date of release, while modeling requires some partial data to be available. Furthermore, the static estimation can be used to improve the estimation of the model's parameters (e.g. parameter B in the Alhazmi-Malaiya model (AML)) [8][9].

6.2.2 The Dynamic Approach

We have used two different implementations of the AML model in addition to the LM model.

- Unconstrained AML: This approach does not place any prior constraints on the parameters; rather, the AML model given by Equation (4) is directly applied. The three parameters are estimated using iterative computation utilizing a least squares fit.
- AML constrained (AML-C): We chose to apply the constraint that the duration between the two transition points is obliged to be within a certain limit. The duration has been shown to be *2.63/AB* for the AML model. Hence, we can limit 2.63/AB between some certain minimum and maximum values. The choice of the minimum and maximum values is determined by the values from the previous software systems. This constraint assumes that the transition points are within the time-frame of the expected lifetime of the software, thus enforcing the S-shape and anticipating the two transition points [8][9].
- Linear (LM): The linear LM model is given by Equation (10). The parameters values are determined simply by linear regression analysis.

6.2.3 The Combined Dynamic/Static Approach

The dynamic vulnerability discovery estimation approaches, (AML), (AML-C) and (LM), are applied to estimate the total number of vulnerabilities. However, when the dynamic model is applied and some extreme estimation occurs due to certain bumps that appear early in the vulnerability datasets, we can also use the static approach to estimate ceiling and floor values. This would require the use of vulnerability density data from previous similar systems. In AML and AML-C, the static estimation was used to obtain the bounds of the parameter B [8][9].

6.3 PREDICTION CAPABILITIES FOR ESTIMATING VULNERABILITIES IN OPERATING SYSTEMS

We applied the estimation approaches mentioned above to estimate the number of vulnerabilities in three systems: Windows 98, Windows 2000, and Linux 7.1. Because the use of static methods and constraints requires the use of prior data, operating systems such as Windows 95 and Windows NT 4.0 could not be used due to lack of data from previous applicable versions.

Table 6-1 illustrates the use of prior data for the three systems. The vulnerability data was obtained in April 2005 [5][56][61]. The static approach is able to give estimates only for the final point as a whole figure, but not estimates for different points of time. From Table 6-1, we note that the vulnerability densities for Windows 98 and Linux 7.1 do lie within ±25% of the value of their respective predecessors. However, the vulnerability density for Windows 2000 is outside ±25% of the value of its predecessor, Windows NT 4.0. This may be due to the fact that Windows 2000 contains a larger fraction of non-access related code compared to NT 4.0.

Figure 6-1 to Figure 6-3 show the normalized error in estimations at different times for the three systems. The time scale is given as a percentage of the total time period considered. As expected, projections made using AML or LM closer to the end time exhibit less error. The shaded region in these figures shows where ceiling and floor values would be if *Ω* was estimated using the static approach, assuming the defect density to be within ±25% of the predecessor.

Table 6-1: Estimating the number of vulnerabilities using the static approach

Software Systems	**Comparable V_D**	**Code Size (Msloc)**	**Est. Ω (Vulns.)**	**With ±25%**	**Actual Ω (Vulns.)**
Windows 98*	0.0033 (Win95)	18	60	45 to 75	66
Windows 2000*	0.0112 (Win NT 4.0)	35	392	294 to 490	172
R H Linux 7.1*	0.00694 (R H L 6.2)	30	208	156 to 260	164

*Data as of April 2005

The plot for AML error for Red Had Linux 7.1 (Figure 6-3) demonstrates that the fitted model first shows underestimation up to approximately 45% of the time, after which it shows overestimation. In about 65% of the time, the projection stabilizes and is reasonably accurate. Use of constraints during the parameter value search can cut out such swings. For Windows 98 (Figure 6-1), AML generally underestimates while AML-C overestimates. For Windows 2000, both AML and AML-C underestimate. In all cases, the error reduces significantly as we approach the latter part of the system's lifetime.

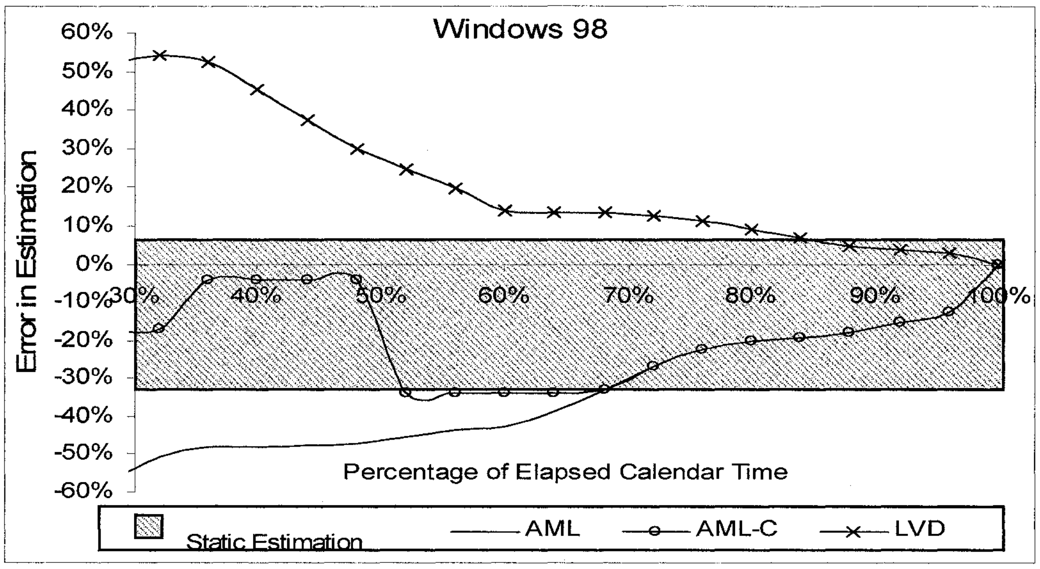

Figure 6-1. Estimation error values for Windows 98

The first constraint on B assumed a 25% range (Table 6-1). This constraint had to be dropped for Windows 2000. The second constraint on Windows 98 and Windows 2000 was chosen to be $24 \leq (2.63/AB) \leq 48$. However, for Red Hat Linux 7.1, the constraint was $18 \leq (2.63/AB) \leq 36$, since Linux versions tend to have a shorter lifetime than Windows. The figures show improvements in the accuracy of estimations for the AML model. The improvement over the earlier estimations is quite significant, as the impact of extreme projections is minimized by applying the two constraints.

Table 6-2 shows the average error (AE) and the average bias (AB) given by Equations (25) and (26), calculated for the three datasets. We note significant improvement in both AE and AB when the constraint is used in AML.

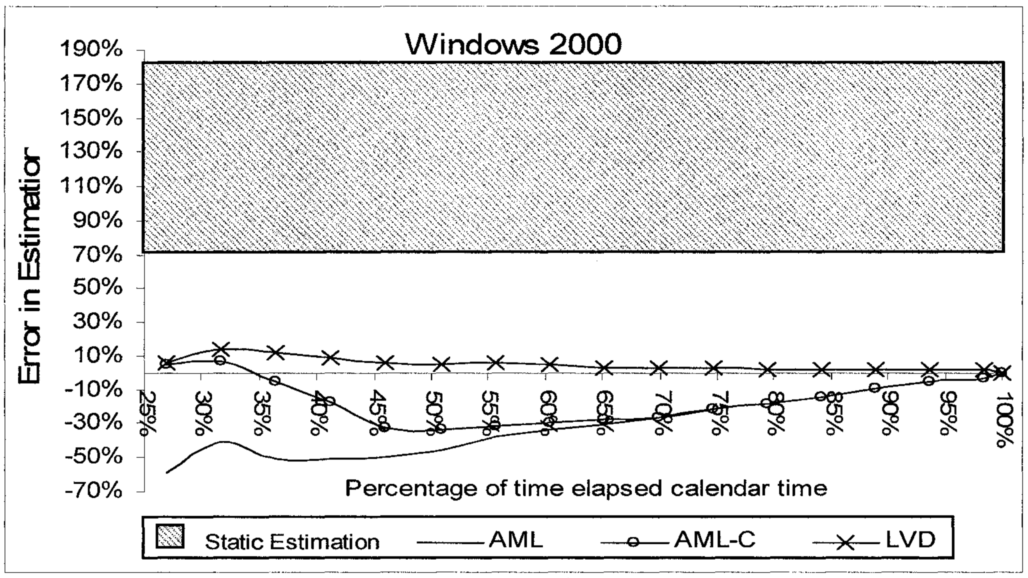

Figure 6-2. Estimation error values for Windows 2000

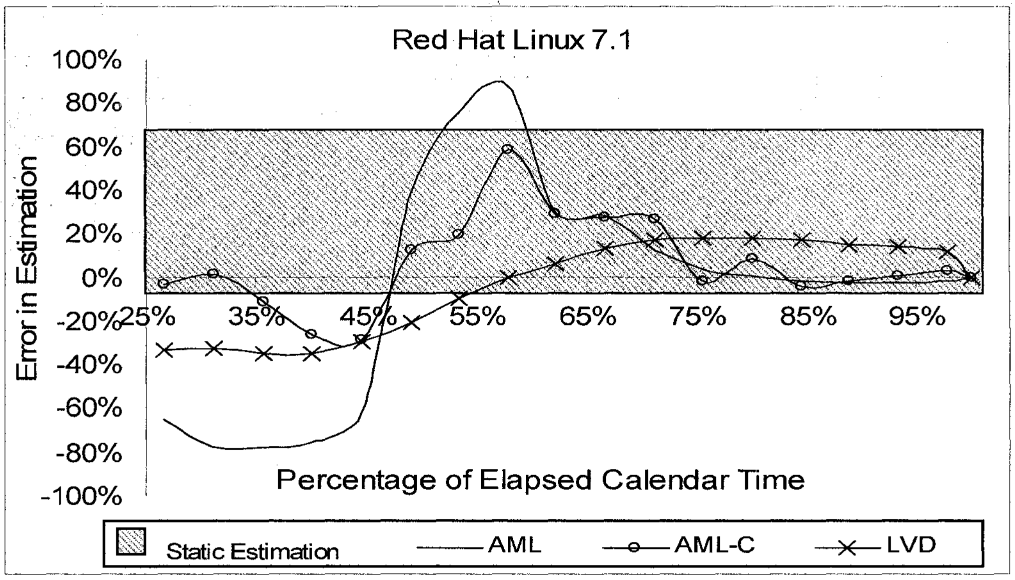

Figure 6-3. Estimation error values for Red Hat Linux 7.1

Table 6-2: The Average Error (AE) and Average Bias (AB) for estimations using the dynamic approach

Approach / Operating System	AML		AML with constraints		Linear (LM)	
	AE	AB	AE	AB	AE	AB
Windows 98	43.8%	-43.8%	20.1%	-20.1%	29.1%	29.1%
Windows 2000	29.2%	-29.8%	16.8%	-15.5%	4.6%	4.6%
Red Hat Linux 7.1	37.9%	-5.3%	15.4%	6.3%	19.2%	-4.0%

It is interesting to note that LM actually does a good job for Windows 2000 and RH Linux 7.1. This is because a noticeable saturation in vulnerability discovery for these two systems has not yet occurred.

From Figure 6-1 to Figure 6-3, we can observe that a combination of static and dynamic approaches can improve prediction capability. This gives the estimator more flexibility in estimating the number of vulnerabilities in a system; moreover, if more than one approach is used, the estimator can have more confidence in the estimation results [8].

6.4 PREDICTION CAPABILITIES FOR ESTIMATING VULNERABILITIES IN WEB SERVERS

The security of systems connected to the Internet depends on several components of the system. These include the operating systems, the HTTP servers, and the browsers. Some of the major security compromises arise because of vulnerabilities in the HTTP servers. The databases for the vulnerabilities are maintained by organizations such as the National Vulnerabilities Database [61], MITRE [56], and Bugzilla [3] BugTraq [75] as well as by the developers of the software.

The exploitations of some of the server vulnerabilities are well known. The Code Red worm [52], which exploited a vulnerability in IIS (described in Microsoft Security Bulletin MS01-033, June 18, 2001), appeared on July 13, 2001, and soon spread world-wide in unpatched systems.

There have been many studies attempting to identify causes of vulnerabilities and potential counter-measures; nonetheless, the development of systematic quantitative methods to characterize security has begun only recently. There has been considerable debate comparing the security attributes of open source and commercial software [14]. However, for a careful interpretation of the data, rigorous quantitative modeling methods are needed. The likelihood of a system being compromised depends on the probability that a newly discovered vulnerability will be exploited. Thus, the risk is better represented by the not yet discovered vulnerabilities and the vulnerability discovery rate rather than by the vulnerabilities that have been discovered in the past and remedied by patches.

Possible approaches for a quantitative perspective of exploitation trends are discussed in [23][32]. Probabilistic examinations of intrusions have been presented by several researchers [20]. In [69], Rescorla has studied vulnerabilities in open source servers. The vulnerability discovery process in operating systems has just recently been examined by Rescorla [69] and by Alhazmi and Malaiya [4][5][6].

Servers are very attractive targets for malicious attacks because they represent the first line of defense that, if bypassed, can compromise the integrity, confidentiality and availability attributes of the enterprise security. Thus, it is essential to understand the threat posed by both undiscovered vulnerabilities and recently discovered vulnerabilities for which a patch has not been developed or applied. In this chapter we address questions such as: How can we predict the vulnerabilities not yet discovered? How accurate are these estimations? We also consider methods for enhancement of the prediction capabilities.

At this time, despite the significance of security in the HTTP servers, very little quantitative work has been done to model the vulnerability discovery process for the servers. Such work would permit the developers and the users to better estimate future vulnerability discovery rates. Use of reliability growth models is now common in software reliability engineering [44][55]. SRGMs take into account the fact that as bugs are found and removed, fewer bugs remain. Therefore, the bug finding rate gradually drops and the cumulative number of bugs eventually approaches saturation. Such growth models are used to determine when a software system is ready to be released and what future failure rates can be expected.

Some vulnerability discovery models were recently proposed by Anderson [14], Rescorla [69] , and Alhazmi and Malaiya [4]. The applicability of these models to several operating systems was examined in [4][7][8]. The results show that while some of the models fit the data for most operating systems, others either do not fit well or provide a good fit only during a specific phase.

In the following sections we will evaluate the prediction capabilities for long term projection (i.e. the total number of vulnerabilities in a software system) and short term projection (i.e. vulnerabilities expected to be discovered within the next year) using those datasets and examine some enhancements that can improve the accuracy of predictions. Lastly, we discuss some of the major observations.

6.4.1 Long Term Prediction

The analysis in the previous section shows that the AML model fits all datasets while the LM model has fitted only some of them. However, we should really judge the models by examining the accuracy of future projections about the vulnerabilities using the data available at hand. A model with good predictive capabilities can be used to estimate the resources needed for maintenance and the risk associated with a particular web server [9].

We examine the accuracy of the predictions made using three approaches, AML, AML-C, and LM. AML-C is the AML model with the constraint (21<2.63/AB<42) used during numerical parameter estimation. The constraint restricts the linear phase to be within 21 to 42 months, which enforces the S-shaped characteristic of the AML model. The constraint was able to help in avoiding extreme parameter values during estimation using AML, especially when the available dataset was still small [7]. The constraint is generally automatically satisfied when the dataset gets larger.

Next, the plots of normalized error values are given against normalized time in Figure 6-4 to Figure 6-11. The x-axis gives time as the percentage of the overall time, and the y-axis give the normalized error.

6.4.2 LONG TERM PREDICTION ACCURACY TEST

The normalized error values for the estimates of the total number of vulnerabilities for Apache 1 are shown in Figure 6-4. AML-C shows significant improvement over AML by avoiding extreme values during the early part. LM has shown stable performance.

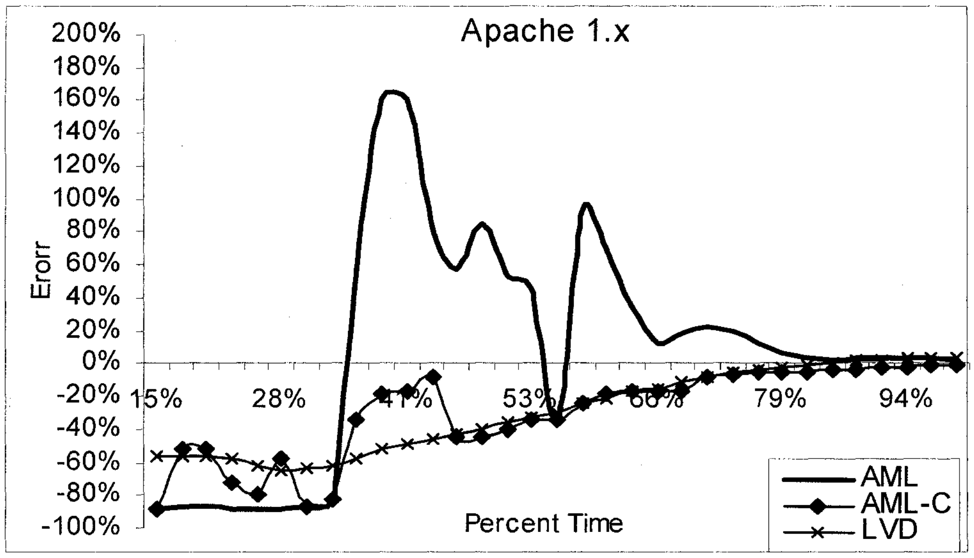

Figure 6-4. Accuracy of the models in estimating Apache 1.x vulnerabilities data

The error values for Apache 2 illustrated by Figure 6-5 shows that LM has performed well, followed by AML-C, then AML. Table 6-3 summarizes the Average error and Average bias of the estimations.

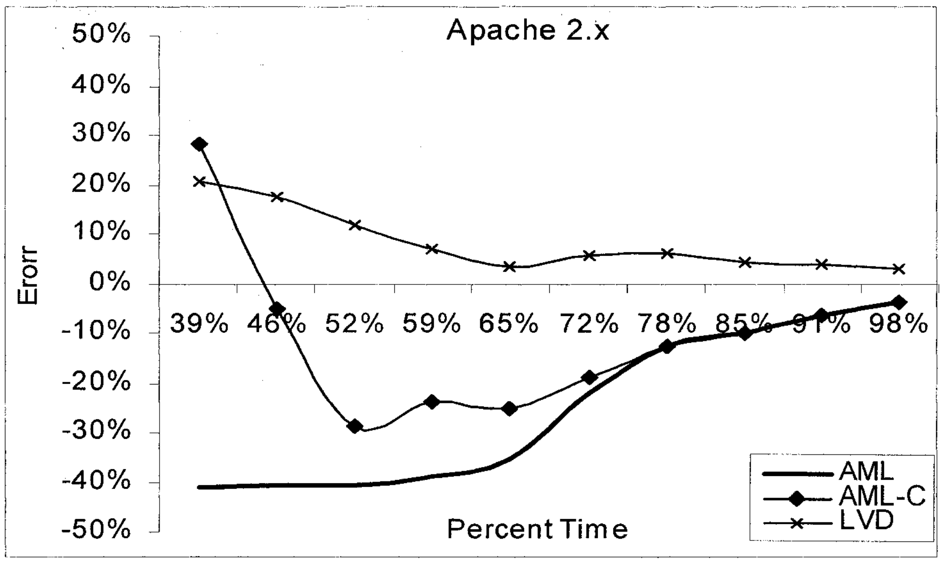

Figure 6-5. Accuracy of the models to estimate Apache 2.x vulnerabilities data

IIS 4 and 5 estimation errors, illustrated by Figure 6-6 and Figure 6-7 respectively, show that AML-C has the most accurate estimations, followed by AML. LM was generally shown to have the higher average error in most cases (see Table 6-3). This suggests that the LM model was unable to model the saturation phase for the vulnerabilities in IIS datasets.

Table 6-4 below shows the average error and average bias values calculated when at least 65% of the data was available for making the projections. After about 65% of the time, the AE value drops to single digits in many cases. This table demonstrates that LM projections still result in high error values for the two IIS versions. The AML-C results are generally the best with AE ranging from 2.1 to 10.4%, whereas LM can result in AE values of 41% for the two IIS versions.

The AB values in the tables suggest that the projections are biased. In the next subsection, an adaptive estimation technique which attempts to reduce the bias is discussed and evaluated.

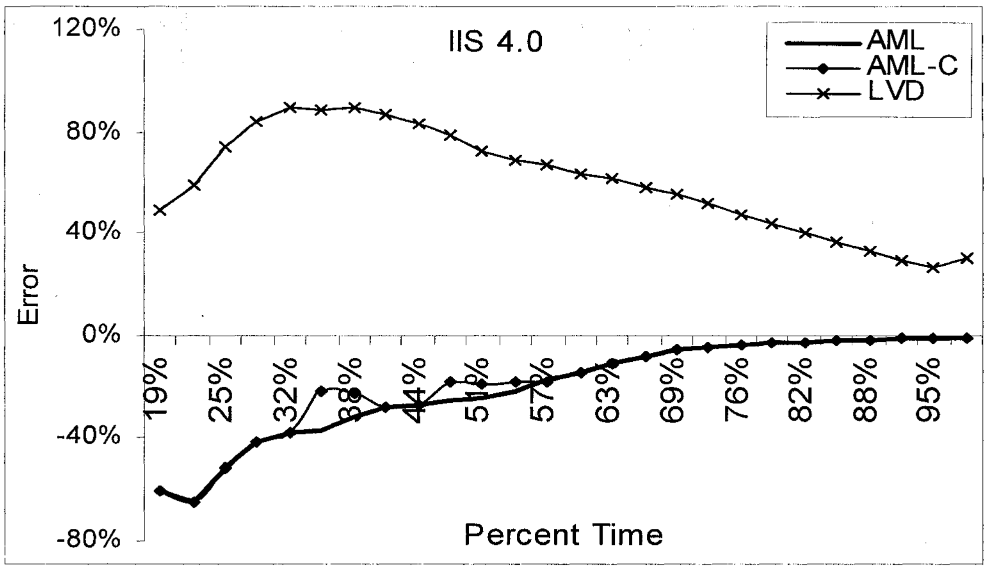

Figure 6-6. Accuracy of the models for estimating IIS 4.0 vulnerability data

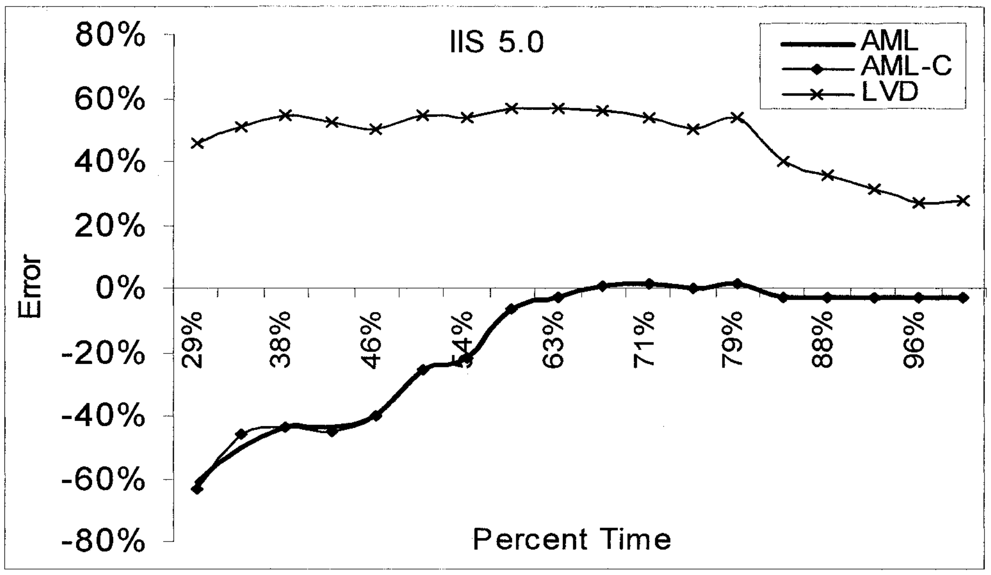

Figure 6-7. Accuracy of the models for estimating IIS 5.0 vulnerability data

Table 6-3. Average Error and Average Bias for Prediction of Ω

Model / Server	AML		AML-C		LM	
	AE	AB	AE	AB	AE	AB
Apache 1.0	52.5%	8.3%	29.8%	-29.8%	29.9%	-29.1%
Apache 2.0	25.2%	-25.2%	16.3%	-10.7%	8.2%	8.2%
IIS 4.0	20.3%	-20.3%	18.9%	-18.9%	60%	60%
IIS 5.0	17.4%	-17.1%	17.4%	-17%	47%	47%

Table 6-4. Average Error and Average Bias for Prediction After 65% of Time Elapsed

Model / Server	AML		AML-C		LM	
	AE	AB	AE	AB	AE	AB
Apache 1.0	8.4%	8.4%	6.2%	-6.2%	4.7%	-2.4%
Apache 2.0	11%	-11%	10.4%	-10.4%	4.4%	4.4%
IIS 4.0	3.2%	-3.2%	3.2%	-3.2%	41.2%	41.2%
IIS 5.0	2.1%	-1.4%	2.1%	-1.4%	41.8%	41.8%

6.4.3 Adaptive long-term prediction accuracy test

Adaptive techniques or recalibration have been applied to software reliability growth models [40] in order to improve prediction. The approaches adjust the predictions by observing the errors of past estimations. This adaptive technique also referred to as recalibrating, was found to improve the accuracy of the estimations of software reliability growth models.

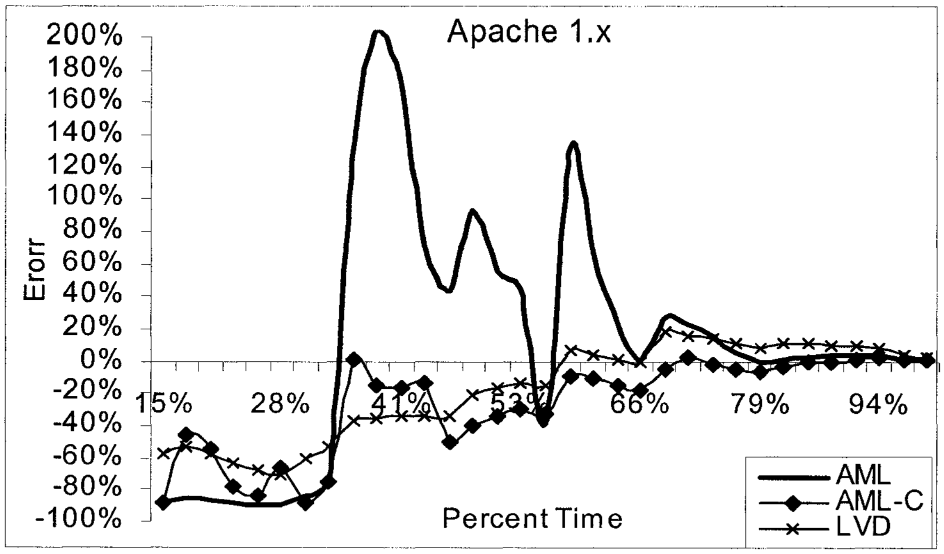

Figure 6-8. Accuracy of the adaptive models in estimating Apache 1.x vulnerabilities data

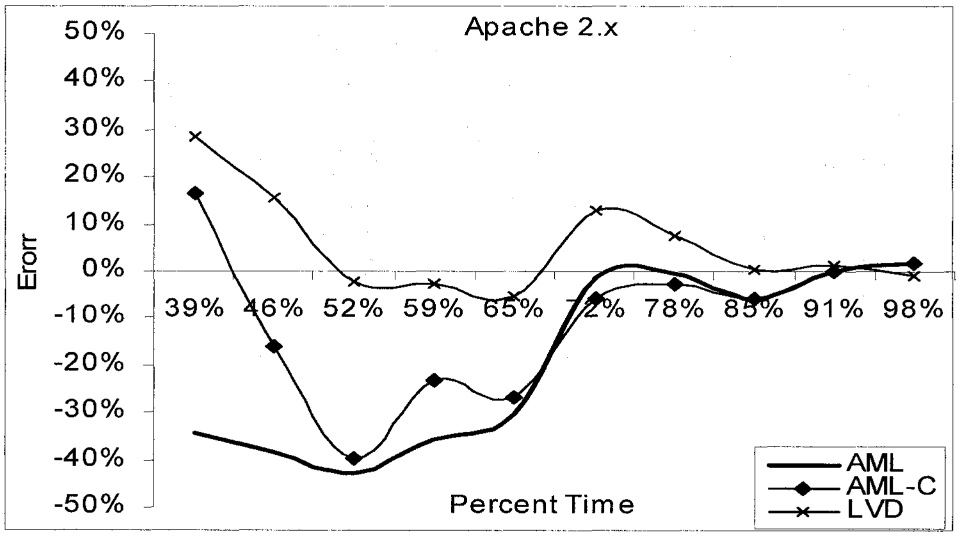

Figure 6-9. Accuracy of the adaptive models in estimating Apache 2.x vulnerabilities data

With adaptive techniques the differences among the models tend to get smaller. One main consideration in the adaptive technique is finding the optimal size of prediction error to eliminate by subtraction or division. There is a trade-off in choosing a larger or a smaller error value used for correction. Larger error corrections can give better results when there is a consistent bias in estimation; however, the model often loses its characteristics. Whereas, choosing smaller error can preserve the overall model characteristic and the overall improvement would be smaller.

Figure 6-8 to Figure 6-11 show errors in estimations using adaptive techniques, they demonstrate significant improvement over the original estimations shown in Figure 6-4 to Figure 6-7.

The recalibration was done by calculating the ratio between observed and estimated values for the past three quarters, and then dividing the next estimate by that ratio to adjust it. Recalibration can be done by using several alternative approaches; the effectiveness of recalibration can vary depending on the size of adjustments used.

The resulting plots shown in Figure 6-8 to Figure 6-11 and summarized in Table 6-5 and Table 6-6 demonstrate some modest improvements over the non-adaptive results.

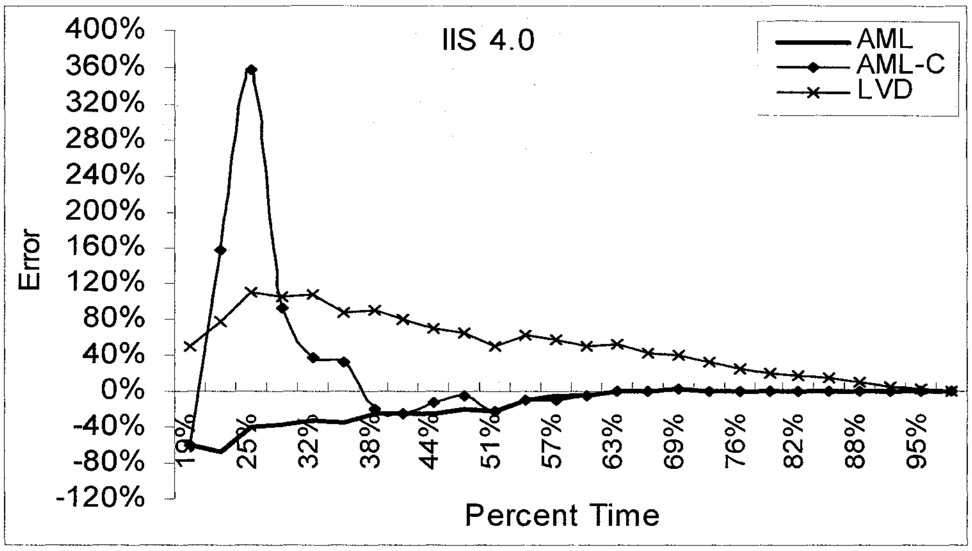

Figure 6-10. Accuracy of the adaptive models in estimating IIS 4.0 vulnerability data

Table 6-5. Average Error and Average Bias for Prediction of Ω (Adaptive Estimation)

Model / Server	AML		AML-C		LM	
	AE	AB	AE	AB	AE	AB
Apache 1.0	56.1%	12.1%	27.2%	-12%	26.1%	-17.7%
Apache 2.0	19.1%	-26.1%	15.3%	-10%	10.5%	5.6%
IIS 4.0	16.1%	-15.6%	32.9%	-20%	50.8%	50.8%
IIS 5.0	16.9%	-14.7%	21.1%	0.1%	40.9%	40.9%

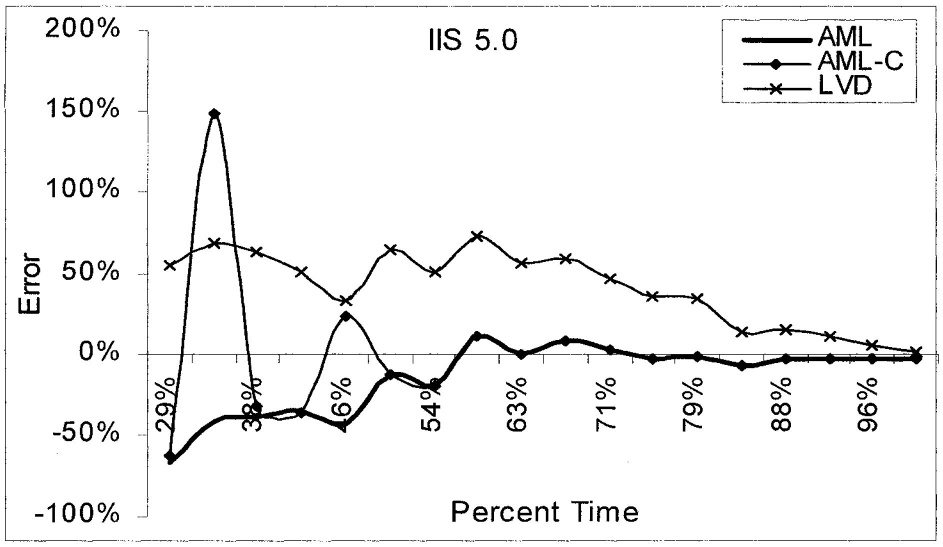

Figure 6-11. Accuracy of the adaptive models in estimating IIS 5.0 vulnerability data

Table 6-6. Average Error and Average Bias for Prediction of Ω after use of 65% of the Data Is Used (Adaptive Estimation)

Model / Server	AML		AML-C		LM	
	AE	**AB**	**AE**	**AB**	**AE**	**AB**
Apache 1.0	6.8%	6.7%	3.7%	-2.4%	9.7%	9.7%
Apache 2.0	1.9%	-1.2%	8.2%	-2.5%	5.5%	4.3%
IIS 4.0	0.6%	0.3%	0.6%	0.3%	19%	19%
IIS 5.0	3.9%	-1.9%	3.9%	-1.9%	24.4%	24.4%

Table 6-6 above gives the error values when the projections are made after only 65% of the data has become available. The errors in general are small for AML and AML-C, ranging from 0.6% to 8.2% and from 5.5% to 24.4% for the LM model.

6.4.4 Estimating Vulnerabilities for the Following Year

In the previous section, we have seen that the AML models and the LM Model can be used to predict the total number of vulnerabilities with good accuracy, especially when suitable methods are used to enhance the predictive capability. However, many applications, such as risk assessment applications, require the estimation of the vulnerability discovery rate for a shorter period of time in the near future. It is therefore essential to test the accuracy of the shorter term estimations so a practitioner can predict the number of vulnerabilities expected, say during the next year, and be aware of any limitations of the estimates made.

Similar to the methodology used for the long term projections, here the projection has been made at each of the n instances ($t_1, t_2, t_3 \ldots t_n$) with three months intervals. For each t_i, the partial data up to t_i has been used to fit the model using regression analysis to determine the best values for the parameters. The parameters are used to project the number of vulnerabilities θ_i' expected to be discovered within the next year and then compared with the actual number of vulnerabilities (θ_i) to determine the estimation error (θ_i-θ_i'). We then take the average of the error values to obtain the measure of the absolute average error (*AAE*) and the absolute average bias (*AAB*) which are defined as follows:

$$AAE = \frac{1}{n}\sum_{i=1}^{n}\left|\theta_i - \theta_i'\right|, \tag{27}$$

$$AAB = \frac{1}{n}\sum_{i=1}^{n}\theta_i - \theta_i', \tag{28}$$

Note that it is not possible to normalize the AAE and AAB measures because in the later periods, the actual value can sometimes be zero, causing the division by zero error if normalization is used.

Next, the estimation approaches will be previewed and the predictive capabilities of these approaches will be evaluated. In this section we suggest an alteration to the direct application of the models in order to improve the accuracy of prediction. Estimations can be done using direct model estimation, or using an adaptive approach. The remainder of this section discusses details and measures accuracy.

6.4.5 Estimating the following year's vulnerabilities by direct model application

In this approach, the prediction is done using a subset of the data to fit the model using the least square method, and then the actual number of vulnerabilities of the following year is compared to the predicted number of vulnerabilities.

The results are shown in Figure 6-12 to Figure 6-15 and the average values are given in Table 6-7. AML and AML-C have been shown to have a higher accuracy than LM in IIS, while LM has shown better accuracy for Apache 2, possibly because the Apache 2 dataset was very linear. For IIS 5, LM accuracy is close to AML and AML-C. All of the models performed close to each other in terms of average error, although it is observed that AML and AML-C did slightly better for the IIS data sets and LM did slightly better for the Apache data sets. This difference in applicability of models can be explained by the fact that AML and AML-C model the saturation effect whereas LM may fits data well when saturation has not occurred.

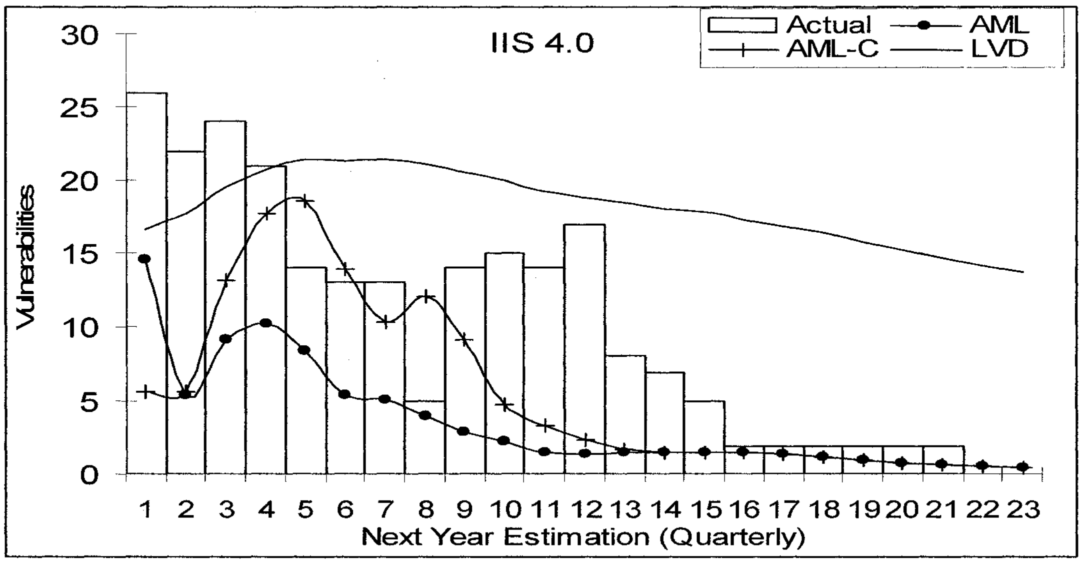

Figure 6-12. Estimations of vulnerabilities for the following year (quarterly)

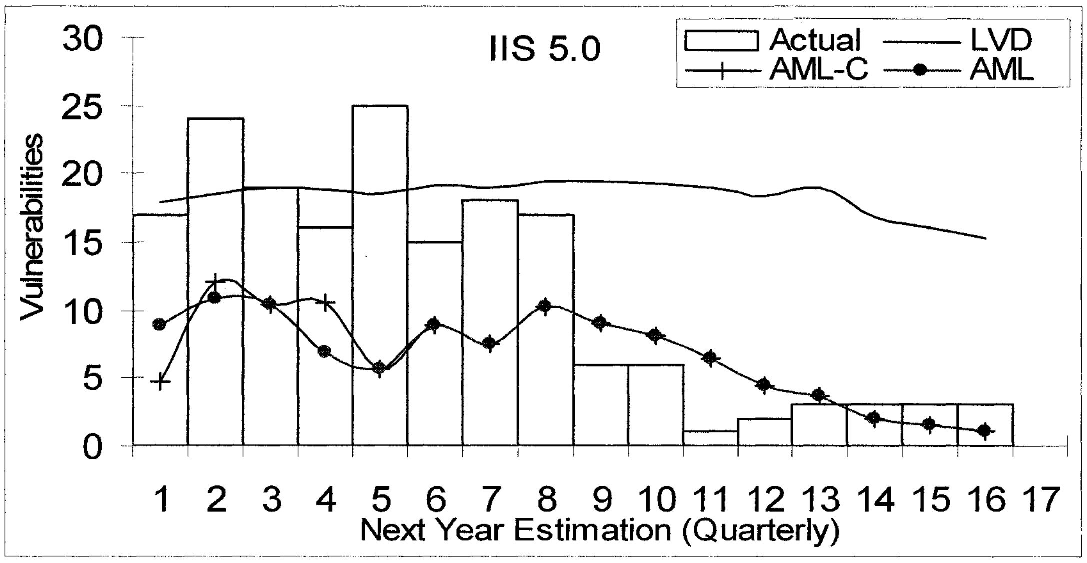

Figure 6-13. Estimations of vulnerabilities for the following year (quarterly)

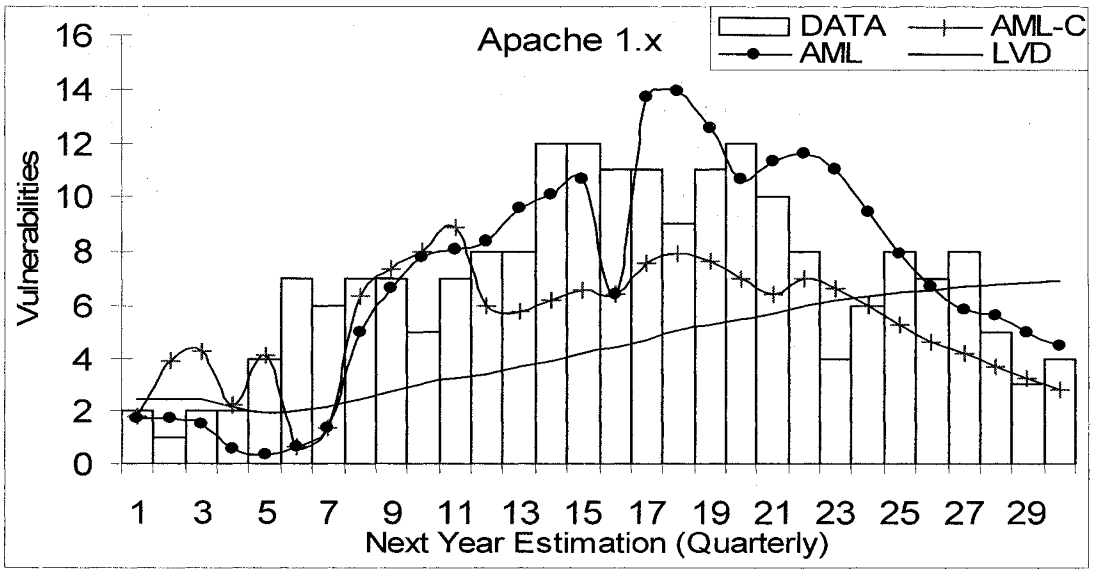

Figure 6-14. Estimations of vulnerabilities for the following year (quarterly)

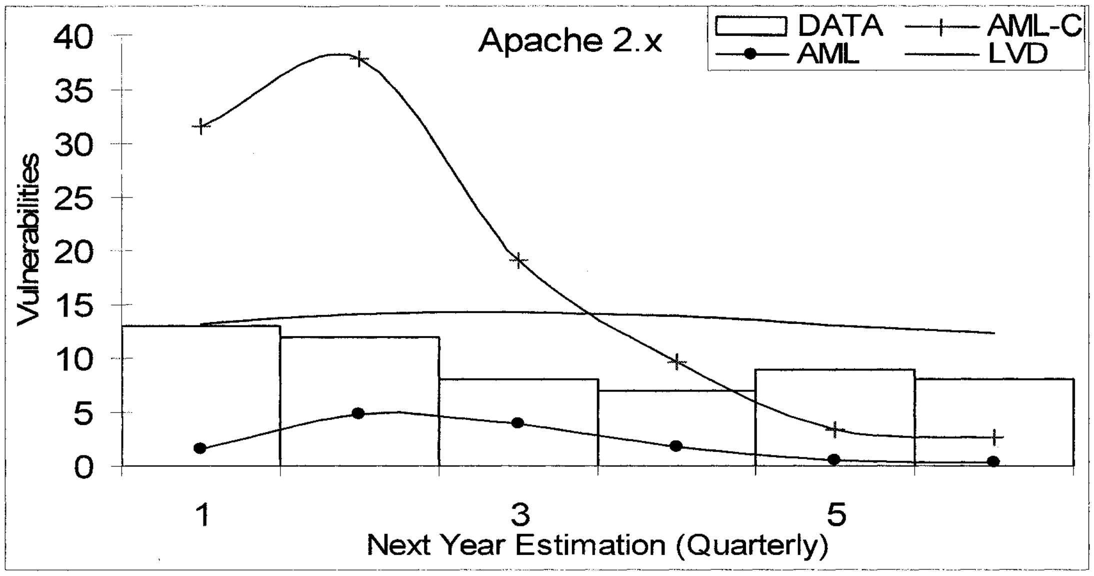

Figure 6-15. Estimations of vulnerabilities for the following year (quarterly)

Table 6-7. Absolute Average Error and Absolute Average Bias for Non-Adaptive Short Term Prediction (in vulnerabilities per year)

Model / Server	AML		AML-C		LM	
	AAE	**AAB**	**AAE**	**AAB**	**AAE**	**AAB**
Apache 1.0	2.16	.11	2.48	-1.57	3.42	-2.51
Apache 2.0	7.32	-7.32	11.57	7.89	4.01	4.01
IIS 4.0	6.51	-6.43	5.57	-4.4	9.7	8.13
IIS 5.0	8.15	-1.97	7.15	-2.87	8.45	-7.13

6.4.6 ESTIMATING THE FOLLOWING YEAR'S VULNERABILITIES BY MODEL APPLICATION WITH ADAPTATION

Here the prediction takes place using a subset of the data to fit the model using least squared method similar to the direct approach; however, half of the error of previous estimations is subtracted from the estimation and a window of the past three errors is used.

Figure 6-16 and Figure 6-17 show the enhancements achieved by using adaptive techniques. The error was chosen to be the average of the past three errors for the LM model. The same was used with AML and AML-C, however, with one difference: the error size was divided by 2 to make it milder. Table 6-8 below shows improvements over Table 6-7 by showing smaller AAE and AAB numbers. Further improvements in finding an optimal choice of error size may yield better estimation accuracy.

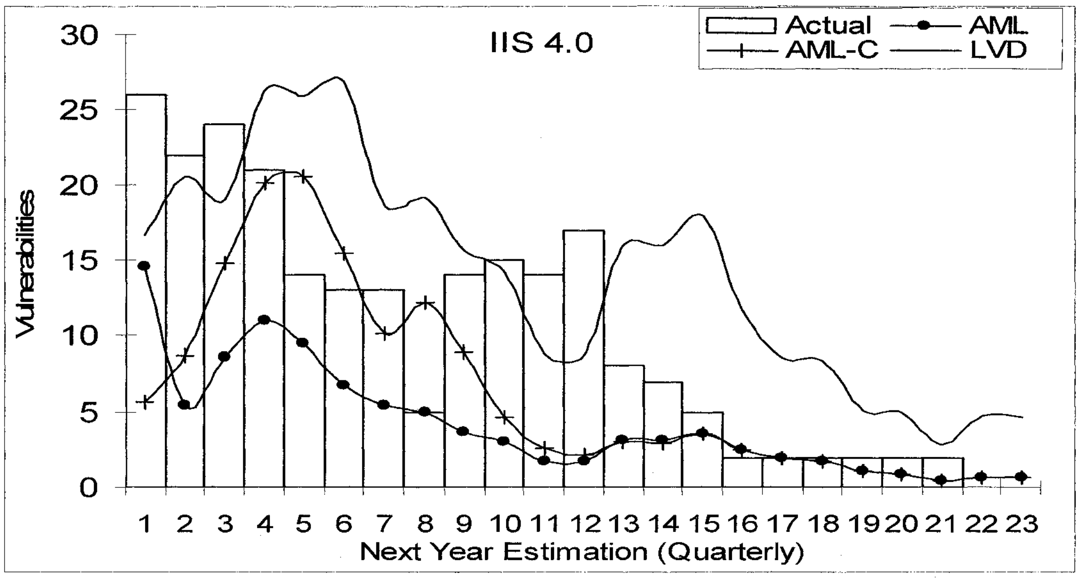

Figure 6-16. Adaptive estimations of vulnerabilities for the following year (quarterly)

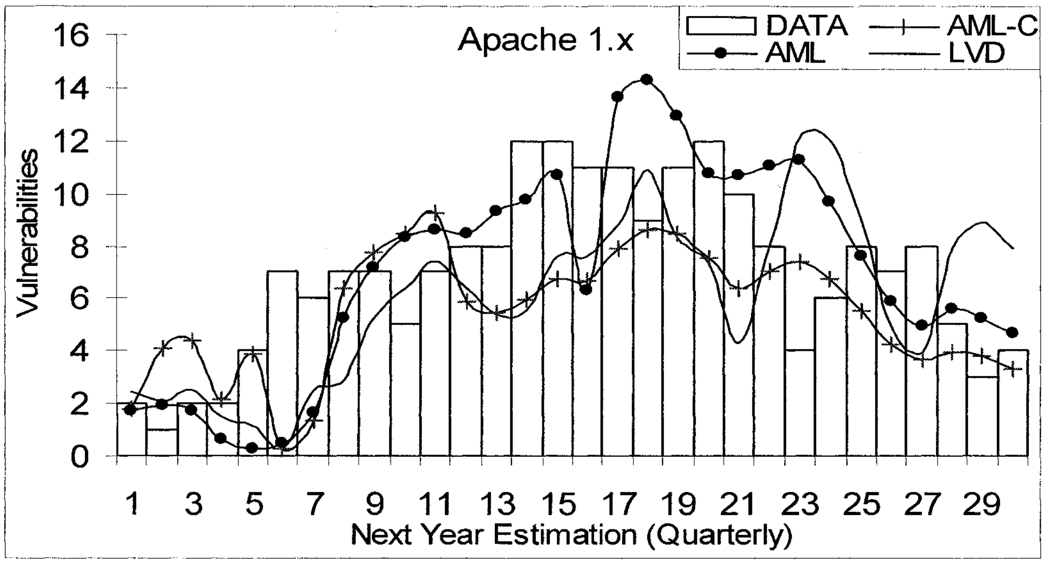

Figure 6-17. Adaptive estimations of vulnerabilities for the following year (quarterly)

Table 6-8. Absolute Average Error and Average Bias for Adaptive Short Term Prediction (in vulnerabilities per year)

Model / Server	AML		AML-C		LM	
	AAE	AAB	AAE	AAB	AAE	AAB
Apache 1.0	2.27	0.12	2.53	-1.4	3.09	-1.4
Apache 2.0	6.54	-6.54	7.31	11.59	3.76	3.76
IIS 4.0	5.98	-5.82	5.24	-3.69	6.57	3.96
IIS 5.0	8.30	-1.74	7.1	-2.42	8.05	6.18

6.5 DISCUSSION

Goodness of fit results for the three-phase AML model show significant fit to all datasets. When using the model for long term predictions, adding a constraint to AML was found to be useful in filtering early extreme estimations. However, the constraint does not significantly improve the estimation of the number of vulnerabilities expected in the next year. Overall, the long term prediction for AML-C has proved to be the most accurate. Moreover, the accuracy of AML/AML-C improves significantly as more data is available, especially with about 65% or more of the data. Recalibration has improved the estimations for both long term prediction and short term prediction. However, in this study, the recalibration adjustment was kept small because sometimes there was no consistent pattern in the bias and, therefore, larger adjustment would make the estimations unstable.

Goodness of fit results for LM show that LM fits when the data set is less mature, suggesting that LM can be better in short term prediction than long term prediction. However, results for LM for the next year's estimation show that overall performance is close to AML and AML-C. For long term prediction LM has larger Average Error; it demonstrates consistent bias, underestimation for the Apache data and overestimation for IIS. For the next year's prediction, heavier recalibration has improved LM performance better than milder recalibration. The overall results suggest that recalibration improves the estimations for all models. The results are similar to those for software reliability growth models [47].

6.6 CONCLUSION

This chapter examines the applicability of two vulnerability discovery models, AML and LM, to the data for two separate versions of Apache and IIS each. The fit of the models was evaluated by computing goodness of fit for each case. The fit was always significant for the AML model. However for LM, the fit was not significant in some of the cases.

The chapter also examines the prediction capability which was tested for both long term predictions (total number of vulnerabilities) and for short term predictions (number of vulnerabilities that will be discovered in the next year). For long term prediction, an adaptive technique was shown to improve estimations. After 65% of the data, the prediction error becomes small. The best results were observed with the AML-C approach, which uses a more stable estimate of the parameter values. For short term prediction, the models were applied directly using recalibration. Recalibration again showed some improvement over direct model application.

AML and LM models can be integrated into the development process to create more secure software systems [73]. An approach recently proposed by Sahinoglu [72] needs such an assessment of the vulnerabilities for estimating risk and the cost of loss. Short term prediction can be used to evaluate the estimated vulnerability discovery rates which would become part of the risk evaluation.

Further work is needed to improve and optimize the adaptive technique. Also, it may be possible to utilize the vulnerability density as a static measure to improve estimation accuracy by stabilizing some of the parameter values.

CHAPTER 7

TAXONOMIZED VULNERABILITIES: ANALYSIS AND MODELING

7.1 INTRODUCTION

Earlier, we have been studying, modeling, and analyzing vulnerabilities, with the assumption that all vulnerabilities are equal. However, vulnerabilities vary in category and severity; some vulnerabilities compromise the system totally, while others have some minimal impact on the system's security. Thus, it would be helpful to take a closer look into categorized vulnerabilities and test if Vulnerability Discovery Models (VDMs) apply to the categorized datasets too. Consequently, we would like to apply some of the proposed models and test the goodness of fit and the adequacy of the models.

Next, we will preview the taxonomy that will be used for our analysis in order to look into how vulnerabilities are classified by category; then, we will look into severity level classification; then, the datasets will be fitted to the models and the goodness of fit will be tested. Finally, the relationship between vulnerabilities category and severity level will be analyzed.

7.2 VULNERABILITY CLASSIFICATIONS BY CATEGORY

Vulnerability taxonomy is still an evolving area of research. A number of vulnerability taxonomies exist, such as Aslam's and Krusl's taxonomies. An ideal taxonomy should have desirable properties such as mutual exclusiveness, clear and unique definition, repeatability, and coverage of all software vulnerabilities. These qualities will provide an unambiguous taxonomy. There is a taxonomy that is close to

Aslam's and Krusl taxonomies, that has gained recognition from organizations such as MITRE Corporation and the NIST NVD which has shown to be acceptable.

Vulnerabilities can be classified using schemes based on cause, severity, impact, source, etc. In this analysis, we use the classification scheme employed by the National Vulnerability Database of the National Institute of Standards and Technology. This classification is based on coding errors and this classification is almost compatible with Aslam's taxonomy[16]. The eight classes are as follows:

- Input Validation Error (IVE) (Boundary condition error (BCE), Buffer overflow (BOF)): Such types of vulnerabilities include failure to verify the incorrect input and read/write involving an invalid memory address.
- Access Validation Error (AVE): These vulnerabilities cause failure in enforcing the correct privilege for a user.
- Exceptional Condition Error Handling (ECHE): These vulnerabilities arise due to failures in responding to unexpected data or conditions.
- Environmental Error (EE): These vulnerabilities are triggered by specific conditions of the computational environment.
- Configuration Error (CE): These vulnerabilities result from improper system settings.
- Race Condition Error (RC): These are caused by the improper serialization of the sequences of processes.
- Design Error (DE): These are caused by improper design of the software structure.
- Others: Includes vulnerabilities that do not belong to the types listed above, sometimes referred to as nonstandard.

Unfortunately, the eight classes are not completely mutually exclusive. A proportion of vulnerabilities can belong to more than one category, because a vulnerability can belong to more than one category, the summation of all categories for a single software system may add up to more than the total number of vulnerabilities. Nonetheless, usually most vulnerabilities fall under one category only.

7.3 VULNERABILITY CLASSIFICATIONS BY SEVERITY

In this section we present an important taxonomy which classifies vulnerabilities by their severity. Here, also where many attempts have been made to provide objective classifications. A heuristic-based classification looks at several attributes of the vulnerability with *Common Vulnerabilities Scoring System* (CVSS) [29].

CVSS is being adapted by the NIST NVD and it provides a standard for vulnerability severity classification. The system gives each vulnerability a value from 1-10, the higher the number, the more severe the vulnerability. The range 1-3.99 corresponds to low severity, 4-6.99 to medium severity and 7-10 to high severity; The National Vulnerability Database of the National Institute of Standards and Technology describes the severity levels, as follows [61]:

- High Severity (CVSS 7 to 10): These vulnerabilities allow remote attackers to violate the security protection or permit a local attack to gain complete control of a system, or if it is important enough to have an associated CERT/CC advisory or US-CERT alert.
- Medium Severity (CVSS 4 to 6.99): This does not meet the definition of either “high” or “low” severity.

- Low Severity (CVSS 1 to 3.99): The vulnerability typically does not yield valuable information or control over a system but rather gives the attacker knowledge or provides the attacker with information that may help him find and exploit other vulnerabilities, or we feel that the vulnerability is inconsequential for most organizations.

7.4 MODELING VULNERABILITIES CLASSIFIED BY CATEGORY

In this section we examine the applicability of the Alhazmi-Malaiya vulnerability discovery models on the datasets for individual vulnerability type or severity level. If the model is applicable to individual categories, it could be expected that estimations for future vulnerabilities of a specific category are possible, giving estimators better details about vulnerability estimation. Furthermore, if the model applies to severity levels, it can also be possible to predict the severity levels of future vulnerabilities [9].

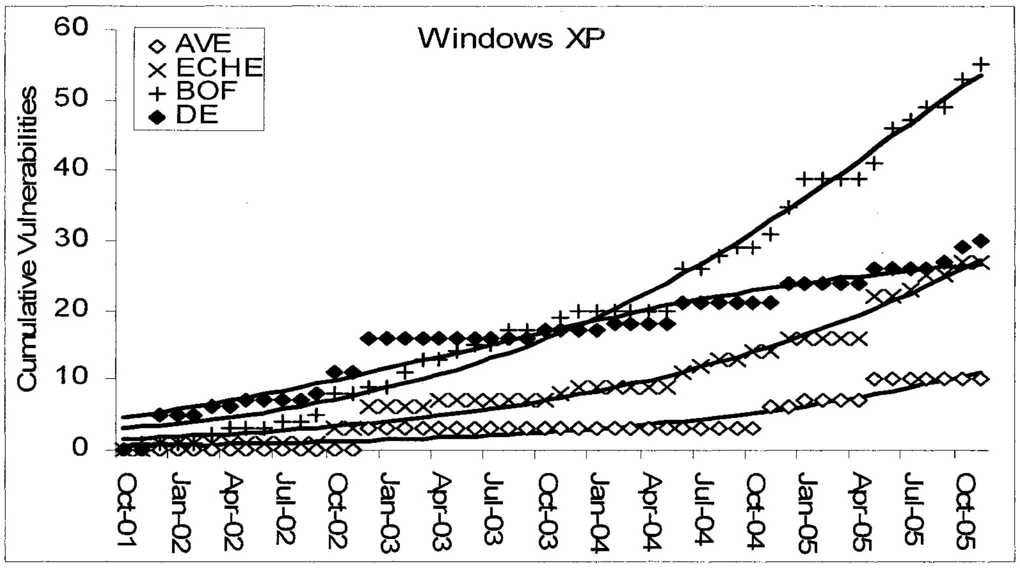

Figure 7-1. The plots for individual vulnerability categories data fitted to the models for Windows XP

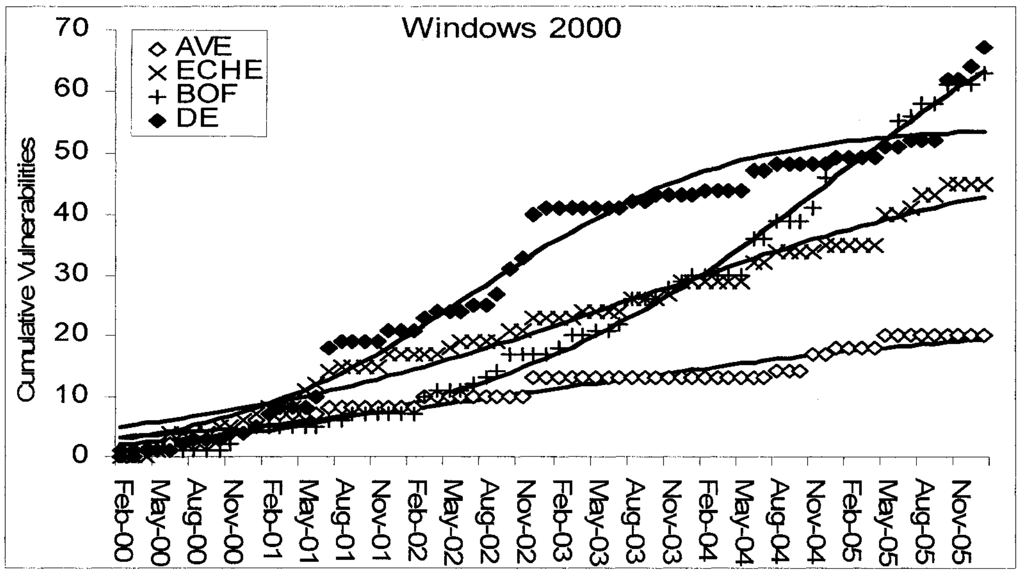

Figure 7-2. The plots for individual vulnerability categories data fitted to the models for Windows 2000

Figure 7-1 and Figure 7-2 show how the model fits the datasets for Windows XP and Windows 2000 for the major individual vulnerability categories; the χ^2 goodness of fit test results in it basically show that there is a significant fit of P-value> .99448 (a P-value closer to 1 indicates a good fit, a P-value less than 0.05 indicates that there is no fit at (α=0.05)).

Table 7-1. Windows XP Goodness of fit results

Vulnerability Type	Parameters			X^2		
	A	B	C	P-value	X^2	$X^2_{Critical}$
AVE	0.001035	63.595	2.0119	1	17.70871	67.50
ECHE	0.000700	92.864	0.674	1	12.10664	
BOF	0.000808	93.578	0.348	1	14.82454	
DE	0.002854	28.984	0.1958	1	13.08505	

Table 7-2. Windows 2000 Goodness of fit results

Vulnerability Type	Parameters			X^2		
	A	B	C	P-value	X^2	$X^2_{Critical}$
AVE	0.002121	23.181	0.279309	1	25.5528	92.81
ECHE	0.001026	51.954	0.189765	0.99956	38.63088	
BOF	0.000676	93.862	0.495049	1	19.62047	
DE	0.001791	54.358	0.343025	0.99448	45.14116	

In this goodness of fit test, we have examined Access Validation Error (AVE), Exceptional Control Handling Error (ECHE), Buffer Overflow (BOF), and Design Error (DE) vulnerabilities. Other vulnerabilities are not statistically significant; therefore, we can not accurately test the goodness of fit for these vulnerabilities (i.e. Race Condition, Environmental Error, etc.).

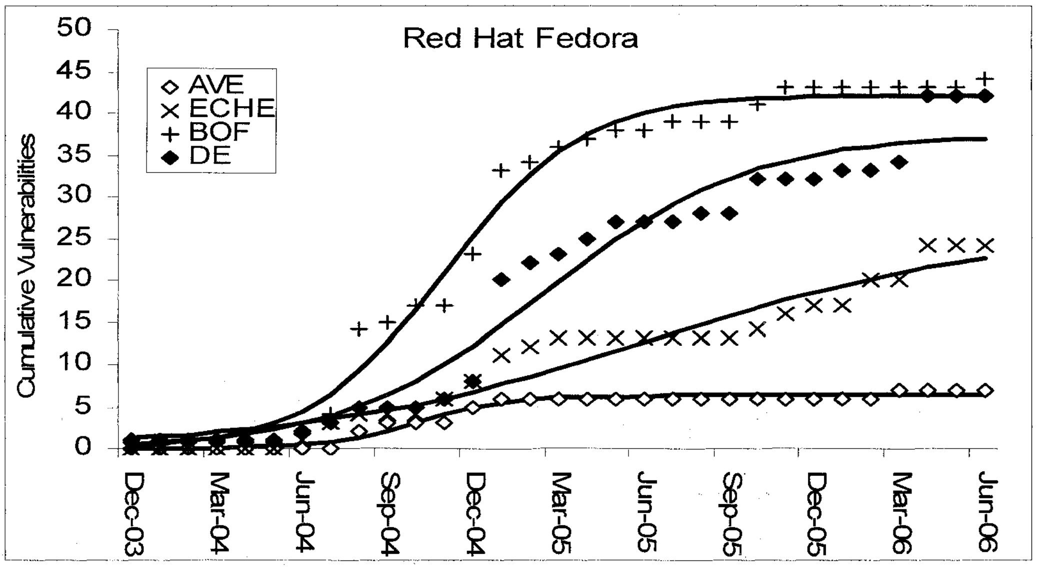

Figure 7-3. Plots for individual vulnerability categories data fitted to the models for Red Hat Fedora

Table 7-3. Red Hat Fedora Goodness of fit results

Vulnerability Type	Parameters			X^2		
	A	B	C	P-value	X^2	X^2_{Critical}
AVE	0.100153	6.3065	172.35	1	3.50293	44.99
ECHE	0.006401	25.7204	0.927	0.90505	21.2541	
BOF	0.0099606	42.1905	3.778	0.99902	12.1771	
DE	0.00753	37.4380	2.1773	0.988797	15.86836	

Figure 7-3 above shows how the AML model fits individual vulnerability categories for Red Hat Fedora; Table 7-3 shows the χ^2 goodness of fit test results; it shows a significant fit for all vulnerability categories examined. Similarly here, we have only examined vulnerabilities with statistically significant data.

Figure 7-4 and Figure 7-5 present the AML model fitting of each web browsers' vulnerabilities by category. We only consider two major categories, design errors, and input validation errors, since other categories have too few vulnerabilities to fit to the model. In Figure 7-4 and Figure 7-5 the bold lines indicate the fitted AML model for each category, and other dotted lines and thin lines indicate cumulative vulnerability data for each category.

Figure 7-4 shows the AML model fitting for categorized IE vulnerabilities. From the beginning until now, the AML model and the cumulative data demonstrate that design errors have been found more frequently than input validation errors, and the gap between design error and input validation error is widening [78].

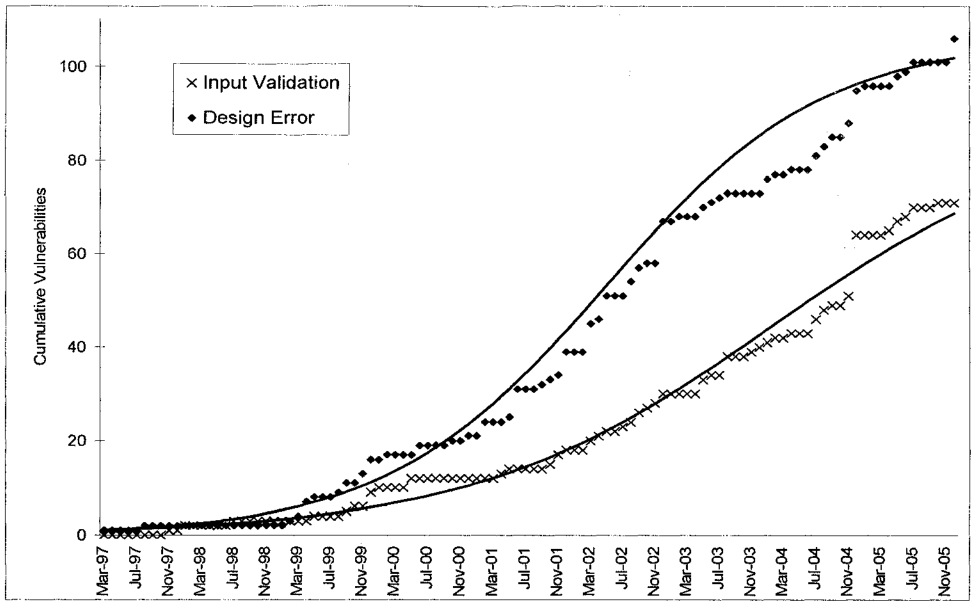

Figure 7-4. Fitting Internet Explorer's Vulnerabilities by Category to AML

Figure 7-5 shows the AML model fitting for categorized Firefox vulnerabilities. Categorized Firefox vulnerabilities show a similar pattern to categorized IE vulnerabilities; design errors have been found more frequently than input validation errors, and the gap between design error and input validation error is widening.

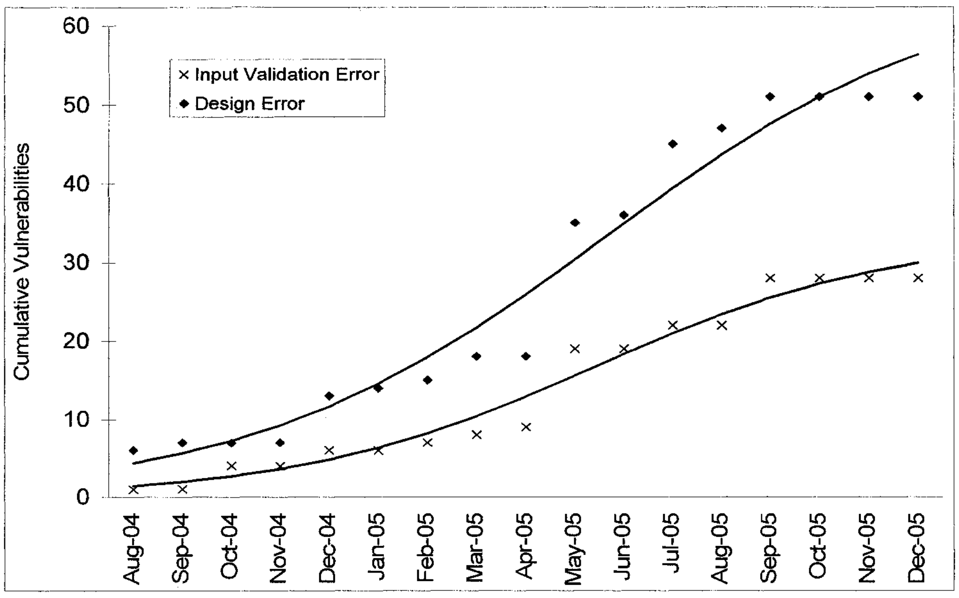

Figure 7-5. Fitting Firefox's Vulnerabilities by Category to AML

Table 7-4 below shows the chi-square goodness of fit tests for IE and Firefox models by category. For each category, the χ^2 value is less than χ^2_{critical}, and the P-values are close to 1. Thus the fit for the two categories is significant for the AML model. It is interesting to note that the fit for the aggregate IE vulnerabilities considered in the previous section was not significant with respect to the significance level chosen.

Table 7-4 Goodness of Fit Test Results for Total Number of Vulnerabilities of categorized vulnerabilities

Browser	**Category**	***A***	***B***	***C***	χ^2	$\chi^2_{critical}$	***P-value***
IE	**Input Validation**	0.00059	89.7	0.984	43.9	135.4	0.99
	Design Error	0.00062	110.9	0.895	47.2	135.4	0.99
Firefox	**Input Validation**	0.0089	35	0.849	6.5	32.6	0.99
	Design Error	0.0042	65.1	0.278	9.1	32.6	0.98

We adapted the AML model to two major vulnerability categories to determine whether there are observable patterns at the level of individual classes. Since we noted a similar pattern for the uncategorized vulnerabilities, a possible fit was examined. These individual classes reflect their own total number of vulnerabilities.

7.5 MODELING VULNERABILITIES CLASSIFIED BY SEVERITY LEVEL

Figure 7-6 and Figure 7-7 show how the AML model fits data separated by severity level. The figures clearly show s-shaped curves with different parameter values, showing that low severity vulnerabilities are discovered at a faster rate. It also opens the possibility of future prediction of vulnerabilities and their severity level.

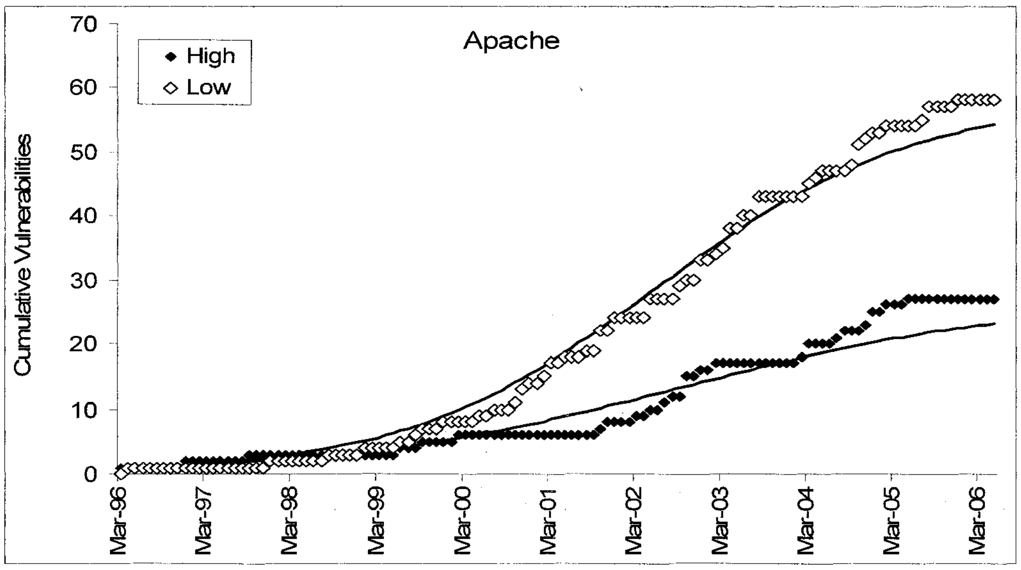

Figure 7-6. Apache by severity level vulnerabilities fitted to the AML

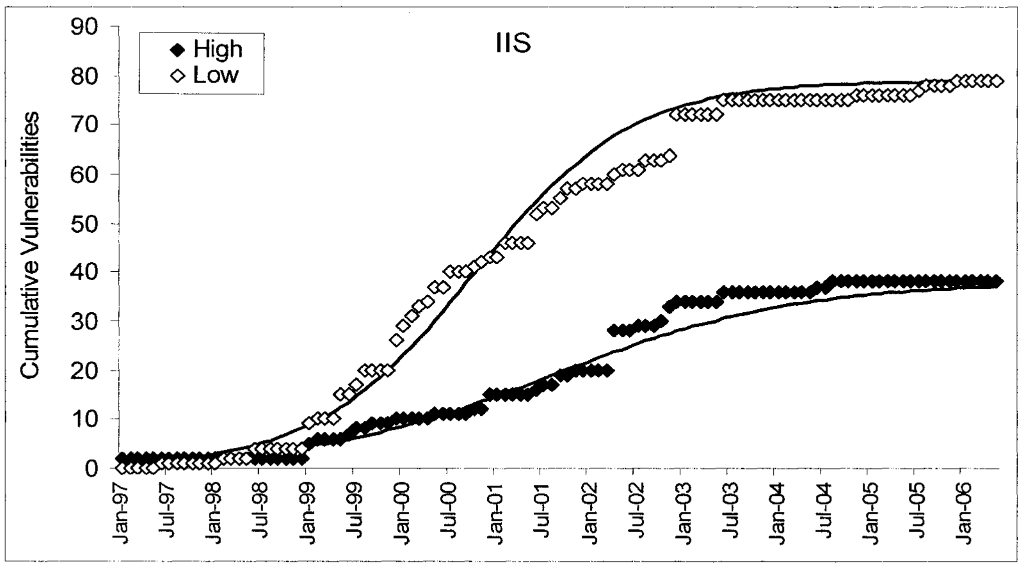

Figure 7-7. IIS by severity level vulnerabilities fitted to the AML

Table 7-5 shows that the fit was significant in all of the cases at (α=.05) with P-values larger than or equal to .999; with the χ^2 value significantly higher than the critical χ^2 in both web servers[77].

Table 7-5.Web servers vulnerability Goodness of fit results for data classified by severity fitted to the AML

Web server	Vulnerability Type	Parameters			χ^2		
		A	B	C	P-value	χ^2	$\chi^2_{Critical}$
IIS	High	.00176	38	.999	1	28.2	133.2
	Low	.00127	77.9	1.21	.999	53.5	
Apache	High	.00156	27.00	1.00	1	42.1	144.3
	Low	.00248	18.00	1.76	1	15.7	

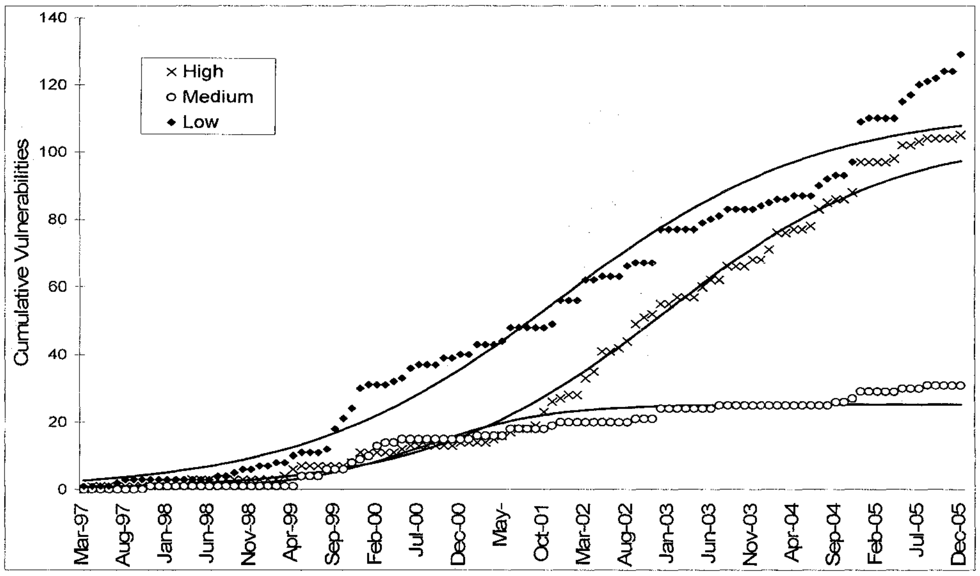

Figure 7-8. Vulnerabilities of Internet Explorer classified by severity level

We apply the AML model to the two web browsers. Figure 7-8 and Figure 7-9 show the results of fitting the AML model to the three severity classes. The fit is significant for all cases[78].

Table 7-6 shows that the severity-classified datasets for Internet Explorer and Firefox fit the AML model. In all cases, the chi-square values were less than the chi-square critical shown in the tables, and the P-values were larger than 0.12.

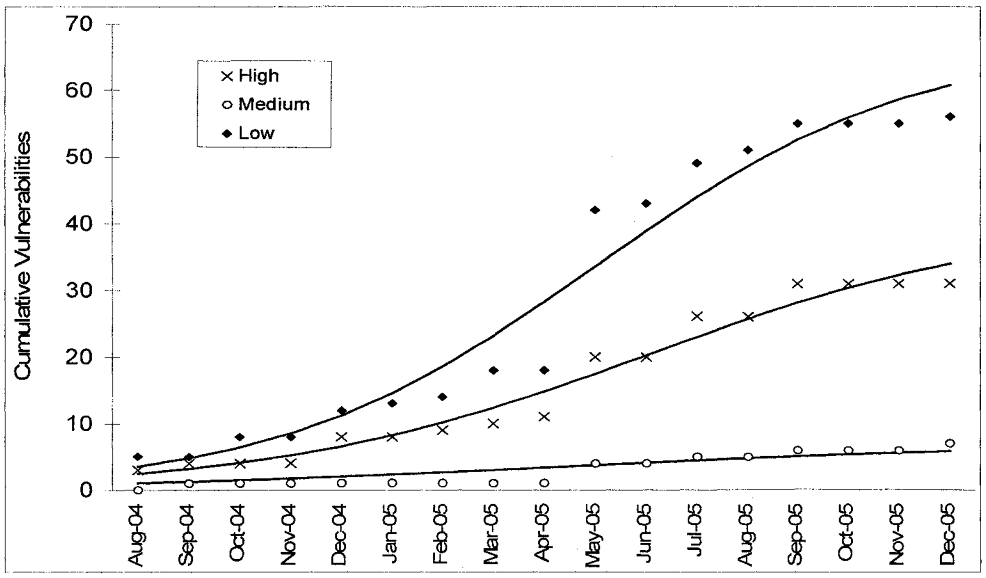

Figure 7-9.Vulnerabilities of Firefox classified by severity level

Table 7-6. Goodness of fit results for IE, and Firefox and datasets classified by severity level fitted to the AML

Browser	Severity	*A*	*B*	*C*	χ^2	$\chi^2_{critical}$	*P-value*
IE	**High**	0.0006	110	1.255	40.23	135.4	1
	Medium	0.0057	25.1	12.47	46.38	135.4	0.99
	Low	0.00049	122.2	0.325	107.42	135.4	0.12
Firefox	**High**	0.0057	44.1	0.447	8.83	32.6	0.98
	Medium	0.0206	11.3	2.56	4.02	32.6	0.99
	Low	0.0046	67.6	0.36	12.1	32.6	0.91

7.6 ANALYZING THE LINK BETWEEN VULNERABILITY CATEGORIES AND VULNERABILITY SEVERITY LEVELS

Figure 7-10 compares vulnerability distributions in three web browsers. More than 60% of found vulnerabilities are related to design or input validation error. When comparing web browsers to web servers (Apache and IIS) and operating systems (Windows 2000 and XP), we find a comparable pattern.

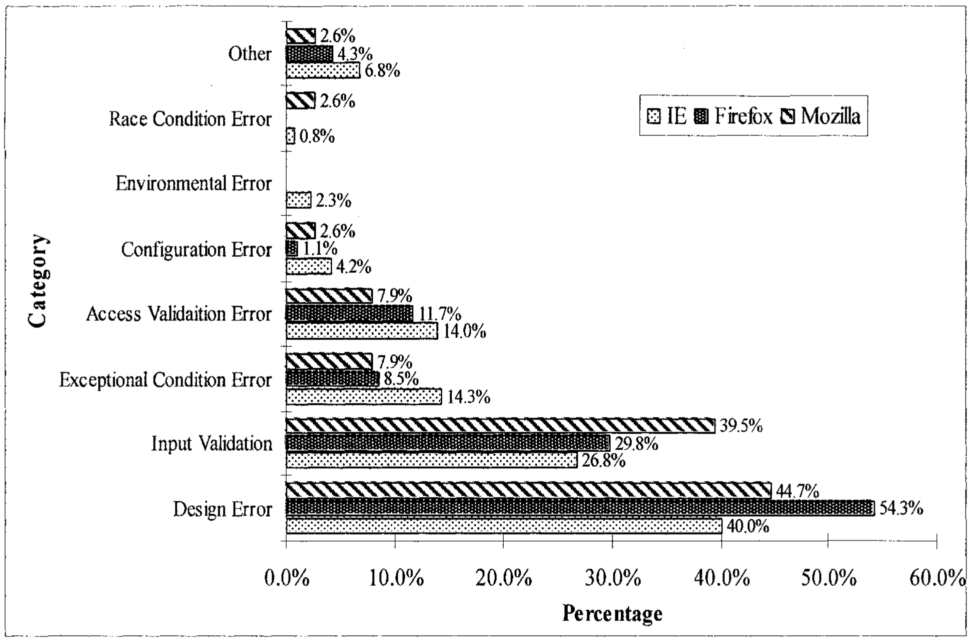

Figure 7-10: Vulnerabilities by category in Internet Explorer, Firefox and Mozella

Usually web server and operating systems have a greater number of vulnerabilities in input validation error than in design error. Except for these two categories, other classes show the following priority order: exceptional condition error, access validation error, configuration error and other classified errors.

The goal of this analysis is to determine which vulnerabilities are more severe so testers can target vulnerabilities of high severity. We can identify the most severe vulnerability categories, so testers can design tests targeting a specific category of vulnerabilities.

The analysis will include the following diverse group of software systems: Windows 98, Windows XP, Windows 2000, Red Hat Fedora, Apache, and IIS web Servers. The vulnerability data are from the National Vulnerabilities Database maintained by NIST. The market share data from Netcraft [59] used shows that high severity vulnerabilities constitute half the number of vulnerabilities, making them the most common vulnerabilities. A significant number of high severity vulnerabilities in Windows 2000 are buffer overflow then design error and boundary condition.

Boundary condition errors (IVE-BCE) have shown to be of high severity only in Windows systems (see Tables 8 and 9) while it is not so in Fedora, IIS and Apache see Tables 10-12. This possibly indicates that there is a common flaw in Windows regarding boundary testing.

Table 7-7 to Table 7-10 show a common observation that exceptional control handling error (ECHE) is associated with low severity vulnerabilities. This observation is common in all examined datasets. The tables show that race condition errors, environmental errors, and configuration errors are very few, which makes it hard to link them to a particular severity level.

Table 7-10 below shows that only 35 Apache vulnerabilities are of high severity, about half of which are Input validation errors with some buffer overflows and general input validation error. Design errors count for 23 vulnerabilities in total with only 5 high severity vulnerabilities.

Table 7-9 shows Red Hat Fedora vulnerabilities. The table shows that about 41.3% of vulnerabilities are of high severity, the majority being buffer overflow. Here we can observe a similarity to both Windows systems in the dominance of buffer overflow and other input validation errors. This may be due to using C programming language, which does not have memory management features. Furthermore, design errors accounts for fifteen high severity vulnerabilities which represents a significant proportion number. Exceptional condition handling errors are mostly low severity vulnerabilities, similar to Windows systems.

Table 7-7. Vulnerabilities Type vs. Severity for Windows 2000

	AVE	DE	ECHE	IVE-BOF	IVE-BCE	IVE-other	RC	EE	CE	other	Total
High	10	29	12	52	18	7	1	2	4	1	135
Medium	6	15	2	5	0	2	0	0	2	1	33
Low	6	26	32	9	3	16	1	2	3	1	100
Total	22	70	46	66	21	25	2	4	9	3	268

Table 7-8. Vulnerabilities Type vs. Severity for Windows XP

	AVE	DE	ECHE	IVE-BOF	IVE-BCE	IVE-other	RC	EE	CE	other	Total
High	8	11	7	47	16	5	1	1	0	2	98
Medium	0	4	2	5	0	3	0	0	0	2	16
Low	3	15	19	4	1	12	1	1	2	0	58
Total	11	30	28	56	17	20	2	2	2	4	172

Table 7-9. Vulnerabilities Type vs. Severity for Red Hat Fedora

	AVE	DE	ECHE	IVE-BOF	IVE-BCE	IVE-other	RC	EE	CE	other	Total
High	2	15	4	33	6	9	3	0	1	1	74
Medium	1	4	0	3	2	2	0	0	0	1	13
Low	3	30	19	9	9	17	0	2	1	2	92
Total	6	49	23	45	17	28	3	2	2	4	179

Table 7-11 below shows that 85 of IIS vulnerabilities are of low severity, while 41 are of high severity. Most high severity vulnerabilities belong to Input Validation Error vulnerabilities and most of them are buffer overflow. Design error vulnerabilities account for twenty-six vulnerabilities, only ten of which are highly severe, showing some errors with the design of IIS. However, IIS has matured over the years and has not shown new vulnerabilities in some time now.

From Table 7-10 and Table 7-11 the distributions of the severities of the Apache and IIS vulnerabilities show similarity. About 60% of total vulnerabilities have low severity, followed by about 30% with high severity, with very few vulnerabilities of medium severity.

Table 7-10. Vulnerabilities Type vs. Severity for Apache

	AVE	DE	ECHE	IVE-BOF	IVE-BCE	IVE-other	RC	EE	CE	other	Total
High	3	5	3	7	1	10	0	3	2	1	35
Medium	0	1	1	3	0	0	1	0	0	0	6
Low	3	17	12	3	1	14	1	1	12	4	68
Total	6	23	16	13	2	24	2	4	14	5	109

Table 7-11. Vulnerabilities Type vs. Severity for IIS

	AVE	DE	ECHE	IVE-BOF	IVE-BCE	IVE-other	RC	EE	CE	other	Total
High	3	10	2	13	1	8	0	0	3	1	41
Medium	0	1	0	1	0	0	0	0	1	0	3
Low	14	15	13	4	8	22	1	3	2	3	85
Total	17	26	15	18	9	30	1	3	6	4	129

Table 7-10 and Table 7-11 illustrate how severity level correlates with error classification. It is noticeable that Exceptional control handling error constituted the majority among low severity vulnerabilities for both Apache and IIS. In IIS, Buffer overflow vulnerabilities are associated with high severity. A relatively smaller fraction of exceptional condition errors are of high severity. In IIS as well, the exceptional condition errors tend to be from among the vulnerabilities with low severity. For IIS, most configuration errors are of medium severity. Table 7-11 shows that Input Validation Errors other than Buffer overflow and Boundary Condition Error are likely to be of low severity.

7.7 CONCLUSIONS

In the past, security vulnerabilities often have been either studied as individual vulnerabilities or as an aggregate number. This chapter examines categories of vulnerabilities. It explores the applicability of quantitative models for the number of vulnerabilities and vulnerability discovery rates for some open source and commercial operating systems and web servers. The results show that individual vulnerability categories and severity level datasets conform to the AML vulnerability discovery models. This work can be extended by examining the prediction capabilities of the models for individual categories and severity level.

A limitation of using the model with vulnerability categories and severity level is that some categories and severity levels could not be modeled because the number of vulnerabilities in those categories is statistically insignificant.

The vulnerability categories-severities analysis was able to link Buffer overflow to high severity vulnerabilities, and exceptional control handling error to low severity vulnerabilities. Input Validation errors other than Buffer overflow and boundary condition error have also shown to be linked to low severity vulnerabilities.

Design Error vulnerabilities were found to be a significant category of which a significant proportion is usually of high severity. This indicates that there is a need to take past mistakes of those vulnerabilities into consideration when designing new software systems. Integrating security design into the lifecycle of the software system has become very important, as suggested by Seacord in[73].

A vulnerability profile is useful documentation that can assist developers to learn from past mistakes and highlight the weaknesses of the software. It can point to ineffective and wrongful practices shown to be responsible for the introduction of some vulnerability. Further research can investigate additional attributes of vulnerabilities possible patterns.

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

8.1 SUMMARY AND CONCLUSIONS

Motivated by the need to develop quantitative measures to characterize a system's security, this research considers using vulnerabilities and vulnerability discovery rate as a major component in a quantitative risk evaluation approach. Quantitative assessment for software security is needed for effective testing, maintenance, security assurance, and risk assessment of software systems. Vulnerabilities that are present in a software system after its release can represent a considerable degree of risk. This work presents solutions to how vulnerabilities can be used to quantitatively assess the security in software systems.

8.1.1 MODELING OF VULNERABILITIES

In this research, vulnerabilities discovered plotted over calendar time trends were analyzed, to determine the major factors impacting vulnerability detectection. Plots showing the cumulative number of vulnerabilities over calendar time for a number of software systems are given. New vulnerability discovery models (VDMs) analogous to Software Reliability Growth Models (SRGMs) are proposed. Just as SRGMs use defects to describe reliability; VDMs use vulnerabilities instead.

A logistic model named Alhazmi-Malaiya Logistic Model (AML) was proposed and found to be conforms to typical vulnerability trends. The logistic model given by Equation (4) is fitted to vulnerability data for a number of software systems and the fit is found to be statistically significant in almost all cases. We also observe that the code shared by a new and hence a competing version of the operating system can impact the vulnerability discovery rate in a previous version. The results show that the proposed models fit the actual vulnerability discovery process very well. Also, a simplification of the AML model, a linear model (LM) was suggested and has shown to fit some datasets in certain situations. An effort-based model that uses the installed-base share data has been proposed and fitted to the data and has shown a significant fit in all tested cases.

The research shows a strong correlation between traditional defects and vulnerabilities. However, there are some fundamental differences between testing for traditional defects and testing for vulnerabilities. For example, systematic software testing in a software development organization occurs prior to release and the bugs are found internally in the developing organization; on the other hand, vulnerability discovery occurs throughout the product lifetime and the vulnerabilities are found both internally and externally. Moreover, compared with ordinary defects, the number of vulnerabilities is very small. The initial growth rate in the number of cumulative vulnerabilities at the release time is small but subsequently accelerates. Generally the plots show a linear trend for a significant period. Eventually these plots tend to show some saturation, often followed by abrupt increases later. This behavior is explained by the variability of the effort that goes into discovering the vulnerabilities.

8.1.2 Vulnerability Density

Vulnerability Density is a static measure that is defined, analyzed, and measured here in several software systems, a conceptually similar measure was named V-Density measure by Ounce labs [63]. As it has been observed for software defect densities, the values of vulnerability densities fall within a range, and for similar products they are closer together. We note that the ratio of vulnerabilities to the total number of defects is often in the range of 1% to 5%, as was speculated to be the case by some researchers [13][42]. As we would expect, this ratio is often higher for operating systems intended to be servers. The results indicate that vulnerability density is a significant and useful metric. We can expect to gain further insight into vulnerability densities when additional data, together with suitable quantitative models, are available. Such models may allow empirical estimation of vulnerability densities along the lines of similar models for software cost estimation or software defect density estimation.

8.1.3 PREDICTION CAPABILITIES

Vulnerability discovery models have been analyzed statistically and analytically. However, models with a good fit will not necessarily have good estimation capability, thus the ability of the models to estimate the future vulnerability discovery rate needed to be examined. Hence, the prediction capabilities of the logistic and linear models have been investigated by evaluating the accuracy of predictions made with partial data. In addition to VDMs, we consider static approaches to estimating some of the major attributes of the vulnerability discovery process, presenting a static approach to estimating the initial values of one of the VDM's parameters. We also suggest the use of constraints for parameter estimation during curve-fitting. Here we develop computational approaches for early applications of the models and examine the predictive capability of the models.

Using data from Windows 98, Windows 2000, and Red Hat Linux 7.1, the impact of imposing a specific constraint on some parameters of the logistic model are evaluated using the average error and average bias measures. Prediction errors are plotted for the estimations taken at different times. The results demonstrate that the prediction average error is significantly less when a constraint based on past observations is added. It is also observed that the linear model may yield acceptable projections for some systems for which vulnerability discovery has not yet reached saturation. The results suggest that it may be possible to improve the prediction capability by combining static and dynamic approaches, or by combining different models.

The prediction capability study was expanded to include web-servers. Here also the AML, AML with constraints, and the LM models were considered. However, at this point, in addition to testing the accuracy of predicting the total number of vulnerabilities (which can be regarded as long term prediction), the accuracy of the prediction of vulnerabilities that will be discovered in the following year is also tested (we refer to this as short term prediction). The estimation was enhanced by utilizing on adaptive technique which was applied and has shown improvement in the accuracy by adjusting the error of estimation based on previous observations. Similar techniques were used in the past for testing the prediction capabilities of Software Reliability Growth models to predict ordinary defects.

The prediction capability test results show that with at least 65% of the data, the error becomes small. The best results were observed with the AML-C approach, which uses a more stable estimate of the parameter values. It is more accurate than the AML or LM approaches. For short term prediction, using adaptation, showed some improvement over direct model application.

8.1.4 Comparing Vulnerability Discovery Models

A few other vulnerability discovery models have been proposed recently. A comparative study was conducted to compare how these models fit the vulnerability trends. The result has shown that the logistic model is generally the most successful and by far the closest to actual trends observed.

Four vulnerability discovery models were examined using Akaike Information Criteria (AIC) and chi-square (χ^2) tests for several operating systems. Results show that the AML model is generally the best for the longer term, performing better for Windows 95 and Linux 6.2. Because it captures the S-shape pattern in the data, it has better fit as determined by using AIC and the chi-square test. RQ, RE, LP and AML all show very good fit for Windows XP, a system that has not yet shown signs of saturation. This can be attributed to the fact that they can fit trends that are largely linear. The AT model considered did not perform well in general.

When VDMs were applied initially, all vulnerabilities were treated equally. Later, taxonomies were used to look at each category individually. The Logistic Alhazmi-Malaiya model (AML) was applied to individual categories, and it has shown to fit the data indicating that it is possible to estimate vulnerabilities belonging to a particular category.

We expect that with further research and significant data collection and analysis, it will be possible to develop reliable quantitative methods for security akin to those used in the software and hardware reliability fields.

Vulnerability discovery models can be used by both the developers and the user community. Developers can assess the product readiness by projecting future vulnerability discovery trends. Developers need to allocate security maintenance resources to detect vulnerabilities, preferably before others do and to release security patches as soon as possible. The users also need to assess the risk due to vulnerabilities before patches are applied. A patch may need to be tested for stability before it is applied, as discussed by Brykczynski et al. [24] and Beattie et al. [18]. Effective security policy at an organization would require time and resources. Vulnerability discovery models can be used to quantitatively guide such policies.

8.1.5 Modeling Categorized Vulnerabilities and Testing For Vulnerabilities

We also studied the modeling of categorized vulnerability datasets; the AML model was fitted to major categories of software vulnerabilities, and has shown to fit the data; the significance of this result is that it opens the door to the prediction of specific categories of vulnerabilities. Moreover, vulnerability datasets classified by severity level have also shown to fit the AML model, indicating a similar conclusion that the AML model is expected to be capable of estimating vulnerabilities of a specific severity level.

Vulnerabilities vary in their categories and severity levels; the research has shown a link between a vulnerability category and its severity level. Therefore, it is feasible to optimize testing to focus it on vulnerabilities with high severity levels and thus to vulnerabilities belonging to certain categories and to design testing cases based on this analysis.

There are two separate processes to be considered. The first is the vulnerability discovery process, while the second is the exploitation of individual vulnerabilities discovered. In this chapter, we examine modeling the first process. While the two processes are distinct, evaluation of the overall risk should involve a joint consideration of both processes. Obviously, a vulnerability needs to be discovered before it can be exploited. While those who attempt to exploit vulnerabilities may often be amateurs, those who discover vulnerabilities must have significant technical expertise.

8.2 FUTURE WORK

The shared vulnerabilities among successive versions constitute an important component of vulnerability discovery process. Evaluation of their impact requires some analysis of the results of applying patches and code re-use because shared vulnerabilities are due to code overlap among consecutive versions of software systems. A suggested approach would be to include careful white box analysis with the available vulnerabilities to determine the circumstances that cause vulnerabilities in the first place. This is especially important because the current analysis disregards the architecture of the software system. This analysis assumes that vulnerabilities are found in random places within software systems. In some specific modules, exploiting information concerning the architecture of the system can focus security testing to the code with a higher chance of finding vulnerability, which will make security testers more efficient.

Further work is needed to enhance the adaptive technique in short term prediction, currently, we use error subtraction method. It has shown some improvement; however, using a method with artificial intelligence capability (e.g. neural networks) may improve the prediction accuracy.

Users need a reliable risk assessment technique that is based on vulnerabilities discovery models and vulnerability density could be practical combined with using real-time incidents analysis. This suggests integrating VDMs with risk assessment models that consider other factors into a comprehensive risk assessment prospective analogous to Sahinoglu's risk assessment model [72].

This research has yielded some guidelines for testing for vulnerabilities based on the relationship between vulnerability categories and severity levels. However, because the taxonomy is not sufficiently testing-oriented, when a more testing-oriented taxonomy is available, better guidelines can be developed for vulnerabilities testers. Recently, a new testing-oriented taxonomy has been proposed [17]; however, the authors were able to classify only around 50% of the vulnerabilities. This is due to the lack of some essential low-level details because some vulnerability data is proprietary.

Further analysis of the vulnerability discovery process in needed in order to help create highly secure software systems. Issues like seasonality could be worthy of investigations, to answer questions like - are there months that are potentially risky? If so, what factors could contribute to this behavior? Also, the reward factor for vulnerability discovery needs to be analyzed; several vendors offer financial compensation to those finding new vulnerabilities, while in other cases the reward to individuals is more indirect. Such, incentives can significantly impact the vulnerability discovery rates.

REFERENCES

[1] A. A. Abdel-Ghaly, P. Y. Chan and B. Littlewood, Evaluation of Competing Software Reliability Predictions, IEEE Transactions on Reliability, Volume SE-12(9), 950-967, September 1986.

[2] H. Akaike, Prediction and Entropy, MRC Technical Summary Report #2397, NTIS, Springfield, VA, USA, 1982.

[3] Apache Software Foundation Bug System, http://issues.apache.org/bugzilla/.

[4] O. H. Alhazmi and Y. K. Malaiya, Quantitative vulnerability assessment of systems software, Proceedings of 51st Annual Reliability and Maintainability Symposium, Alexandria, VA, USA, 615–620, January 2005.

[5] O. H. Alhazmi, Y. K. Malaiya and I. Ray, Security Vulnerabilities in Software Systems: A Quantitative Perspective, Proceedings Annual IFIP WG11.3 Working Conference on Data and Information Security, Storrs, CT, USA, 281-294, August 2005.

[6] O. H. Alhazmi and Y. K. Malaiya, Modeling the Vulnerability Discovery Process, Proceedings International Symposium on Software Reliability Engineering, Chicago, IL, USA, 129-138, November 2005.

[7] O. H. Alhazmi and Y. K. Malaiya, Prediction Capability of Vulnerability Discovery Models, Proceedings of 52nd Annual Reliability and Maintainability Symposium, Newport Beach, CA, USA, 86-91, January 2006.

[8] O. H. Alhazmi and Y. K. Malaiya, Measuring and Enhancing Prediction Capabilities of Vulnerabilities Discovery Models for Apache and IIS HTTP Servers, to appear in Proceedings of 17th IEEE International Symposium on Software Reliability Engineering, Raleigh, NC, USA, 343-352, November 2006.

[9] O. H. Alhazmi, S.-W. Woo, and Y. K. Malaiya, Security Vulnerability Categories in Major Software Systems, Proceedings of IASTED International Conference on Communication, Network and Information Security, Cambridge, MA, **USA,** 138-143, October 2006.

[10] O. H. Alhazmi, Y. K. Malaiya and I. Ray, Measuring, Analyzing and Predicting Security Vulnerabilities in Software Systems, Journal of Computers & Security, 2007. (To appear)

[11] J. Alves-Foss and S. Barbosa, Assessing Computer Security Vulnerability, Operating Systems Review, Volume 29(3), 3-11, 1995.

[12] J. Amor-Iglesias, J. González-Barahona, G. Robles-Martínez, and I. Herráiz-Tabernero, Libre Software as A Field of Study, The European Journal for the Informatics Professional, Volume VI(3), 13-16, June 2005.

[13] R. Anderson, Why Information Security is Hard—An Economic Perspective, Proceedings of the 17th Annual Computer Security Applications Conference, New Orleans, LA, USA, 358-365, December 2001,

[14] R. Anderson, Security in Open versus Closed Systems—The Dance of Boltzmann, Coase and Moore, Proceedings of Conference on Open Source Software: Economics, Law and Policy, Toulouse, France, 1-15, June 2002,

http://www.ftp.cl.cam.ac.uk/ftp/users/rja14/toulouse.pdf

[15] W. A. Arbaugh, W. L. Fithen and J. McHugh, Windows of Vulnerability: A Case Study Analysis, IEEE Computer, Volume 33(12), 52-59, December 2000.

[16] T. Aslam, and E. H. Spafford, A Taxonomy of Security Faults, Technical Report TR-096-051, Carnegie Mellon, 1996.

[17] Anil Bazaz, James D. Arthur, Joseph G. Tront, Modeling Security Vulnerabilities: A Constraints and Assumptions Perspective, Proceedings the 2nd IEEE International Symposium on Dependable Autonomic and Secure Computing, Indianapolis, IN, USA, 94-102, September 2006.

[18] S. Beattie, S. Arnold, C. Cowan, P. Wagle and C. Wright, Timing the Application of Security Patches for Optimal Uptime, Proceedings of the 14th Large Installation System Administration Conference, New Orleans, LA, USA, 233-242, November 2002.

[19] P. G. Bishop and R. E. Bloomfield, A Conservative Theory for Long-Term Reliability Growth Prediction, IEEE Transactions on Reliability, Volume 45(4), 550-560, December 1996.

[20] H. K. Browne, W. A. Arbaugh, J. McHugh and W. L. Fithen, A Trend Analysis of Exploitation, In Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA, USA, 214–229, May 2001.

[21] R. M. Brady, R. J. Anderson and R. C. Ball, Murphy's Law, the Fitness of Evolving Species, and the Limits of Software Reliability, Cambridge University Computer Laboratory Technical Report Number 471, September 1999,

http://www.cl.cam.ac.uk/ftp/users/rja14/babtr.pdf.

[22] S. Brocklehurst, P. Y. Chan, B. Littlewood and J. Snell, Recalibrating Software Reliability Models, IEEE Transactions on Software Engineering, Volume 16(9), 458-470, April 1990.

[23] S. Brocklehurst, B. Littlewood, T. Olovsson and E. Jonsson, On Measurement of Operational Security, Proceedings of 9th Annual IEEE Conference on Computer Assurance, Gaithersburg, MD, USA, IEEE Computer Society, 257-266, 1994.

[24] B. Brykczynski and R. A. Small, Reducing Internet-Based Intrusions: Effective Security Patch Management, IEEE Software, Volume 20(1), 50-57, January-February 2003.

[25] M. Dacier, Y. Deswarte and M. Kaâniche, Quantitative Assessment of Operational Security: Models and Tools, Technical Report, LAAS Report 96493, May 1996.

[26] D. E. Denning, A lattice model of secure information, Communications of the ACM, Volume 19(5), 236--243, May 1976.

[27] Google Press Center, Google Zeitgeist, www.google.com/press/zeitgeist, June 2004.

[28] L. Goel and K. Okumoto, Time-Dependent Error Detection Rate Model for Software and Other Performance Measures, IEEE Transactions on Reliability, (R-28)3:206-211August 1979.

[29] Forum of Incident Response and Security Teams (FIRST), http://www.first.org/, October 2006.

[30] R. Ford, H. Thompson and F. Casteran, Role comparison report—web server role. Technical Report, Security Innovation, 2005.

[31] R. Gopalakrishna and E. H. Spafford, A Trend Analysis of Vulnerabilities, CERIAS Tech Report 2005-05, Center for Education and Research in Information Assurance and Security, Purdue University, https://www.cerias.purdue.edu/tools_and_resources/bibtex_archive/archive/, 2005.

[32] J. Hallberg, A. Hanstad, and M. Peterson, A Framework for System Security Assessment. Proceeding 2001 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 214–229, May 2001.

[33] A. Hussain, J. Heidemann, and C. Papadopoulos. A Framework for Classifying Denial of Service Attacks-extended. Technical Report ISI-TR-2003-569b, USC/Information Sciences Institute, June 2003.

[34] Identifying Important Operating Systems, http://www.philosophe.com/audience/operating_systems.html, May 1999.

[35] E. Jonsson and T. Olovsson, A Quantitative Model of the Security Intrusion Process Based on Attacker Behavior, IEEE Transactions on Software Engineering, 235-245, April 1997.

[36] I. Krusl, E. Spafford and M. Tripunitara, Computer Vulnerability Analysis, Department of Computer Sciences, Purdue University, COAST TR 98-07, 1998.

[37] B. Krebs, Real world impact of IE flaw, Brian Krebs on Computer Security, http://blog.washingtonpost.com/securityfix/2006/04/real_world_impact_of_internet_1.html, April 2006.

[38] C. Landwehr, Formal Models for Computer Security, Computer Surveys, Volume (13)3: 247-278, September 1981.

[39] D. Larochelle and D. Evans, Statically Detecting Likely Buffer Overflow Vulnerabilities, Proceedings of 10th USENIX Security Symposium, Washington, DC, USA ,177-190, August 2001.

[40] N. Li, Y. K. Malaiya, Enhancing Accuracy of Software Reliability Prediction, Proceedings 4th International Symposium on Software Reliability Engineering 71–79, Research Triangle, NC, USA, November 1993.

[41] B. Littlewood, S. Brocklehurst, N. Fenton, P. Mellor, S. Page and D. Wright, Towards Operational Measures of Computer Security, Journal of Computer Security, Volume (2) 2/3: 211-230, 1993.

[42] T. Longstaff, CERT Experience with security problems in Software, Carnegie Mellon Univ., http://research.microsoft.com/projects/SWSecInstitute/slides/Longstaff.pdf, June 2003.

[43] D. D. Longley, and M. Shain, Data and Computer Security: Dictionary of Standards, Concepts, and Terms. Stockton Press, 1987.

[44] M. R. Lyu, Ed., Handbook of Software Reliability Engineering, McGraw-Hill, 1995.

[45] B. B. Madan, K. Goseva-Popstojanova, K. Vaidyanathan and K. S. Trivedi, Modeling and Quantification of Security Attributes of Software Systems, In Proceedings of IEEE Int. Performance and Dependability Symposium, June 2002.

[46] Y. K. Malaiya and P. K. Srimani, Software Reliability Models: Theoretical Development, Evaluation and Applications, IEEE Computer Society Press, 1990.

[47] Y. K. Malaiya, N. Karunanithi and P. Verma, Predictability of Software Reliability Models, IEEE Transactions on Reliability, Volume 41(4), 539–546, December 1992.

[48] Y. K. Malaiya, A. von Mayrhauser and P. K. Srimani, An Examination of Fault Exposure Ratio, IEEE Transactions on Software Engineering, Volume 19(11), 1087-1094, November 1993.

[49] Y. K. Malaiya and J. Denton, Module Size Distribution and Defect Density, In Proceedings of the 7th IEEE International Symposium on Software Reliability Engineering, San Jose, CA, USA, 62-71, October 2000.

[50] Y. K. Malaiya and J. Denton, What Do the Software Reliability Growth Model Parameters Represent? Proceedings of International Symposium on Software Reliability Engineering, Albuquerque, NM, USA, 124-135, 1997.

[51] G. McGraw, From the Ground Up: The DIMACS Software Security Workshop, IEEE Security & Privacy, Volume 1(2), 59-66, March/April 2003.

[52] D. Moore, C. Shannon, and K. C. Claffy, Code-red: a case study on the spread and victims of an internet worm. In Proceedings of Internet Measurement Workshop, Marseilles, France, 273–284, November 2002.

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

[54] J. D. Musa and K. Okumoto, A Logarithmic Poisson Execution Time Model for Software Reliability Measurement, Proceedings of the 7th International Conference on Software Engineering, Sorrento, Italy, 230-238, May 1984.

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

[56] The MITRE Corporation, http://www.mitre.org, February 2005.

[57] R. Mohagheghi, R. Conradi, O. Killi and H. Schwarz, An Empirical Study of Software Reuse vs. Defect-Density, Proceedings of the 26th International Conference on Software Engineering, Edinburgh, Scotland, UK, 282-291, May 2004.

[58] A. Mockus, R. Fielding and J. Herbsleb, Two Case Studies of Open Source Software Development: Apache and Mozilla, ACM Transactions on Software Engineering and Methodology, Volume 11(3), 309-346, 2002.

[59] Netcraft, http://news.netcraft.com/, April 2006.

[60] D. M. Nicol, W. H. Sanders, K. S. Trivedi, Model-Based Evaluation: From Dependability to Security, IEEE Transaction on Dependable and Secure Computing, Volume 1(1), 48-65, January-March 2004.

[61] National Vulnerability Database, http://nvd.nist.gov, September 2006.

[62] O. S. Data, Windows 98, http://www.osdata.com/oses/win98.htm, March 2004.

[63] Ounce Labs, Security by the Numbers: The Need for Metrics in Application Security, http://www.ouncelabs.com/library.asp, 2004.

[64] A. Ozment and S. E. Schechter, Milk or Wine: Does Software Security Improve with Age? The 15th USENIX Security Symposium, Vancouver, BC, Canada, 93-104, July 2006.

[65] C. P. Pfleeger, Security in Computing, Prentice-Hall, 1997.

[66] I. Ray, S. Tideman: A Secure TCP Connection Migration Protocol to Enable the Survivability of Client-Server Applications under Malicious Attack. Journal Network Systems Management, 12(2), 251-276, June 2004.

[67] R. L. Rivest and A. Shamir and L. M. Adelman, A Method for Obtaining Digital Signature and Public-key Cryptosystems, MIT/LCS/TM-82, citeseer.ist.psu.edu/rivest78method.html. 1977.

[68] Red Hat Bugzilla, https://bugzilla.redhat.com/bugzilla, January 2005.

[69] E. Rescorla, Is finding security holes a good idea? Economics of Information Security, Volume 3(1), 14-19, January-February 2005.

[70] P. Rodrigues, Windows XP Beta 02. Only 106,500 Bugs, http://www.lowendmac.com/tf/010401pf. html, August 2001.

[71] E. E. Schultz, Jr., D. S. Brown and T. A. Longstaff, Responding to Computer Security Incidents, Lawrence Livermore National Laboratory,

ftp://ftp.cert.dfn.de/pub/docs/csir/ihg.ps.gz, July 23, 1990.

[72] M. Sahinoglu, Quantitative risk assessment for dependent vulnerabilities, Proceedings of 52nd Reliability and Maintainability Symposium, Newport Beach, CA, USA, 82-85, January 2006.

[73] R. Seacord, Secure Coding in C and C++, Addison Wisely, 2005.

[74] Secunia, http://secunia.com/, April 2006.

[75] Security Focus, http://www.securityfocus.com, 2006.

[76] C. Y. Shim, R. E. Gantenbien and S. Y. Shin, Developing a Probabilistic Security Measure Using a Software Reliability Model, INFORMATION: International Interdisciplinary Journal, Volume 4(4), 409-418, October 2001.

[77] S.-W. Woo, O. H. Alhazmi, and Y. K. Malaiya, Assessing Vulnerabilities in Apache and IIS HTTP Servers, Proceedings the 2nd IEEE International Symposium on Dependable Autonomic and Secure Computing, Indianapolis, IN, USA, 103-110, September 2006.

[78] S.-W. Woo, O. H. Alhazmi, and Y. K. Malaiya, An Analysis of the Vulnerability Discovery Process in Web Browsers, Proceedings of 10th IASTED International Conference on Software Engineering and Applications, Dallas, TX, USA, 172-177, November 2006.