DISSERTATION

QUANTITATIVE ANALYSES OF SOFTWARE VULNERABILITIES

Submitted by

HyunChul Joh

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Fall 2011

Doctoral Committee:

Advisor: Yashwant K. Malaiya

Indrajit Ray
Indrakshi Ray
Anura P. Jayasumana

ABSTRACT

QUANTITATIVE ANALYSES OF SOFTWARE VULNERABILITIES

There have been numerous studies addressing computer security and software vulnerability management. Most of the time, they have taken a qualitative perspective. In many other disciplines, quantitative analyses have been indispensable for performance assessment, metric measurement, functional evaluation, or statistical modeling.

Quantitative approaches can also help to improve software risk management by providing guidelines obtained by using actual data-driven analyses for optimal allocations of resources for security testing, scheduling, and development of security patches. Quantitative methods allow objective and more accurate estimates of future trends than qualitative manners only because a quantitative approach uses real datasets with statistical methods which have proven to be a very powerful prediction approach in several research fields.

A quantitative methodology makes it possible for end-users to assess the risks posed by vulnerabilities in software systems, and potential breaches without getting burdened by details of every individual vulnerability. At the moment, quantitative risk analysis in information security systems is still in its infancy stage. However, recently, researchers have started to explore various software vulnerability related attributes quantitatively as the vulnerability datasets have now become large enough for statistical analyses.

In this dissertation, quantitative analysis is presented dealing with i) modeling vulnerability discovery processes in major Web servers and browsers, ii) relationship between the performance of S-shaped vulnerability discovery models and the skew in vulnerability datasets examined, iii) linear vulnerability discovery trends in multi-version software systems, iv) periodic behavior in weekly exploitation and patching of vulnerabilities as well as long term vulnerability discovery process, and v) software security risk evaluation with respect to the vulnerability lifecycle and CVSS.

Results show good superior vulnerability discovery model fittings and reasonable prediction capabilities for both time-based and effort-based models for datasets from Web servers and browsers. Results also show that AML and Gamma distribution based models perform better than other S-shaped models with skewed left and right datasets respectively. We find that code sharing among the successive versions cause a linear discovery pattern. We establish that there are indeed long and short term periodic patterns in software vulnerability related activities which have been only vaguely recognized by the security researchers. Lastly, a framework for software security risk assessment is proposed which can allow a comparison of software systems in terms of the risk and potential approaches for optimization of remediation.

ACKNOWLEDGEMENTS

First and foremost, I would like to express the sincere appreciation to my outstanding advisor, Dr. Yashwant K. Malaiya, who has constantly supported me with his exceptionally superb insight and generous patience. Without the excellent research environment provided by Dr. Malaiya, I could have not completed my doctoral degree. In addition, I am thankful to my doctoral committee members, Dr. Indrajit Ray, Dr. Indrakshi Ray, and Dr. Anura P. Jayasumana, for agreeing to be on the board, reviewing my works, and their suggestions, criticism and support.

I would like to thank my mother MyungJa Kim, father SangHak Joh, elder sister JuEun Joh, and my brother-in-law Dr. HyunSoo Ahn in Korea. Their endless support and love, especially my mother's countless LONG phone calls for more than six years, keep me focus on the goal. Lastly, I thank to my best friend KyungDong Kim for always supporting me.

To my parents

TABLE OF CONTENTS

1 Introduction **1**
1.1 Motivation 1
1.2 Dissertation statement 4
1.3 Outline 6
1.4 Publication history 7

2 Software vulnerabilities **8**
2.1 Definition of a software vulnerability 8
2.2 Common vulnerability standards 11
2.2.1 Common Vulnerabilities and Exposures (CVE) 11
2.2.2 Common Weakness Enumeration (CWE) 12
2.2.3 Common Vulnerability Scoring System (CVSS) 14
2.3 Public online vulnerability databases 17
2.4 Actors in software vulnerability ecosystem 19

3 Vulnerability discovery model (VDM) **22**
3.1 Alhazmi-Malaiya Logistic model 25
3.2 Linear model 27
3.3 Alhazmi-Malaiya effort-based model 28
3.4 Factors influencing VDM 29

4 Vulnerability discovery process in Web servers **31**
4.1 Introduction 31
4.2 Aggregate vulnerabilities in HTTP server 35
4.3 Vulnerability categories 40
4.4 Vulnerability severity level 43
4.5 Predictive capability 47
4.6 Discussion 49
4.7 Summary 52

5 Modeling vulnerability discovery process in Web browsers **53**
5.1 Introduction 54
5.2 Related works and background 55
5.3 Preliminaries to modeling vulnerabilities in Web browsers 56
5.4 Aggregate vulnerabilities in Web browsers 59
5.5 Vulnerability severity levels 67

5.6 Prediction capability . . . 75
5.7 Discussion . . . 80
5.8 Summary . . . 82

6 Web browser secureness with respect to CVSS score 83
6.1 Comparing trends of grouped CVSS scores . . . 84
6.2 Exploitability and impact sub-metrics . . . 87
6.3 Combination of exploitability and impact . . . 90
6.4 Summary . . . 91

7 Modeling skewness in vulnerability discovery 94
7.1 Motivation . . . 95
7.2 Skewness . . . 96
7.3 S-shaped vulnerability discovery models . . . 99
7.3.1 Normal Distribution based model . . . 101
7.3.2 Gamma distribution based model . . . 102
7.3.3 Weibull distribution based model . . . 103
7.3.4 Beta distribution based model . . . 104
7.4 Goodness of fit analysis . . . 105
7.5 Prediction capabilities of S-shaped models . . . 109
7.6 Categorization by vulnerability type . . . 115
7.7 Summary . . . 119

8 Extended linear vulnerability discovery process 123
8.1 Motivation . . . 123
8.2 Possible causes for the extended linear phase . . . 125
8.3 Observations . . . 126
8.4 Prediction for the extended linear rate . . . 130
8.5 Things that influence on slope level . . . 135
8.6 Summary . . . 139

9 Seasonal behavior of vulnerability activities 141
9.1 Introduction . . . 142
9.2 Related work on seasonality . . . 143
9.3 Statistical methods for seasonality analysis . . . 146
9.4 Annual variations in vulnerability discovery processes . . . 149
9.4.1 Seasonal index analysis . . . 149
9.4.2 Autocorrelation function analysis . . . 153
9.5 Seven-day periodicity in the vulnerability scan data . . . 155
9.6 Possible factors causing periodic behavior . . . 158
9.7 Summary . . . 163

10 Defining and assessing quantitative security risk in software 165
10.1 Motivation 166
10.2 Related work on risk measurement 168
10.3 Risk matrix: scales and discretization 170
10.4 Possible improvement in CVSS metrics and related work 171
10.5 Defining conditional risk measures 175
10.6 Software vulnerability lifecycle 177
10.7 Evaluating lifecycle risk 180
10.8 Simulate lifecycle risk model 183
10.9 Measuring risk due to known unpatched vulnerabilities 185
10.10Summary 190

11 Conclusions 192
11.1 Modeling vulnerability discovery process 192
11.2 Periodic behavior in vulnerability activities 194
11.3 Software risk analysis 195
11.4 Future work 196

Bibliography 199

LIST OF FIGURES

1.1 Number of reported vulnerabilities (2003 ~ 2010) 2

2.1 Three CVSS metric groups 15
2.2 Major actors influencing software vulnerability ecosystem 19
2.3 Generalized vulnerability lifecycle; empirical findings 21

3.1 Vulnerability discovery model taxonomy. 23
3.2 Three phases in S-shaped AML model 25
3.3 Upper and lower boundaries for linear phase testing 27

4.1 Apache and IIS release timeline for major versions 32
4.2 Web server market share trends 37
4.3 Fitting Web server - aggregated vulnerability data 38
4.4 Categorizing Web server vulnerabilities by type 42
4.5 Fitting Web server - categorized by type 43
4.6 Fitting Web server - categorized by severity 45
4.7 Vulnerability proportions by severity 47
4.8 Prediction error 49

5.1 How to estimate the number of Web browser users 58
5.2 Estimating the number of users 59
5.3 Fitting; Time-based model to browser vulnerabilities 61
5.4 Fitting; AME Effort-based model to browser vulnerabilities 61
5.5 Shared vulnerabilities among Netscape, Mozilla and Firefox 62
5.6 Web browsers' severity variation 68
5.7 AML model fitting by severity - Web browsers 68
5.8 Linear model fitting by severity - Web browsers 73
5.9 AME model fitting by severity - Web browsers 73
5.10 Prediction errors - Web browsers 77
5.11 Average bias and average error - Web browsers 79

6.1 Grouped CVSS base scores 84
6.2 Grouped CVSS exploitability and impact sub-scores 86
6.3 Exploitability matric group - Web browser 89
6.4 Impact metric group - Web browser 90
6.5 Relationship between exploitability and impact - Web browser 91

7.1 Run chart for the number of vulnerabilities - eight S/W 97

7.2 Skewness: positive, negative and symmetrical . . . 98
7.3 Number of vulnerabilities grouped by six month - RHL . . . 99
7.4 The five probability density functions . . . 100
7.5 Model fitting for the five VDM models - eight S/W . . . 106
7.6 Prediction errors for the S-shaped models - eight S/W . . . 110
7.7 Average bias and average error - eight S/W . . . 112
7.8 Run chart for the number of vulnerabilities - by type . . . 116
7.9 Model fitting for the five VDM models - by type . . . 117
7.10 Prediction errors for the S-shaped models - by type . . . 120

8.1 S-shaped discovery process and extended linear phase . . . 124
8.2 Extended linear caused by shared vulnerabilities . . . 125
8.3 Extended linear caused by constant effort . . . 127
8.4 Linear vulnerability growth trends . . . 131
8.5 Linear vulnerability growth trends for unique vulnerabilities . . . 132
8.6 Estimated Max/Min slopes by AML . . . 134
8.7 Example matrix for referring developer and vendor levels . . . 137
8.8 Market shares for the six systems . . . 138
8.9 Estimated number of users . . . 138

9.1 Boxplots for seasonal behaviors . . . 142
9.2 Run chart for the number of vulnerabilities for the eighteen S/W . . . 144
9.3 Seasonal indices . . . 149
9.4 Run charts for Qualys datasets. . . . 157
9.5 Autocorrelation functions corresponding for Qualys datasets . . . 159
9.6 Seasonality possible reasons . . . 161
9.7 Frequency of disclosure, published, and data loss report dates . . . 161

10.1 Distributions for CVSS base metric scores . . . 172
10.2 Intuitive life cycle of a system-security vulnerability . . . 177
10.3 Possible vulnerability lifecycle journey . . . 178
10.4 Stochastic model for a single vulnerability . . . 181
10.5 Simplified single vulnerability lifecycle . . . 183
10.6 Simulated probability distributions in lifecycle . . . 185
10.7 Conceptual discovery & patch . . . 187
10.8 Evaluated risk gaps and normalized risk levels . . . 189

LIST OF TABLES

2.1 Vulnerability databases on the Web 18

4.1 Functional support comparison in Apache and IIS 33
4.2 Market share & vulnerabilities found 36
4.3 χ^2 goodness of fit test - Time-based 39
4.4 χ^2 goodness of fit test - Effort-based 39
4.5 χ^2 goodness of fit test results by type - Time-based 44
4.6 χ^2 goodness of fit test results by type - Effort-based 44
4.7 χ^2 goodness of fit test results by severity - Time-based 46
4.8 χ^2 goodness of fit test results by severity - Effort-based 46
4.9 Prediction error 48
4.10 Known D_{KD} vs. Known V_{KD} 50

5.1 Market share and vulnerabilities found - Web browsers 57
5.2 Phase test for aggregate vulnerability in Web browser 64
5.3 χ^2 goodness of fit test - AML - Web browsers 64
5.4 χ^2 goodness of fit test - LVD - Web browsers 64
5.5 χ^2 goodness of fit test - AME - Web browsers 64
5.6 χ^2 goodness of fit test by severity - AML - Web browsers 70
5.7 Phase test by severity in Web browser 70
5.8 χ^2 goodness of fit test by severity - LVD - Web browsers 74
5.9 χ^2 goodness of fit test by severity - AME - Web browsers 74
5.10 Initial parameter values for prediction 77
5.11 Average bias & average error - Web browsers 79

6.1 Primitive information about datasets used - Web browser 83
6.2 Grouped CVSS base scores - Web browser 85
6.3 Grouped CVSS exploitability sub-scores - Web browser 85
6.4 Grouped CVSS - impact sub-scores - Web browser 85
6.5 Exploitability metrics - Web browser 88
6.6 Impact metrics - Web browser 88
6.7 Top three high frequency combinations 92
6.8 Summary of CVSS version 1 and version 2 base scores 92
6.9 Most common CVSS version 2 vectors 92

7.1 Skewness and total number of vulnerabilities - eight S/W 96
7.2 Model parameters and χ^2 goodness of fit test 107
7.3 Average bias and average error 111

7.4 ANOVA table for the skewed negative datasets in AE . . . 114
7.5 ANOVA table for the skewed positive datasets in AE . . . 114
7.6 LSD for negatively skewed datasets in AE . . . 114
7.7 LSD for positively skewed datasets in AE . . . 114
7.8 Skewness and number of vulnerabilities by type I . . . 118
7.9 Skewness and number of vulnerabilities by type II . . . 118
7.10 ANOVA table for categorized OSX in AE . . . 119
7.11 ANOVA table for categorized IIS in AE . . . 119
7.12 LSD for categorized OSX in AE . . . 119

8.1 Software release date . . . 128
8.2 Shared number of vulnerabilities and percentages . . . 129
8.3 R^2 values from Figure 8.4 and 8.5 . . . 133
8.4 Estimated slopes in Figure 8.4 . . . 133
8.5 Information estimated by AML - Parameters . . . 135
8.6 Information estimated by AML - Transition points . . . 135

9.1 Number of vulnerabilities for the eighteen software systems . . . 142
9.2 Vulnerability discovery process seasonal indices. . . . 148
9.3 ANOVA table for seasonal indices from Windows OSes. . . . 151
9.4 ANOVA table for seasonal indices from non-Windows OSes. . . . 151
9.5 ANOVA table for seasonal indices from Web server/browser . . . 151
9.6 Seasonal LSD for Windows OSes . . . 152
9.7 Seasonal LSD for non-Windows OSes . . . 152
9.8 Seasonal LSD for non-Windows Web Server/Browser . . . 152
9.9 Individual ACF values for Windows OSes . . . 154
9.10 Individual ACF values for non-Windows OSes . . . 154
9.11 Individual ACF values for Web servers/browsers . . . 154
9.12 Weekly Seasonal index values . . . 160
9.13 χ^2 test for weekly seasonal index values . . . 160

10.1 Stem states and next possible hops in vulnerability lifecycle . . . 179
10.2 Average patch time . . . 188
10.3 Simulated patch dates . . . 188

Chapter 1

INTRODUCTION

1.1 Motivation

The dependence of human society on the Internet is growing. Online banking, stock market trading, automated military and governmental facilities depend heavily on the Internet based computing. As a result, the risk to the society from the potential exploitation of vulnerabilities in automated systems is enormous. As a consequence, these days, security and privacy issues caused by software vulnerabilities have been under intense scrutiny.

In spite of the risks, people are willing to take the risk of using the Internet since it has created enormous virtual markets and made the transactions much more efficient (Scoy, 1992). In spite of the recent advances in secure coding, it is unlikely that completely safe systems will become possible anytime soon (Farrell, 2010). Thus, it is necessary to accept a measured degree of risk and precautionary measures commensurate.

As development of the software systems is getting complex, the number of defects in software systems has been increased. Consequently, the number of security related defects is also continually growing in both open source and proprietary systems. Figure 1.1 shows the number of vulnerabilities reported from year 2003 to 2010 grouped by their severities. Although the number seems to be stabilized after 2008 due to the recognition of the Internet security crisis and the emergence of secure coding practices, still the notification rate represents a severe security threat.

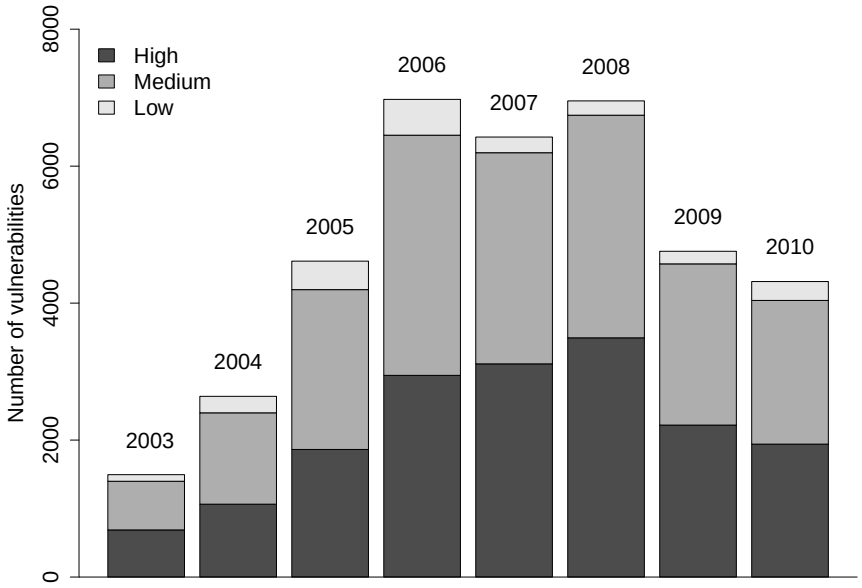


Figure 1.1: Number of reported vulnerabilities (year 2003 ~ 2010) from NVD (nvd.nist.gov) on March 2011

Like Lord Kelvin (1824 ~ 1907) said in his famous quote of "If you cannot measure it, you cannot improve it", somehow software security researchers need to be able to measure the secureness in software systems to address the threat. Secureness in a software system can be understood as a degree of the lact of high severity security related software defects that can be potentially exploited.

In short, software vulnerabilities can be defined as software defects or weaknesses in the security system which might be exploited by malicious users causing loss or harm (Pfleeger and Pfleeger, 2003) (we will see the definition in detail at Chapter 2.1). It is next to impossible to eliminate them completely. Those vulnerabilities are great concern since they provide attackers the ability to gain full control of the system or leakage of highly sensitive information.

Since security vulnerabilities are a class of software defects which can lead to security violations, the vulnerabilities have some of the attributes of the parent class, the ordinary defects. However, there are some differences between the two (Anbalagan and Vouk, 2008; Zimmermann et al., 2010). Ordinary defects are often not fixed until the next version release because they generally do not represent the

high degree of risk. On the other hand, due to the great risks caused by vulnerabilities, the system developers need to release patches as soon as possible after a vulnerability is discovered. Zimmermann et al. (2010) have empirically found that the traditional software reliability growth models may not be able to handle the newly rising security related defects properly.

Even though an ethical question has been arisen in security vulnerability research (Matwyshyn et al., 2010) regarding potential misuse of the information, nowadays, IT related vulnerabilities are systemically recorded using several security vulnerability standards, and they are publically available. One of the standards is Common Vulnerabilities and Exposures[1] (CVE) Identifiers. The CVE Identifiers are commonly used by information security related product or service vendors and researchers as a standard method for identifying vulnerabilities and for cross-linking with other repositories that also use CVE Identifiers. Moreover, public online vulnerability databases are maintained by governmental organizations, open communities or private companies. Some of the examples are National Vulnerability Database (NVD)[2], Open Source Vulnerability Database[3], US-CERT[4], Secunia[5], etc.

During software development process cycles, managers and developers like to know whether their products meet certain requirements before the releasing date. Even after the date, they would like to be informed about the defects discovered after release to provide quick after-sale service. These include: why and what kind of vulnerabilities tend to be found, who are the people finding them, how they can be fixed, and when would be the best time to announce them publicly. If the software development managers can make accurate predictions of vulnerability discovery trends and related activities, they can optimally allocate the needed resources that are likely to be needed for patch development. In the dissertation, we

[1] http://cve.mitre.org/
[2] http://nvd.nist.gov/
[3] http://osvdb.org/
[4] http://www.kb.cert.org/vuls/
[5] http://secunia.com/advisories/historic/

have quantitatively investigated some of the software vulnerability related processes in detail.

1.2 Dissertation statement

A better understanding of the quantitative characteristics of software vulnerabilities allows for the IT security practitioners with better insight into their works. Not only the security domain experts but also general end-users would potentially benefit from the better quantitative perspective since they can take actions before the risk reaches an unacceptable level.

Being a new research area, the quantitative aspects of software vulnerabilities and risk assessments have not been fully investigated. Only a few major topics, such as modeling vulnerability discovery process, have been examined. Further detailed studies are required related to the security risk assessment, using rigorous analysis of actual data which can assist decision makers to maximize the returns on their security related efforts.

To investigate security, both qualitative and quantitative methods are needed. Generally, qualitative analysis deals with descriptions and data which can be observed but are not measurable. On the other hands, quantitative analysis deals with numbers and data which can be measured, and usually followed by statistical tests.

Frequently, in qualitative risk management, each risk has accompanied by the attributes like severity level, impact score, mitigation plans, etc. The approaches considered are based on heuristics and the perception of experts[6]. As a result, qualitative analysis heavily depends on the experts' opinions which tend to be subjective.

In comparison with qualitative methods, quantitative analyses allow evaluations to be more precise when considering how much effort is needed to protect the system or how high the system exploitation risk is when a security policy in in place (Vache,

[6] `http://www.intaver.com/Articles/Article_QuantitativeRiskAnalysis.pdf`

2009). Until recently, most of the existing discussion on software quality has been limited to qualitative discussions (Gousios et al., 2007). However, an actual data-driven empirical analysis, followed by some statistical tests to make a final decision, provides objective results, which seems to be a more proper technique for answering questions regarding to security.

Some (Jones, 2007) say that a qualitative method is more like an art which does not depend on predefined definitions whereas quantitative method can be referred as a science which based on clear predefined definitions.

At the moment, quantitative risk analysis in information systems is considered too difficult to conduct due to a shortage of data and the costs associated with collecting and analyzing datasets. A study by Ford et al. (2005) compares several software venders by considering the number of vulnerabilities and severity, and suggests a need to use quantitative approaches for estimating the risks posed by vulnerabilities.

Recently, however, several software vulnerability datasets have become available at publicly available databases, and researchers have started to examine characteristics of software vulnerabilities quantitatively. The examples include dependencies among vulnerabilities (Neuhaus and Zimmermann, 2009; Sahinoglu, 2006; Cukier and Panjwani, 2009), vulnerability lifespan (Arbaugh et al., 2000; McQueen et al., 2009), optimal security patching timing (Beattie et al., 2002), optimal vulnerability announcement timing (Arora et al., 2008), vulnerability discovery models (Condon et al., 2008; Kim et al., 2007; Alhazmi et al., 2007; Chen et al., 2010), security vulnerability metrics (Scarfone and Mell, 2009; Liu and Zhang, 2010).

Possible approaches for a quantitative perspective of exploitation trends are discussed by Hallaraker and Vigna (2005). Probabilistic examinations of intrusions have been presented by several researchers (Browne et al., 2001; Madan et al., 2004). Rescorla (2005) has studied vulnerabilities in open source servers. The vulnerability

discovery process in operating systems has recently been examined by Rescorla (2003), Alhazmi and Malaiya (2005), Alhazmi et al. (2005), Alhazmi and Malaiya (2008), Kim et al. (2007), and Chen et al. (2010).

In this dissertation, we have explored quantitative software vulnerability analyses focussing mainly on vulnerability discovery process, periodic behavior in vulnerability activities, and software risk assessment based on the attributes of the vulnerabilities. The results from the research will help to develop methods for better understanding of software vulnerabilities in an effective manner.

1.3 Outline

The dissertation is organized as follows. In the next chapter, Chapter 2, background information about the software vulnerability research area is presented. Chapter 3 introduces some of the vulnerability discovery models in detail which are mainly utilized in the dissertation. Chapter 4 and 5 investigate the applicability of time-based and effort-based vulnerability discovery models for the two most popular Web servers and the six well known Web browsers by examining the model fitting and prediction capabilities. For the browsers, we have investigated application of the linear vulnerability discovery model due to the systems' continued evolutions. In addition, in Chapter 6, risk assessment based on CVSS base scores from the four popular Web browsers is examined quantitatively.

In Chapter 7, several new S-shaped vulnerability discovery models are introduced to examine the relationship between performance of the S-shaped vulnerability discovery models with the skewness in target vulnerability datasets. In Chapter 8, linear vulnerability discovery trend, which is often observed among some of the popular software systems, has been investigated.

Chapter 9 examines periodic behavior for long term and short term vulnerability activities based on vulnerability discovery information in popular software systems

and globally scanned data collected by one of the leading providers of on-demand IT security risk and compliance management. In Chapter 10, a novel approach for evaluating the software security risk based on vulnerability lifecycle and CVSS metrics for given systems is presented. Since this research topic has just been started, the model is in its preliminary form. Finally, Chapter 11 concludes the dissertation with future directions of the research.

1.4 Publication history

Much of the material in this dissertation has previously been presented at peer reviewed workshop (Joh and Malaiya, 2010a), conferences (Joh and Malaiya, 2008a,b, 2009, 2010b; Joh et al., 2010; Joh and Malaiya, 2011a), and journal article (Woo et al., 2011a). Also, some of the materials in the dissertation are currently under the review (Joh and Malaiya, 2011b; Woo et al., 2011b) or in preparation (Joh and Malaiya, 2011c), in all journal versions. I also coauthored (Younis et al., 2011). All the publications mentioned above are under the supervision of Dr. Malaiya.

Chapter 2

SOFTWARE VULNERABILITIES

2.1 Definition of a software vulnerability

Due to the lack of widely accepted standards and definitions in the information technology research area, researchers in the field are frequently confused while doing peer reviews. Since this dissertation is all about the software vulnerability related materials, first, we are trying to define the word vulnerability.

The software vulnerability is a subset of vulnerability in general, so the software vulnerability should inherit the characteristics what the general vulnerability has. According to the Collins English dictionary [1], "Someone who is vulnerable is weak and without protection, with the result that they are easily hurt physically or emotionally." In other words, it represents a susceptibility to malevolent manipulations.

Even though the concept is crystal clear, it is not that simple to define the software vulnerability due to the lack of standards in the field; there's no widely accepted definition for the word currently (Krsul, 1998; Ozment, 2007a). Yet, there are many definitions proposed, and here are some of them:

- "A flaw in a product that makes it infeasible – even when using the product properly – to prevent an attacker from usurping privileges on the user's system, regulating its operation, compromising data on it, or assuming ungranted trust.[2]"

[1] http://www.collinslanguage.com/

[2] http://technet.microsoft.com/en-us/library/cc751383.aspx

- "Security flaws, defects, or mistakes in software that can be directly used by a hacker to gain access to a system or network" (Wang et al., 2009)

- "Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source" (NIST, 2006)

- "Weakness in the security system which might be exploited by malicious users causing loss or harm" (Pfleeger and Pfleeger, 2003)

- "A vulnerable system is an authorized state from which an unauthorized state can be reached using authorized state transitions; a vulnerability is a characterization of a vulnerable state which distinguishes it from all non-vulnerable states" (Cheswick and Bellovin, 1994)

- "Defect which enables an attacker to bypass security measures" (E. E. Schultz et al., 1990)

So far, there are not many literatures discussing the terminology in depth. As one of the early works, Otwell and Aldridge (Otwell and Aldridge, 1989) examined the treatment of vulnerability at the 1988 Risk Model Builders' Workshop. They say that defining the word of vulnerability formally is proven to be a complex task while showing the several proposed definitions from the researchers in the workshop. Some of them are:

- "Weaknesses that allow a threat to compromise the security (confidentiality, integrity, or availability) of an asset." (Mayerfeld, 1988)

- "Achievable bad events," which "implies that the protections against them are nonexistent, insufficient, or insufficiently protected." (Lewis, 1988)

- "The ability of an agent to cause an attack event" (Snow, 1988)

A decade later, in 1998, Krsul (Krsul, 1998) defined the software vulnerability in his doctoral dissertation as "an instance of an error in the specification, development, or configuration of software such that its execution can violate the security policy." And another decade later, in 2007, Ozment supports the Krsul's definition with some minor modification; after the modification, the definition is read as "an instance of [a mistake] in the specification, development, or configuration of software such that its execution can violate the [explicit or implicit] security policy" (Ozment, 2007a). Ozment made two changes. The first one is that the "mistake" is used instead of the "error" since in software engineering, an "error" is already defined as "the amount by which the result is incorrect" (IEEE, 1990). The second is that he put the "explicit or implicit" in the modified definition to emphasize the fact that all systems have a security policy whether it is explicit or not.

Meanwhile, after showing the definitions, Otwell and Aldridge (Otwell and Aldridge, 1989) stated that it is clear that all the researchers have the same general conception of vulnerability and differ mainly how vulnerabilities of a particular system are specified and measured, and also clear that "more vulnerable" means "easier to adversely affect" and "less vulnerable" is better, other things being equal.

In this dissertation, we follow the definition from CVE[3]: *an information security "vulnerability" is a mistake in software that can be directly used by a hacker to gain access to a system or network.* Just like what Frei stated in his dissertation (Frei, 2009), we also only consider vulnerabilities listed in the CVE directories. Hence, it does make sense for this dissertation to use the definition from CVE since all the vulnerability datasets used in this dissertation have CVE identification numbers. CVE will be discussed in the following section.

[3] `http://cve.mitre.org/about/terminology.html`

2.2 Common vulnerability standards

There are several commonly used vulnerability standards by researchers to make vulnerability measurable. In this section, three popular vulnerability standards to vulnerability researchers are introduced: CVE, CWE, and CVSS.

2.2.1 Common Vulnerabilities and Exposures

Common Vulnerabilities and Exposures (CVE) is a publicly available and free to use list or dictionary of standardized identifiers for common computer vulnerabilities and exposures [4]. It is a dictionary of publicly known information of security vulnerabilities and exposures. CVE's common identifiers enable data exchange between security products and provide a baseline index point for evaluating coverage of tools and services.

CVE was launched in 1999 when most information security tools used their own databases with their own names for security vulnerabilities which make hard to communicate among the security vendors and security advisories. CVE's standardized identifiers enable to solve this problems. Currently, CVE is treated as *de facto* industry standard for vulnerability and exposure names. Originally, in 1999, there were 321 CVE entries, and now there are more than 46,000 CVE entries as of June 2011. Each CVE identifier includes:

- CVE identifier number (i.e., "CVE-2010-0034").
- Indication of "entry" or "candidate" status.
- Description of the security vulnerability or exposure.
- pertinent references (i.e., vulnerability reports, mailing list postings and advisories).

[4] `http://cve.mitre.org/cve/`

Not all the discovered vulnerabilities receive CVE entry position automatically from the start. After the discovery, the information is assigned a CVE identifier with "candidate" status by a CVE Candidate Numbering Authority (CNA), and proposed to the CVE editorial board by the CVE editor. The board talks over the CNAs and votes on whether it should become a CVE entry. If the candidate is accepted, its status is updated to "entry" on the CVE list. If not, the reason for rejection is noted in the editorial board archives posted on the CVE Web site.

The assignment of a candidate number is not a guarantee that it will become an official CVE entry. Usually, it takes one day to one month to assign a candidate number. Then it takes another a year or more for the candidate to become an official CVE entry. In some cases, it takes much longer due to the obscure or insufficient issues, or unstabilized CVE editorial policies. All of the datasets speculated in the dissertation are from NVD database which is based upon and synchronized with the identifiers on the CVE List. Therefore, all the vulnerabilities in the dissertation are assigned with CVE identifiers.

2.2.2 Common Weakness Enumeration

Common Weakness Enumeration[5] (CWE) is a list of software weakness types, and is sponsored by the National Cyber Security Division in the US Department of Homeland Security. It aims to be a complete dictionary for software weaknesses. It provides a unified, measurable set of software weaknesses that is enabling more effective discussion, description, selection, and use of software security tools and services that can find these weaknesses in source code and operational systems as well as better understanding and management of software weaknesses related to architecture and design. In short, a unique number is assigned to each weakness type. Since CWE provides fine detail classifications, the CWE Web site contains

[5] `http://cwe.mitre.org/`

the information more than 860 programming, design, and architecture error types that can lead to exploitable vulnerabilities as of June 2011.

Each year, MITRE[6] and SANS[7] issue the top 25 most dangerous programming errors[8] which is a list of the most widespread and critical programming errors that can lead to serious software vulnerabilities. Since they are easy to exploit and let hackers to gain a complete control over a system, the errors are treated as very dangerous weaknesses. Howard (2009) gives each vulnerability's characteristics from the 25 CWE list and some advises to improve software security by eliminating the top 25 vulnerability types. The study is based on the top 25 list in 2009.

Recently, CWE has been used to calculate a security related metrics. For example, Wang et al. (2010) try to rank attack patterns for a given software system systematically by mainly analyzing CAPEC[9] (Common Attack Pattern Enumeration and Classification) connected to the 14 types of CWE. The authors get the vulnerability information from OVM (Wang and Guo, 2009) (Ontology for vulnerability management) which is populated with vulnerabilities in NVD, and NVD uses 19 CWE categories. Since five of 19 are not able to be mapped to the attack patterns of CAPEC, it is 14. First, information about products' vulnerabilities is classified based on the 14 categories. Then calculate a weight for each type of vulnerabilities based on the three factors: time (recently discovered vulnerability has bigger weight), frequency (the number of vulnerabilities that reside in a type), and severity (CVSS score). Finally, the attack patterns are ranked based on their weights calculated with an equation they created.

Other than CWE, there are some of the other different vulnerability classifications, although none of them is recognized as a standard. Huang et al. (2010) classified vulnerabilities from the NVD into 45 main clusters. They used a novel

[6] http://www.mitre.org/
[7] http://www.sans.org/
[8] http://cwe.mitre.org/top25/
[9] http://capec.mitre.org/

method for the classification. They use text mining tools to cluster vulnerability description patterns to avoid overlaps and create a relatively objective classification criterion among the vulnerabilities. One doubtful statement they made is that the reason why the number of discovered vulnerabilities getting decreased from year 2006 is because of strengthening network security. However, some researchers say that the reason is just due to the back logging in the CVE and NVD approval process. Meanwhile, Jin et al. (2009) list 25 vulnerability classification methods with very brief comparisons among them. Even though the paper is titled as "A Review of Classification Methods for Network Vulnerability," the paper handles vulnerability classification in general, not for only network related one.

Since around September 2007, National Vulnerability Database have provided the selected 19 CWE names in the vulnerability database instead of the eight categories (Woo et al., 2011a) used before. We found because of that, in NVD dataset, an information disconnection has been happened for the vulnerability types before and after the time point.

Recently, Neuhaus and Zimmermann (Neuhaus and Zimmermann, 2010) semi-automatically analyzed the description text part from 39,393 CVEs by using Latent Dirichlet Allocation (LDA) to categorize the type of vulnerabilities called a topic model. The motivation of the study is that currently NVD is using 19 CWE categories even though there are several hundreds of CWE names. The reason is there are simply too many categories. Problems derived from this could be loss of information and many early CVE entries do not have CWE classification at all. Therefore, to analyze the trends for the entire CVE data, a new classification system is needed. The proposed methodology categorizing the CVE vulnerabilities might overcome the problem of disconnection of category information pointed out above.

2.2.3 Common Vulnerability Scoring System

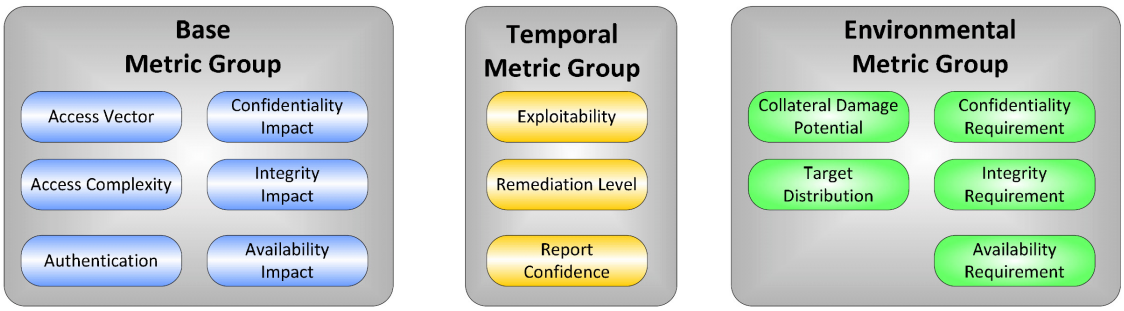


Figure 2.1: Three CVSS metric groups (Mell et al., 2007)

In July 2003, National Infrastructure Advisory Council (NIAC) commissioned a project to address the problem of multiple and incompatible IT related vulnerability scoring systems. As a result, the Common Vulnerability Scoring System (CVSS) has been adopted by many vendors since its first launch in 2004 such as application vendors, vulnerability scanning and compliance tools, risk assessment products, security bulletins, and academics (Stango et al., 2009; Mkpong-Ruffin et al., 2007; Houmb et al., 2010). The CVSS scores for known vulnerabilities are readily available on the majority of public vulnerability databases on the Web. The CVSS score system provides vendor independent framework for communicating the characteristics and impacts of the known vulnerabilities. Security analysts do not need to think about qualitative evaluation of vulnerability severity when they estimate it with the CVSS metric because it designed to be quantitative method in the final scores in each vulnerability.

The scoring system is now on its second version which is finalized its design in June 2007, and currently maintained by CVSS Special Interest Group (CVSS-SIG) at Forum of Incident Response and Security Teams (FIRST)[10]. The CVSS is composed of three metric groups: base, temporal and environmental as shown in Figure 2.1. It attempts to evaluate the degree of risks posed by vulnerabilities, so mitigation efforts can be prioritized. The score is range of [0.0, 10.0]; scores close to 0.0 indicates more stable whereas scores close to 10.0 means more vulnerable to

[10] `http://www.first.org/`

exploitation and causes more serious outcome. Here, CVSS is explained a bit more detail than other standards since it is used in Chapter 6 and 10 significantly.

The base metric group, ranges of [0.0, 10.0], represents the intrinsic and fundamental characteristics of a vulnerability, so the score is not changed over time. The base metric has two sub-scores of exploitability and impact sub-scores. The two sub-scores are also ranges of [0.0, 10.0]. The exploitability sub-score captures how a vulnerability is accessed and whether or not extra conditions are required to exploit it while the impact sub-score measures how a vulnerability will directly affect an IT asset as the degree of losses in confidentiality, integrity, and availability.

The exploitability sub-score is composed by three elements of access vector (AV), access complexity (AC), and authentication (Au). The access vector reflects how the vulnerability is exploited in terms of local (L), adjacent network (A), or network (N). The access complexity measures the complexity of the attack required to exploit the vulnerability once an attacker has gained access to the target system in terms of High (H), Medium (M), or Low (L). The authentication counts the number of times an attacker must authenticate to a target in order to exploit a vulnerability in terms of Multiple (M), Single (S), or None (N).

On the other hand, the impact sub-score is composed by the three key aspects in information security components: confidentiality, integrity and availability. The impact attributes are all assessed in terms of None (N), Partial (P), or Complete (C).

Before CVSS scores are entered into NVD, security experts analyze the vulnerabilities and assign one of the qualitative letter grades mentioned above on the vulnerabilities (Houmb et al., 2010). Since the central goal of CVSS is producing comparable vulnerability scores, analyzers are allowed to rate the vulnerabilities only with those letters. Finally, scoring is the process of combining all the metric values according to the specific formulas from (Mell et al., 2007).

The temporal metric group, ranges of [0.0, 10.0], is measured dynamically in terms of exploitability (E), remediation level (RL), and report confidence (RC). Exploitability measures the current state of exploit techniques or code availability, and is evaluated in terms of unproved (U), proof-of-concept (POC), Functional (F), High (H), or Not Defined (ND). The remediation level refers to the type of remediation available for the vulnerability in terms of Official Fix (OF), Temporary Fix (TF), Workaround (W), Unavailable (U), or Not Defined (ND). The report confidence attribute refers to the confidence in the existence of the vulnerability and the credibility of the known technical details, and is evaluated in terms of Unconfirmed (UC), Uncorroborated (UR), Confirmed (C), or Not Defined (ND). The levels of exploitability will be positively affected by the three factors.

The environmental metric group, ranges of [0.0, 10.0], measures the characteristics of a vulnerability that are associated with an user's IT environment: all related to the system environment and the stakeholders' values. It is measured in terms of collateral damage potential (CDP), target distribution (TD), and security requirements (SR). The collateral damage potential measures the potential damage to life and the loss of value for physical assets. The possible values for this metric are None (N), Low (L), Low-Medium (LM), Medium-high (MH), High (H), and Not Defined (ND). The target distribution measures the proportion of vulnerable systems, and is evaluated in terms of None (N), Low (L), Medium (M), High (H), or Not Defined (ND). The security requirements attributes enable the analyst to customize the CVSS score depending on the importance of the affected IT asset to a user's organization, measured in terms of confidentiality, integrity, and availability; each is measured as Low (L), Medium (M), High (H), or Not Defined (ND).

2.3 Public online vulnerability databases

One of the first thing to do for the quantitative vulnerability research is collecting the datasets to be analyzed. Fortunately, there are many publically available

Table 2.1: Vulnerability databases on the Web

Vulnerability Database	URL
National Vulnerability Database (NVD)	http://nvd.nist.gov/
Open Source Vulnerability Database (OSVDB)	http://osvdb.org/
IBM Internet Security Systems (X-Force)	http://xforce.iss.net/
CVE Details	http://www.cvedetails.com/
Security Lab by Positive Technologies	http://en.securitylab.ru/
DragonSoft Vulnerability Database	http://vdb.dragonsoft.com/
US-CERT Vulnerability Notes Database (CERT)	http://www.kb.cert.org/vuls
French Security Incident Response Team (FrSIRT)	http://www.vupen.com/english/
Secunia	http://secunia.com/
VUPEN	http://www.vupen.com/english/

vulnerability databases on the Web. Table 2.1 shows some of them. Usually, they are overlap and complement each other, so there is no the best source. Many of them provide CVE identifiers, severity levels, CVSS scores, and published date. If it is available, they also provide vulnerability patch date, discovery date, vulnerability type, etc.

There are more than 100 vulnerability databases and security advisories on the Web. Some of them are freely available, others are not. Some of them are managed by governments and others are run by private security companies or open security communities.

Massacci and Nguyen (Massacci and Nguyen, 2010) tried to answer the questions of how good researchers are at sampling for their quantitative vulnerability analyses. They compare 14 representative vulnerability databases showing what kind of information is available, and introduced 26 papers studying about quantitative vulnerability analyses with respect to the what kind of vulnerability databases the papers used. To compare and verify the number of vulnerabilities for Mozilla Firefox (Version 1.0, 1.5, 2.0 and 3.0), they created their own vulnerability database based on multiple vulnerability databases from the Web. The comparison is made between the number of vulnerabilities from their database and Mozilla Foundation Security Advisories (MFSA). The differences are about 68% for version 1.0 to

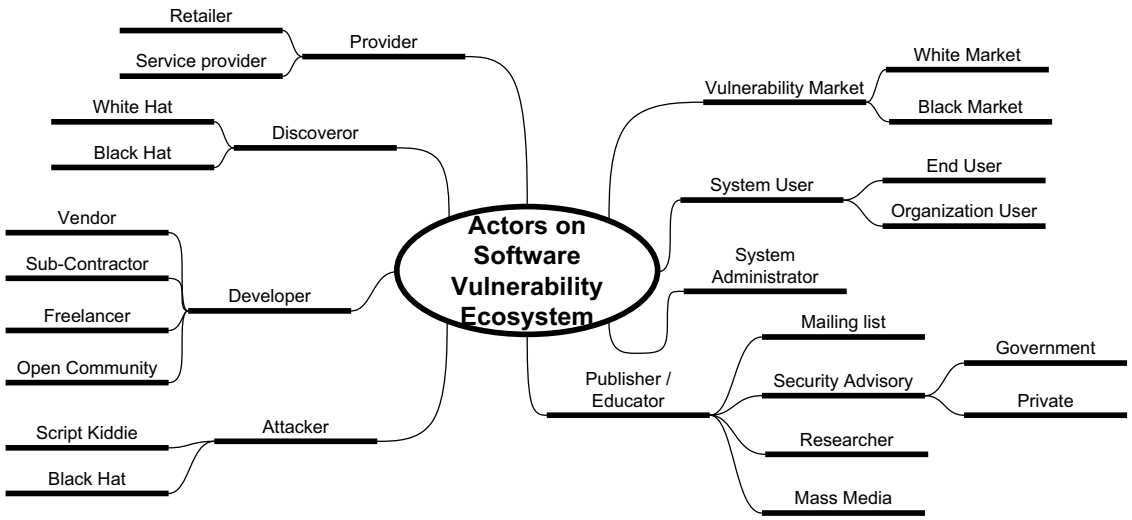


Figure 2.2: Major actors influencing software vulnerability ecosystem

2.0, and 20% for version 3.0. They concluded that prediction models using MFSA would miss 68% of total vulnerabilities in the end, called *Vulnerability missing phenomenon.*

Liu and Zhang (2010) compares three vulnerability databases of X-force, Vupen, and NVD, and concluded that the existing vulnerability rating systems are not consistent with the normal distribution. As a result, they proposed a new system for rating and scoring vulnerabilities called Vulnerability Rating and Scoring System (VRSS) which taking into account many kinds of existing vulnerability rating systems.

Meanwhile, in his Ph.D. dissertation (Frei, 2009), Frei compares several SIPs over several aspects in terms of history of information, organization location, number of publications, and disclosure performances. He refers the vulnerability database providers as Security Information Providers (SIP).

2.4 Actors in software vulnerability ecosystem

As Figure 2.2 shows, there are many players in the software vulnerability ecosystem. Developers are the creators of security vulnerabilities in software systems.

They could be a commercial or governmental vendor, sub-contractor, freelancer, or open source community. In general, unsafe or careless programming behaviors cause the security defects.

There are largely two types of discoverer: white hat and black hat. When white hats discover the security vulnerability, they follow the responsible disclosure practice, which usually means a full disclosure under the all stakeholders' agreement during a period of time for developing patches on the vulnerability before publishing the details. On the other hand, if black hats detect the vulnerability, they use the information for their own goods. Meanwhile, Ozment (2007b) divided the discoverer into the three parts: vendor detector, external detector, and accidental detector.

Radianti et al. (2009) empirically shows that there is indeed vulnerability black markets along with the white markets run by security companies such as Tipping-Point [11]. Whether it is black or white, the markets gives motivations and incentives for the vulnerability hunters.

Many commercial software vendors directly sell their products online, but more often retailers and service providers do business for the software producers. The product buyers could be home users or organizations. For the home users, they need to install the products and patches in their systems by themselves whereas, in organizations, usually specialized administrators do the jobs.

Administrators' role is very important for defending efficiently the systems against malicious users and attackers. They need to decide when to install the newly released security updates because some patches or updates causes problems which not exist before, yet if it is too late, the system is helpless for the attacks. Beattie et al. (2002) have examined 136 CVE entries, and they found that 92 patches were good patches, 20 were revised or pulled patches, and 24 had no patches.

The terminology of "script kiddie" is frequently used to distinguish from "black hat" who is able to create a hacking tools and able to analyze target system's security

[11] `http://www.eweek.com/c/a/Security/Price-War-iDefense-Doubles-Bounty-for-Security-Flaws/`

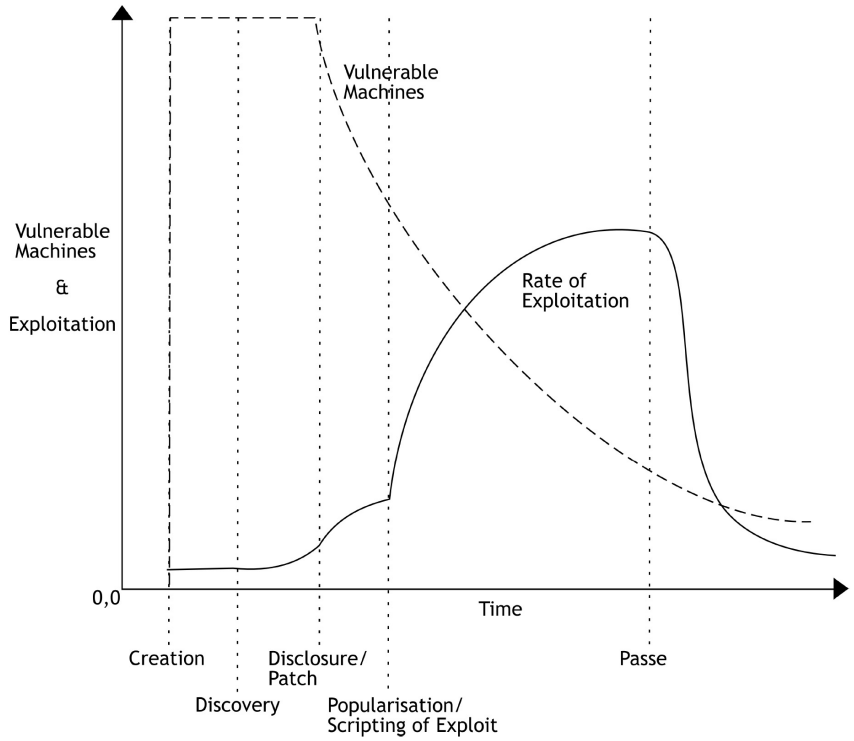


Figure 2.3: Generalized Model of Empirical Findings (White, 2006)

holes. The script kiddie uses scripts or programs made by other skilled hackers to attack computer systems and networks.

Even though script kiddies' skills are not mature enough to create neither their own hacking tools nor analyzing software systems to find critical security holes, they are increasingly recognized as a big concern in security community. In his master's thesis, White (2006) demonstrated the relationship between the number of exploited machines and vulnerability lifecycle as shown in Figure 2.3. White's lifecycle clearly tells that right after the scrip is available the number of exploited systems is dramatically increasing.

Distributing the detail information to the public about the discovered vulnerability under the responsible disclosure agreement is role of software vendors, security advisories, mass media, and researchers. At the same time, they educate users about the seriousness of the vulnerabilities and how to handle with the problems.

Chapter 3

VULNERABILITY DISCOVERY MODEL (VDM)

Like Yogi Berra said "Prediction is very hard, especially about the future", calculating what will be happening in the future is not an easy task, even for a near future. In weather forecast, we can clearly observe how the prediction is difficult. With all those tremendous calculating powers, supercomputers frequently make mistakes in the forecasting. Rather, old people's foresight, based on their body condition, whether it will be rainy or not, often more accurate than the forecast from the broadcast: I remember that my father said it will rain because his knee was aching.

The reason why the luxurious computers make mistakes is mainly because meteorologists are not able to feed the sufficient information or parameters for the machines to predict the future accurate enough, not to even mention about the butterfly effect. It sounds not feasible to provide all the possibly related factors to the machines for always accurate predictions. However, in some degree, meteorologists deliver reasonable forecasting. The benefit from accurate weather forecasting is obvious. We could schedule events affected by weather conditions in advance which could prevent from wasting the time, effort and money.

Just like the weather forecast, when we can estimate the vulnerability discovery trend in advance in some degree, let alone accurately, people in IT industry can allocate their resources optimally such as software deployment, patch management,

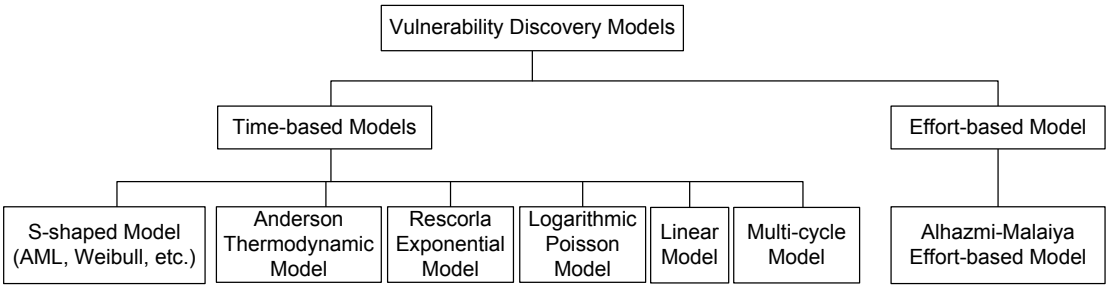


Figure 3.1: Vulnerability discovery model taxonomy.

and risk assessment. Even for end-users could take some advantages from it by assessing how vulnerable their systems are, so that they could take actions before something bad happens.

Use of quantitative reliability growth models is now common in the software reliability engineering (Musa, 1999; Lyu, 1995). Software reliability growth models (SRGM) project bug found-and-remove process: as bugs are found and removed, fewer bugs remain. 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.

Vulnerability discovery models allow prediction of the number of vulnerabilities that are likely to be discovered in the future. Hence, they allow the vendors and the end users to manage risk by optimizing resource allocation. Most vulnerability discovery models proposed so far use the calendar time as an independent variable. Effort based modeling has also been proposed, which requires the use of market share data.

In general, model derivations assume that a software system is stable. In other words, injecting a new chunk of source code is not expected. However, in reality, models are being applied to software that has evolved for several years. Software evolution is the process that describes a gradually changing software system. It has been suggested that the software evolution affects on growth shapes of the proposed

time-based models since new vulnerabilities may get introduced along with the new version of software in the process of evolution. Kim (2007) has addressed the impact of software evolution and has suggested that the S-shaped model may still apply while the significance of parameters may change.

Researchers have proposed software vulnerability discovery models (VDMs) (Alhazmi and Malaiya, 2005) to project the current trend and to estimate the discovery process since vulnerability datasets have been publically available from early 2000s. A few VDMs have been proposed by researchers which include Anderson (2002), Rescorla (2003), Alhazmi and Malaiya (2008), Ozment and Schechter (2006), Kim et al. (2007), Joh and Malaiya (2010b) and Chen et al. (2010). Figure 3.1 shows classification of vulnerability discovery models. Vulnerability discovery models are separated into time-based and effort-based models. The time-based models use calendar time as the independent variable factor and the effort-based model uses the number of installed system or the number of system users as the main factor. These models incorporate the effect of the rising and declining market share on the software. Each model uses a different approach with different assumptions, and parameters which restrict their performances. As a result, the VDMs frequently predict different vulnerability discovery rates using the same datasets.

Recently, Vulnerability Discovery Models considering about sudden increases caused by successive versions during vulnerability discovery process have been proposed (Kim et al., 2007). In their study, Kim et al. (2007) proposed a new approach for modeling the vulnerability discovery process based on shared source code measurements among multi-version software systems, and verified their theory by examining two open source software systems. Chen et al. (2010) presents a multi-cycle vulnerability discovery model utilizing a sinusoidal function. The model illustrates the relationship between the number of vulnerabilities and their release time, and is compared with other VDMs based on the datasets from eight versions of Windows operating systems.

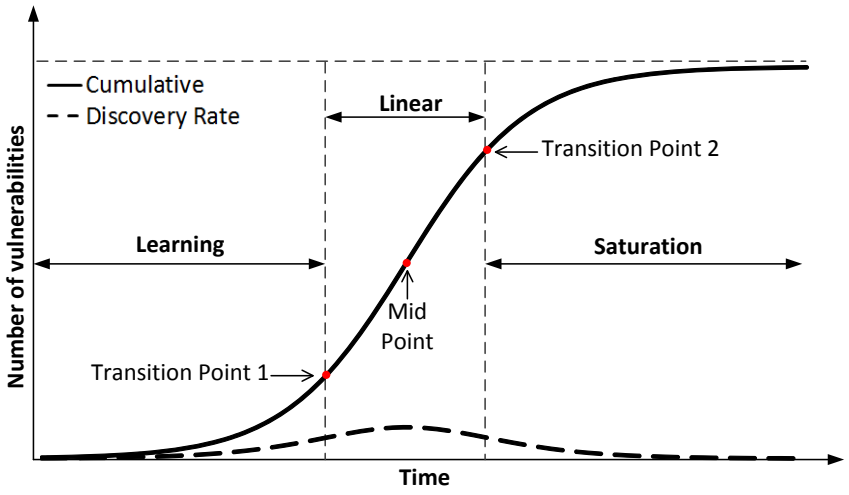


Figure 3.2: Three phases in S-shaped AML model

The applicability of VDMs to several operating systems was examined in (Alhazmi and Malaiya, 2008). The results indicate that while some of the models fit the data for most operating systems well, others do not fit well or just provide a good fit only during a specific phase. In this Chapter, Alhazmi-Malaiya Logistic (AML), linear, and Alhazmi-Malaiya effort-based models are reviewed, which are the models mainly used in the dissertation.

3.1 Alhazmi-Malaiya Logistic model

In Figure 3.2, the solid S-shaped line shows the shape of the Alhazmi-Malaiya Logistic (AML) model (Alhazmi and Malaiya, 2008). The model is based on the observation that the attention given to an operating system increases as it gains market share, it peaks at some time and then drops when a newer competing version is introduced. It is assumed that the cumulative number of vulnerabilities is governed by two factors in Equation 3.1. The first factor increases with the time because of the rising share of the installed base. The second factors declines as the number of remaining undetected vulnerabilities declines. The saturation effect is modeled by the second factor. Assuming that the vulnerability discovery rate is given by Equation 3.1, Equation 3.2 can be obtained by solving the differential

equation which gives the cumulative number of vulnerabilities Ω. Parameters A and B are empirical constants determined from the recorded data. C is a constant introduced while solving Equation 3.1. The model is defined for time values t from the negative infinity to the positive infinity.

$$\omega_{AML}(t) = A\Omega(B-\Omega) \tag{3.1}$$

$$\Omega_{AML}(t) = \frac{B}{BC^{-ABt}+1} \tag{3.2}$$

During the release period, the vulnerability discovery rate gradually increases. At this phase, called learning phase, as shown in Figure 3.2, the software is gaining market share gradually. In the linear phase, the discovery rate reaches the maximum due to the popularity, and finally, in the saturation phase, vulnerability discovery rate slows down. The two transition points and mid-point are mathematically defined in next section or in (Alhazmi and Malaiya, 2006b).

The transition points, especially for mid-point and the second transition point are not deterministic. They could be possibly altered depending on the future environment of how much codes are going to be newly injected on the original version. As a result, under certain circumstances, the S-shape could be mutated in various ways. Among those mutant S-shapes, a linear discovery pattern seems to appear noticeably in many popular software systems recently (Schryen, 2009).

As mentioned previously, Kim et al. (2007) proposed an advanced version of AML model considering multi-version software systems which incorporates the impact of vulnerabilities discovered in the code inherited by the later versions. They empirically shows that when shared vulnerabilities from a successive version are added, the linear phase in the overall vulnerability discovery process is extended, and they says it is an example of the superposition effect (Eick et al., 2001). Despite of the higher accuracy than the original AML, the multi-version AML is quite ponderous to be utilized right away due to the model parameter of how much code and functionality are shared between the successive versions.

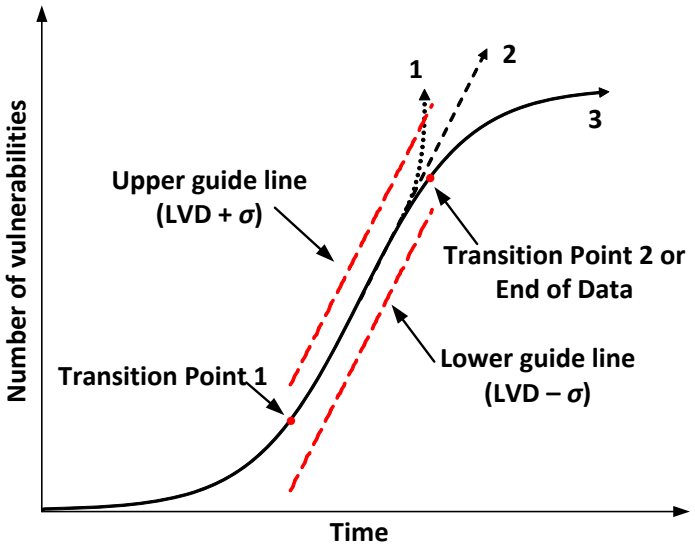


Figure 3.3: Upper and lower boundaries for linear phase testing

3.2 Linear model

Linear Vulnerability Discovery model (LVD) is another example of time-based vulnerability discovery model that we examine in this dissertation. The LVD model assumes that the vulnerability discovery rate is constant in any circumstance. The simple linear model can be expressed as Equation 3.3.

$$\Omega(t) = k + S \times t \tag{3.3}$$

where S is a slope that indicates vulnerability discovery rate and k is a constant. Applying the LVD model to the entire lifecycle of a vulnerability dataset and obtaining a good result is difficult. However, we can apply the LVD model to the linear phase of AML model in Figure 3.2.

To apply the LVD to vulnerability discovery datasets, it is necessary to identify the two transition points. The derivatives of Equation 3.2 arises the vulnerability discovery rate in Figure 3.2. The maximum and minimum for second derivatives of Equation 3.2 derives two transition points. These two points are the transition point 1 and point 2 in Figure 3.2. Equation 3.4 shows the derivatives of Equation

3.2.

$$\frac{d\Omega}{dt} = \frac{AB^3Ce^{-ABt}}{(BCe^{-ABt}+1)^2} \tag{3.4}$$

Note that the highest value of Equation 3.4 occurs at the midpoint of Figure 3.2 at time $T_m = lnBC/AB$. Equation 3.5 shows the derivatives of Equation 3.4.

$$\frac{d^2\Omega}{dt^2} = \frac{2A^2C^2B^5Ce^{-ABt}}{(BCe^{-ABt}+1)^3} - \frac{A^2B^4Ce^{-ABt}}{(BCe^{-ABt}+1)^2} \tag{3.5}$$

The maximum and minimum values in Equation 3.5 represent the two transition points in Figure 3.2. The solutions of the third derivative from Equation 3.2 to zero are the two transition points which occur at times $T_1 = ln[BC/(2 + sqrt(3))]/AB$ and $T_2 = ln[BC/(2 - sqrt(3))]/AB$.

We set the upper boundary LVD $+\sigma$ and the lower boundary LVD $-\sigma$ shows in Figure 3.3 to decide whether a discovery process is in the linear phase or not, where σ is a standard deviation between the LVD projection and actual datasets. When a real data is between these two boundaries or upper side of LVD $-\sigma$, it is in a linear phase. Line 1 and 2 in Figure 3.3 is examples of these cases. When a real data is lower than LVD $-\sigma$, in 3.3, we can say it reach a saturation phase like line 3 in the figure. The model is utilized in Chapter 5 for the rapidly evolving software systems, Web browsers.

3.3 Alhazmi-Malaiya effort-based model

VDMs are usually based on the calendar time due to the easiness of tracking and associating vulnerabilities to the time of discovery. Another point of view, instead of the calendar time, is share of the installed base of the specific system. The philosophy of this method is that a larger share of the installations should cause more vulnerabilities discovered since more people should test the system with a bigger market sharing. 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 find or exploit vulnerabilities that exist in a large number of computers. Hence, it can be expected that a larger share of the effort, which is going into the discovery of vulnerabilities, both internal and external, would go toward a system with a larger installed base.

The effort-based model, introduced by Alhazmi and Malaiya (2005), is using the concept of Equivalent Effort (E) as shown in Equation 3.6, where U_i and P_i represent, at the period of time i, the total number of users of all systems and the percentage of users using the system. It is based on the assumption that the vulnerability finding effort is proportional to actual usage, as given by the total number of installed systems. The number is measured in million system month (MSM) usually.

$$E = \sum_{i=0}^{n}(U_i \times P_i) \tag{3.6}$$

$$\Omega(t) = B(1 - e^{-\lambda_{vu}E}) \tag{3.7}$$

Sometimes, equivalent effort reflects the effort that would have gone into finding vulnerabilities more accurately than time alone, and it is analogous to using CPU time for software reliability growth models (Musa, 1999). When we assume that the vulnerability detection rate with respect to effort is proportional to the fraction of remaining vulnerability, the exponential model as shown in Equation 3.7 can be achieved, where λ_{vu} is a parameter similar to failure intensity in SRGMs, and B is another parameter representing the number of vulnerabilities which will be found eventually. However, we should keep in mind that the usage or market share information is not always available for many software systems. The model is examined in Chapter 4 and 5.

3.4 Factors influencing VDM

There are important factors to impact a vulnerability discovery rate. Most significant factors are code size, software age, popularity and software evolution.

Several studies (Akiyama, 1971; Compton and Withrow, 1990; Hatton, 1997; Rosenberg, 1997) have examined the relationship between the code size and the number of defects. The studies show that the number of defects or errors is increasing as code size is getting bigger. A first order approximation assumes a linear relationship between the code size and the number of defects, which allows measuring the defect density. Since the vulnerabilities are a class of defects, we can similarly define a measurement called vulnerability density (Alhazmi et al., 2005). Available data allows us to calculate the densities of the discovered vulnerabilities.

Market Share is one of the most significant factors impacting the effort expended in exploring potential vulnerabilities. A higher market share provides more incentive to explore and exploit vulnerabilities for both black hats and script kiddies since both would find it more profitable or satisfying to spend their time on a software system with a higher market share. The effect of rise-and-fall in the market share is implicit in the AML model. Meanwhile, the time-based models reflect software age more explicitly than the effort-based model.

Chapter 4

VULNERABILITY DISCOVERY PROCESS IN WEB SERVERS

In this chapter, the feasibility of characterizing the vulnerability discovery process in the two major HTTP servers, Apache and IIS, is quantitatively examined using both time and effort based vulnerability discovery models, explained in Chapter 3.1 and 3.3 respectively, using data spanning more than a decade. The data used incorporates the effect of software evolution for both servers. In addition to aggregate vulnerabilities, different groups of vulnerabilities classified using both the error types and severity levels are also examined. Results show that the selected vulnerability discovery models of both types can fit the data of the two HTTP servers very well. Results also suggest that separate modeling for an individual class of vulnerabilities might be done. In addition to the model fitting, predictive capabilities of the two models are also examined. The results demonstrate the applicability of quantitative methods to widely used products, which have undergone evolution.

4.1 Introduction

Two of the major software components of the Internet are HTTP (Hyper Text Transfer Protocol) servers (also termed Web servers) and Web browsers, which serves as clients. Both components were introduced in 1991 by Tim Berners-Lee of CERN [1]. They have now become indispensable parts of both organizational and personal interactions. The early Web servers provided information using static HTML pages.

[1] `http://info.cern.ch/`

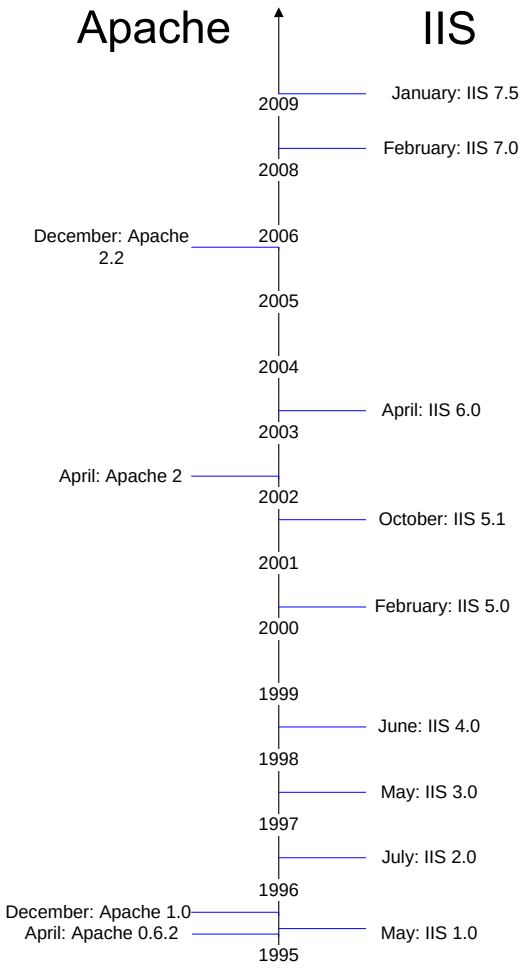


Figure 4.1: Apache and IIS release timeline for major versions

The Web servers now provide dynamic and interactive services to the clients using database queries, executable script, etc. Web servers can support functions such as serving streaming media, email, etc. In the emerging cloud computing systems, the HTTP servers support virtual implementations of applications and operating systems. HTTP servers have, thus, emerged as a focal point for the Internet.

The Apache Web server was introduced in 1995, and the Microsoft Internet Information Services (IIS) Web server was originally supplied as part of the NT operating systems in 1995-1996. Figure 4.1 shows the major versions' timeline for the two Web servers. While Apache has a much larger overall market share, roughly 55% on March 2010, IIS may have a higher share of the corporate Websites. The

Table 4.1: Functional support comparison in Apache and IIS

Feature	IIS	Apache
ASP	Native	With Chilisoft, Apache::ASP, etc.
Active directory authentication	Yes	With third-party modules
Live configuration editing	Yes	No
CGI, Perl, Pyhon, PHP, JSP	YES	Yes
Runs under Windows OSes	Yes	Yes
Runs under Unix-like OSes	No	Yes

Source: `http://www.serverwatch.com/tutorials/article.php/3074841`

market share for other servers is very small, and thus, they are not examined here. IIS is the only major HTTP server that is not open-source. Both Apache and IIS are generally comparable in features. IIS runs only under the Windows operating systems and comes bundled with some of the versions, whereas Apache supports all the major operating systems. Table 4.1 gives a brief functional comparison between the two.

The security of systems connected to the Internet depends on several components of the system. These include the operating systems, HTTP servers and Web browsers. Some of the major security compromises arise because of vulnerabilities in the HTTP servers.

The exploitations for some of the server vulnerabilities are well known. The Code Red worm (Moore et al., 2002), 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.

All the computing systems connected to the network are subjects to some security risk. While there have been many studies attempting to identify causes of vulnerabilities and potential counter-measures, the development of systematic quantitative methods to characterize security have begun only recently. There has been considerable debate comparing the security attributes of open source and proprietary software (Anderson, 2002). 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 which will be exploited. Thus, the risk is better represented by the vulnerabilities which are not yet discovered and the vulnerability discovery rate rather than by the vulnerabilities that have been already discovered in the past and remedied by patches.

HTTP servers are very attractive targets for malicious attackers. The servers can represent the first line of defense against attacks 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 patches have not been developed or applied. Despite the significance of security in the HTTP servers, only limited work has been done related to the vulnerability discovery process for the servers (Woo et al., 2006b; Álvarez and Petrovic, 2003). Such work would permit the developers and the users to better estimate future vulnerability discovery rates. It would also be highly desirable to be able to project what types of vulnerabilities are more likely to be discovered.

Some of the available works on HTTP servers discuss some specific problems or attacks that the servers might be faced, such as denial of service attacks (DoS) (Aura et al., 2000; Kargl et al., 2001), in which the authors suggest some countermeasures to be applied when attacks for these types take place. Our focus is the discovery rates of vulnerabilities in the two most popular Web server software systems, which have undergone significant evolution. Unlike some of the recent studies on the discovery rates in specific operating systems (Alhazmi and Malaiya, 2008), complete data for all the versions of the two servers is examined in the chapter.

Here, the applicability of the two most successful models for HTTP servers is investigated. The models used here are the logistic time-based model and the

effort-based model proposed by Alhazmi and Malaiya (Alhazmi and Malaiya, 2005) introduced in Chapter 2. These two models have been found to fit datasets of the major Windows and Linux operating systems, as determined by goodness of fit and other measures (Alhazmi and Malaiya, 2008; Alhazmi et al., 2007). In this chapter, the Alhazmi-Malaiya Logistic Model is referred to as the time-based model, and Alhazmi-Malaiya effort-based model is termed the effort-based model. Several time-based models have been proposed, however the Alhazmi-Malaiya Logistic Model has selected for the analysis and comparison here since it has generally provided a better fit compared to other models (Alhazmi and Malaiya, 2008). The effort-based model has been selected because it is the only model proposed in the literature that uses effort instead of time. The two models contrast two different approaches.

In the chapter, for the effort-based model, in Equation 3.6, U_i is the total number of installations of the HTTP servers at the period of time i, n represents the last usage period, and P_i is the percentage of the servers using the specific software for measuring E. N_i is the number of machines running the specific server during time i. The result is obtained in terms of system-months. The measure U_i can be calculated using the data about total number of Web servers.

4.2 Aggregate vulnerabilities in HTTP server

In this section, the datasets for the total vulnerabilities of the Apache and IIS Web servers are fitted to the models. The goodness of fit is evaluated to determine how well the models reflect the actual vulnerability discovery process. The vulnerability datasets are from NVD. The market share data from Netcraft[2] are used. Note that Apache represents an open source software system whereas IIS represents a proprietary closed source system.

Market share is one of the most significant factors impacting the effort expended in exploring potential vulnerabilities. Higher market share indicates more incentive

[2] `http://news.netcraft.com/`

Table 4.2: Market share & vulnerabilities found

Web server	Apache	IIS	Nginx	Lighttpd	Other
Market share	52.26%	32.91%	1.87%	1.61%	11.35%
Vulnerabilities	132	137	0	18	N/A
Release year	1995	1995	2005	2003	N/A
Latest version	2.2.11	7.5	0.6.35	1.4.20	N/A

to explore and exploit vulnerabilities since people would find it more profitable or satisfying to spend their time on a software system with a higher market share.

Table 4.2 presents data obtained from NVD and Netcraft on January 2009, showing the current Web server market share and total number of vulnerabilities found to date. Since very little information about Google Web server is publically available, Google Web server is omitted from the table although its market share is approximately 5%. For servers with a lower percentage of the market, such as Google Web server, Nginx and Lighttpd, the total number of vulnerabilities found is zero or very few. However, that does not mean that these systems are more secure, but merely that only limited effort has gone into finding their vulnerabilities. A significant number of vulnerabilities has been found in both Apache and IIS, illustrating the impact of the market share on the motivation for exploring or finding vulnerabilities. Here, the market share is used as an indicator of effort for the effort-based model.

Figure 4.2 shows the Web server market share for Apache and IIS. As demonstrated by the figure, the number of Web servers continues to grow steadily. Among the various Web servers, Apache and IIS dominate the Web server market. Other Web servers such as Nginx and Lighttpd occupy a very small share of the market, as shown in Table 4.2. Since the total share of all the Web servers, except Apache and IIS, represents less than 15% of the market share, few vulnerabilities have been found in them and, hence, the data for these servers has not been used here.

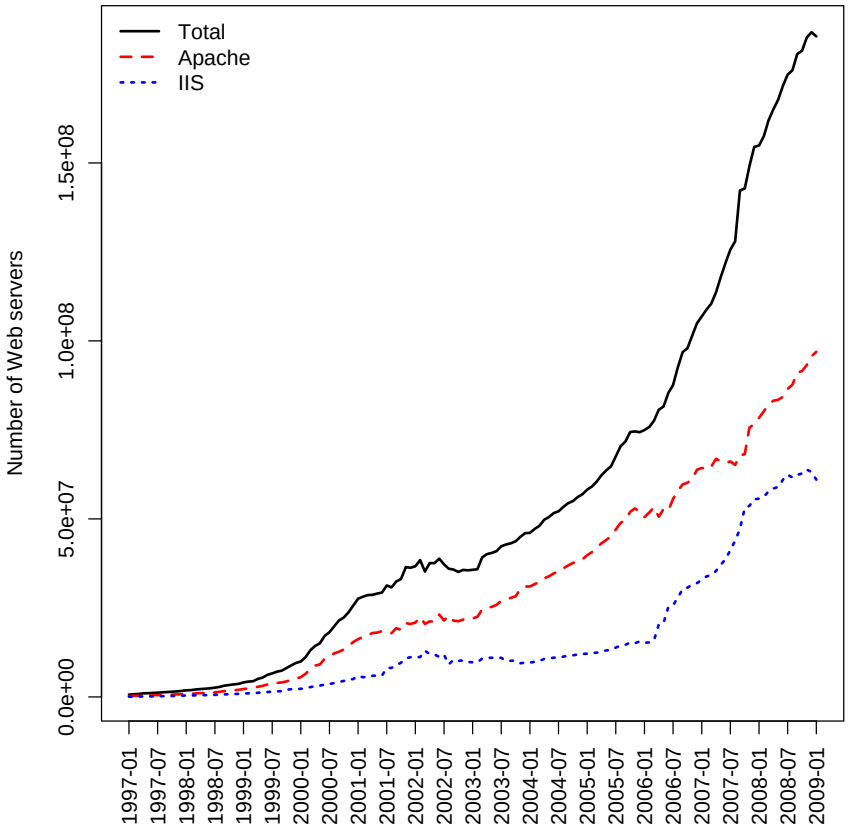


Figure 4.2: Web server market share trends

There is a marked gap between the Apache and IIS market shares, as shown in Figure 4.2. This difference in market share may be due to several factors. Perhaps the most important of these is that Apache is available for all major operating system platforms and can be obtained without cost. Apache may also have benefited from not having been exposed to serious security issues such as the Code Red (Moore et al., 2002) or Nimda worms (Machie et al., 2001) that were faced by IIS in 2001.

Since its release in 1995, Apache HTTP server has achieved and maintained a large installed base and was used by over 90 million Web server systems during January 2009. In this section, the vulnerability datasets are fitted to the time-based and the effort-based models. Figure 4.3 gives the vulnerability data from NVD for the period between January 1996 and December 2008. Netcraft provides the market share data covers this time period.

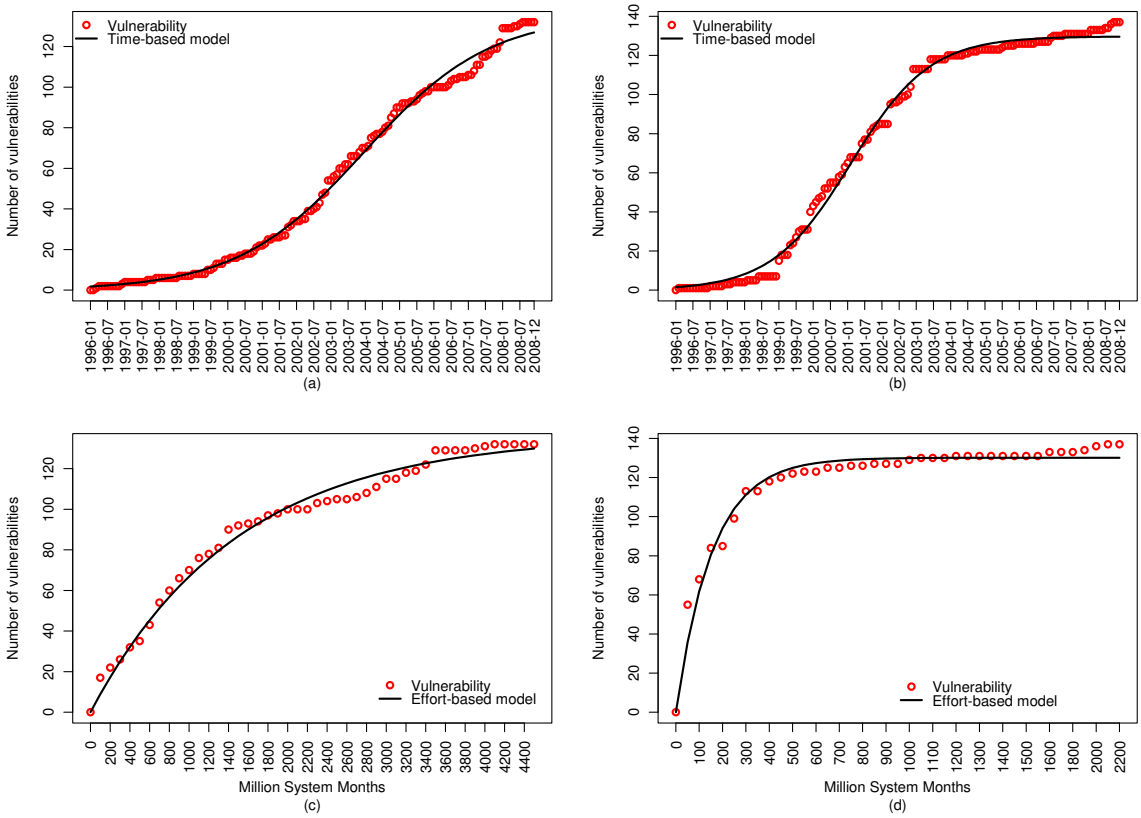


Figure 4.3: Fitting Web server - aggregated vulnerability data; (a) and (b) show the time-based model fittings for Apache and IIS respectively. (c) and (d) are effort-based model fittings for Apache and IIS respectively

In Figure 4.3, the solid lines indicate the fitted models while the "O" marks show cumulative vulnerabilities for the servers. Figure 4.3 (a) shows cumulative Apache Web Server's vulnerabilities by month for the time-based model. The slope of the curve for Apache rises gently until about January 2000, after which the slope had remained steady until the end of 2007. From the point of the three phases in the vulnerability discovery process (Alhazmi and Malaiya, 2005), Apache may be entering the saturation phase, since only three vulnerabilities were found in 2008.

Figure 4.3 (c) shows cumulative vulnerabilities by the number of Apache installations in terms of million system-months and the fitted effort-based model. This effort-based model also shows that Apache is approaching the saturation phase since any vulnerability has not been found after 4000 million system months as the number of Apache severs increases.

Table 4.3: χ^2 goodness of fit test - Time-based

	A	B	C	χ^2	$\chi^2_{critical}$	P-value
Apache	0.0003	135.57	0.5745	24.69	186.1458	1
IIS	0.0005	129.74	0.7034	79.82	186.1458	1
Windows NT	0.00018	253.3	0.1293	136.14	173.0041	0.64535
Windows 98	0.0004	100.94	0.1002	81.11	114.2679	0.99688

Table 4.4: χ^2 goodness of fit test - Effort-based

	B	λ_{vu}	χ^2	$\chi^2_{critical}$	P-value
Apache	136.50	0.0007	16.224	61.65623	0.99996
IIS	130.13	0.0064	14.688	55.75848	0.99961

IIS, released in 1995, is the only major proprietary Web server with over 60 million installations during January 2009. The vulnerability dataset from January 1996 to December 2008 is used in this analysis. Figure 4.3 (b) shows the cumulative number of vulnerabilities by month and the fitted time-based model for the IIS Web server. The time-based and effort-based models fit the data for IIS very well. The IIS Web server appears to have reached the saturation phase since 2004 the vulnerability finding rate has been low. There was a recent increase because of the six new vulnerabilities found during 2008. A possible explanation for this could be that the number of IIS Web servers installed has increased since 2006 and a new version of IIS was released in February 2008.

Figure 4.3 (d) shows the cumulative number of vulnerabilities for the IIS server and the effort-based model by million system-months. The figure shows a significant degree of saturation.

The model fittings as shown in Figure 4.3 have been examined in Table 4.3 and 4.4. The two tables give the χ^2 (chi-square) values and model parameter values for the time-based and effort-based models respectively. For more information about the χ^2 goodness of fit test, see Chapter 7.4. For χ^2 goodness of fit test, the alpha level here is 0.05. For comparison, the corresponding parameter values are also provided

for the Windows 98 and NT operating systems, as well as the chi-square values. The chi-square values are less than the critical values which demonstrates that the fit for Apache, IIS, Windows 98 and NT is significant for the both models with P-values ranging from 0.64535 to 1, indicating that the fit is statistically significant. It is observed that parameter A is always less than 0.005 and parameter C is also less than 0.71, while parameter B corresponds approximately to the number of vulnerabilities.

4.3 Vulnerability categories

In the previous section, the application of the two models for the total aggregated number of vulnerabilities of Apache and IIS has been examined. Here, the models applied to portioned data using a classification scheme for server vulnerabilities.

Distinction among vulnerabilities is useful when practitioners want to examine the nature and extent of the problem. It can help to determine what kinds of protective actions would be most effective. Vulnerability taxonomy is still an evolving area of research. Several taxonomies have been proposed (Aslam et al., 1996; Bishop, 1999; Landwehr et al., 1994; Seacord and Householder, 2005; Rajeev Gopalakrishna and Vitek, 2005; Venter et al., 2008). An ideal taxonomy should have such desirable properties as mutual exclusiveness, clear and unique definition, and coverage of all software vulnerabilities.

Vulnerabilities can be classified using schemes based on cause, severity, impact, source, etc. In this analysis, the classification scheme is used which was employed by the NVD. This classification is based on the causes of vulnerabilities. The eight classes are as follows (Alhazmi et al., 2006):

1. Input Validation Error (Boundary condition error, Buffer overflow): Such types of vulnerabilities include failure to verify the incorrect input and read-write involving an invalid memory address.

2. Access Validation Error: These vulnerabilities cause failure in enforcing the correct privilege for a user.

3. Exceptional Condition Error: These arise due to failures in responding to unexpected data or conditions.

4. Environmental Error: These are triggered by specific conditions of the computational environment.

5. Configuration Error: These vulnerabilities result from improper system settings.

6. Race Condition Error: These are caused by the improper serialization of the sequences of processes.

7. Design Error: These are caused by improper design of the software structure.

8. 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. 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 (also the percentages may exceed 100%) (Gopalakrishna and Spafford, 2005).

Figure 4.4 compares vulnerability distributions in Apache and IIS. The categories with the highest numbers are input validation errors, followed by design and configuration errors. There is a slight difference in category ordering between Apache and IIS. Apache has more configuration errors than access validation errors whereas IIS has more access validation errors. While IIS has been more vulnerable to access

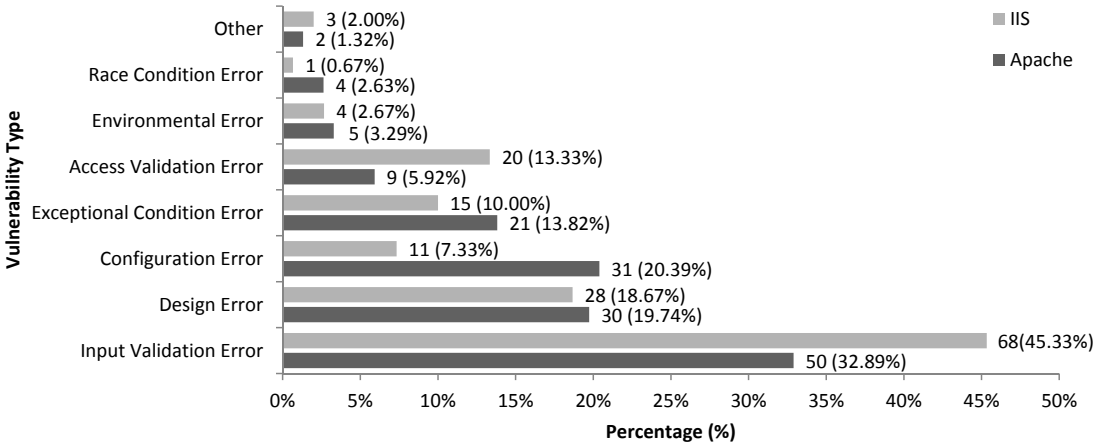


Figure 4.4: Categorizing Web server vulnerabilities by type

validation errors, the fact that Apache has been more vulnerable to configuration errors may be due to Apache's more complex installation requirements.

To determine whether there is an observable pattern at the level of individual classes, the vulnerabilities are plotted for the major categories. Since a similar pattern for the uncategorized vulnerabilities is noted, a possible fit was examined. Figure 4.5 show the fit for the Apache and IIS. It may be noted that the number of input validation errors and design errors are the most common category in Apache and IIS. We chose to fit the categories which have enough data points available for fitting. In the figures, these three major categories are shown: input validation errors, design and access validation errors.

The categorized number of vulnerabilities shows the same pattern as demonstrated by the total number of vulnerabilities. Thus, each category shows a related pattern regarding to the total number of vulnerabilities. Time-based and effort-based models are fitted for each category. Table 4.5 and 4.6 show the χ^2 goodness of fit tests for the Apache and IIS models by category respectively. The tables demonstrate that the χ^2 values for each category are less than the corresponding χ^2 critical values and the P-values are close to 1. The fits for input validation, design and access validation error classes are significant for both models.

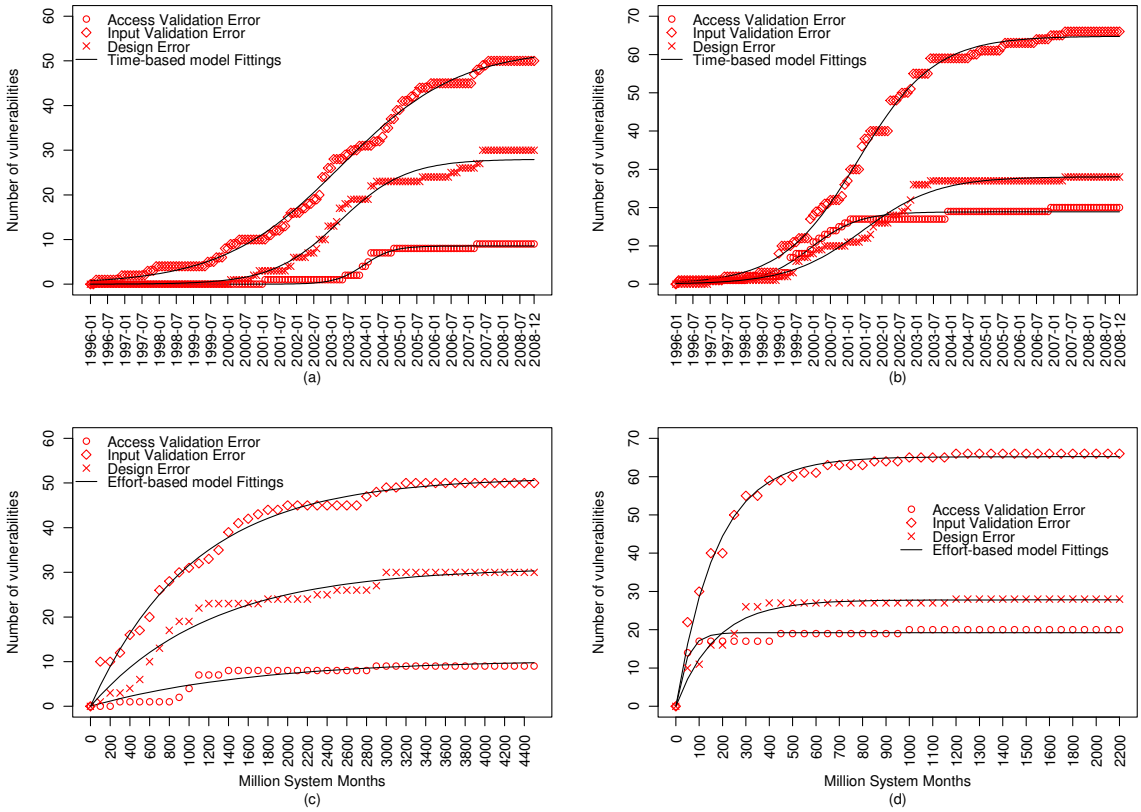


Figure 4.5: Fitting Web server - vulnerability by type; (a) and (b) show the time-based model fittings for Apache and IIS respectively. (c) and (d) are effort-based model fittings for Apache and IIS respectively

4.4 Vulnerability severity level

Severity is another way of classifying vulnerabilities. The severity level of a vulnerability indicates how serious the impact of an exploitation can be. Three severity levels are often defined; high, medium and low. Some other organizations such as Secunia [3] use three to five levels and use their own definition for severity. The NVD has used Common Vulnerability Scoring System (CVSS) (Mell et al., 2007) metric for vulnerability severity with ranges from 0.0 to 10.0; CVSS uses many factors to determine the severity where the range from 0.0 to 3.99 corresponds to low severity, 4.0 to 6.99 to medium severity and 7.0 to 10.0 to high severity. The NVD describes three severity levels as follows:

[3] `http://secunia.com/`

Table 4.5: χ^2 goodness of fit test results by type - Time-based

		A	B	C	χ^2	$\chi^2_{critical}$	P-val.
Apache	Input	0.00036	52.75307	1.342086	19.87597	186.1458	1
	Design	0.00064	27.99523	67.37607	27.92677	186.1458	1
	Access	0.001474	8.554207	22060622	120.5440	186.1458	1
IIS	Input	0.000559	64.79746	1.964722	24.007	186.1458	1
	Design	0.000887	18.89434	15.97189	81.139	186.1458	1
	Access	0.000574	28.06895	5.850765	31.74241	186.1458	1

Table 4.6: χ^2 goodness of fit test results by type - Effort-based

		B	λ_{vu}	χ^2	$\chi^2_{critical}$	P-val.
Apache	Input	51.2449	0.000958	0.006129	61.65623	0.99998
	Design	31.1015	0.000817	0.003253	61.65623	0.99999
	Access	10.4792	0.0006	0.06152	61.65623	1
IIS	Input	65.20718	0.005732	3.242342	55.75848	1
	Design	19.24239	0.022189	2.060538	55.75848	1
	Access	27.79932	0.005897	3.050723	55.75848	1

1. High Severity: vulnerabilities make it possible for a remote attacker to violate the security protection, or permit a local attack that gains complete control, or are otherwise important enough to have an associated CERT/CC advisory or US-CERT alert.

2. Medium Severity: vulnerabilities are those not meeting the definition of either 'high' or 'low' severity.

3. Low Severity: vulnerabilities typically do not yield valuable information or control over a system but may provide the attacker with information that may help him find and exploit other vulnerabilities or may be inconsequential for most organizations.

The distributions of the severity levels of the Apache and IIS vulnerabilities show similarity. About 60% to 70% of total vulnerabilities are the medium severity, followed by 30% to 40% with high severity, with low severities at about 5%.

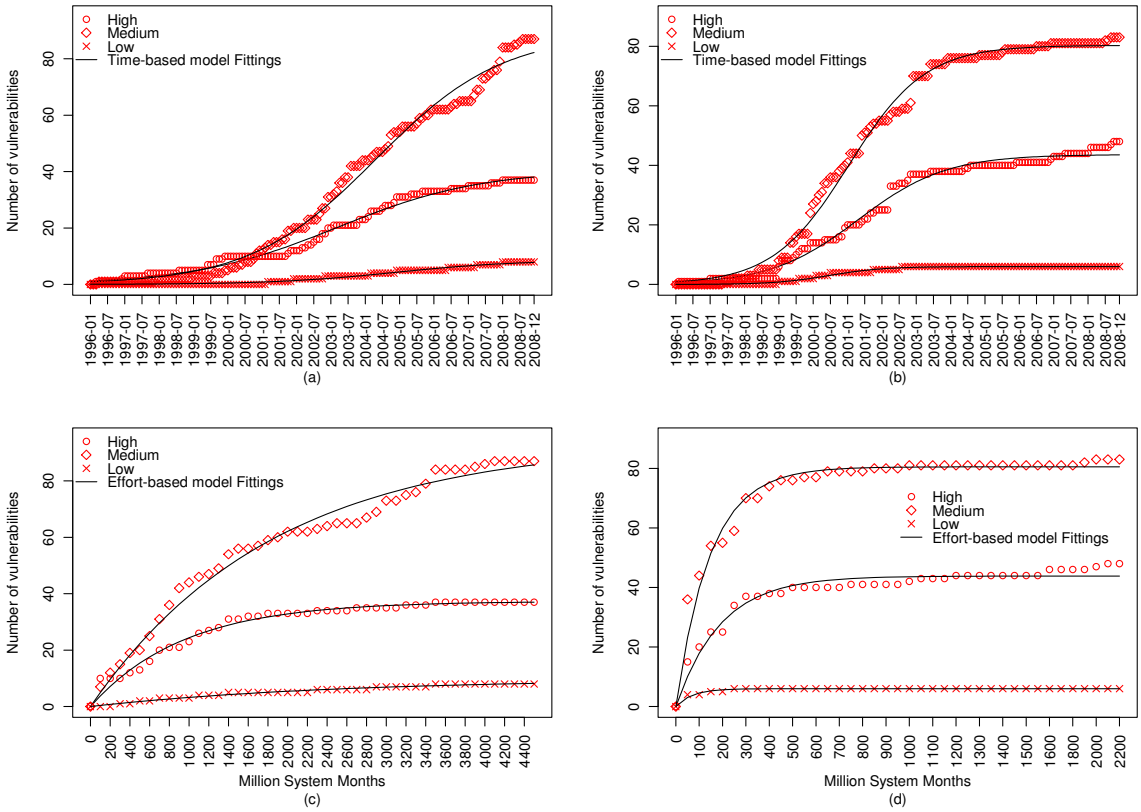


Figure 4.6: Fitting Web server - vulnerability by severity; (a) and (b) show the time-based model fittings for Apache and IIS respectively. (c) and (d) are effort-based model fittings for Apache and IIS respectively

The time-based and effort-based models have been applied to the three severity classes. In Figure 4.6, the solid lines indicate the fitted time-based and effort-based models for each severity level. The figures show the result of fitting the time-based and effort-based models to the three severity classes. The plots suggest that especially for the medium severity vulnerabilities, the IIS vulnerability data appears to be in the saturation phase while the Apache vulnerabilities are still being discovered.

Table 4.7 and 4.8 show the χ^2 goodness of fit tests and the parameter values for Apache and IIS by severity level respectively. The parameter values are obtained from data corresponding to Figure 4.6 using regression analysis. As before, for χ^2 goodness of fit test, the test alpha level is 0.05. This χ^2 test shows that the

Table 4.7: χ^2 goodness of fit test results by severity - Time-based

		A	B	C	χ^2	$\chi^2_{critical}$	P-val.
Apache	High	0.000337	135.575	0.574563	28.85319	186.1458	1
	Medium	0.000298	40.56143	0.805567	55.79249	186.1458	1
	Low	0.000339	88.93068	1.092679	24.48047	186.1458	1
IIS	High	0.000542	129.7421	0.703459	32.68021	186.1458	1
	Medium	0.000487	43.63152	1.577562	72.58211	186.1458	1
	Low	0.000562	80.2965	1.271963	10.52017	186.1458	1

Table 4.8: χ^2 goodness of fit test results by severity - Effort-based

		B	λ_{vu}	χ^2	$\chi^2_{critical}$	P-val.
Apache	High	37.32424	0.001061	12.88105	61.65623	0.9999
	Medium	93.90457	0.000541	9.335251	61.65623	1
	Low	9.835684	0.0004	2.359279	61.65623	0.9999
IIS	High	43.83469	0.005239	6.066258	55.75848	0.9996
	Medium	80.51197	0.006809	9.343797	55.75848	1
	Low	5.987127	0.014206	0.460018	55.75848	0.9999

fits for the three severity categories are significant, and the χ^2 tests show that the vulnerabilities classified by severity datasets fit the model well.

The fraction of high and medium severity vulnerabilities is substantial and presents a significant risk to the HTTP servers potentially leading to problems such as unauthorized system access, denial of service (DoS) attack, exposure of sensitive information, etc. Figure 4.7 plots the percentage of the cumulative number of vulnerabilities for each severity class for each month for Apache and IIS. Both Apache and IIS show a similar pattern. Note that the first few points in the plots are not significant since they represent only a small number of vulnerabilities. The plots suggest that a larger fraction of the high severity vulnerabilities is found early while the medium severity vulnerabilities represents about 60%~70% share after three or four years later. This data suggests that there may be a deliberate effort to focus on high severity vulnerabilities in early phase. This is supported by the observations about the patching rate of input validation errors which tend to have higher severity

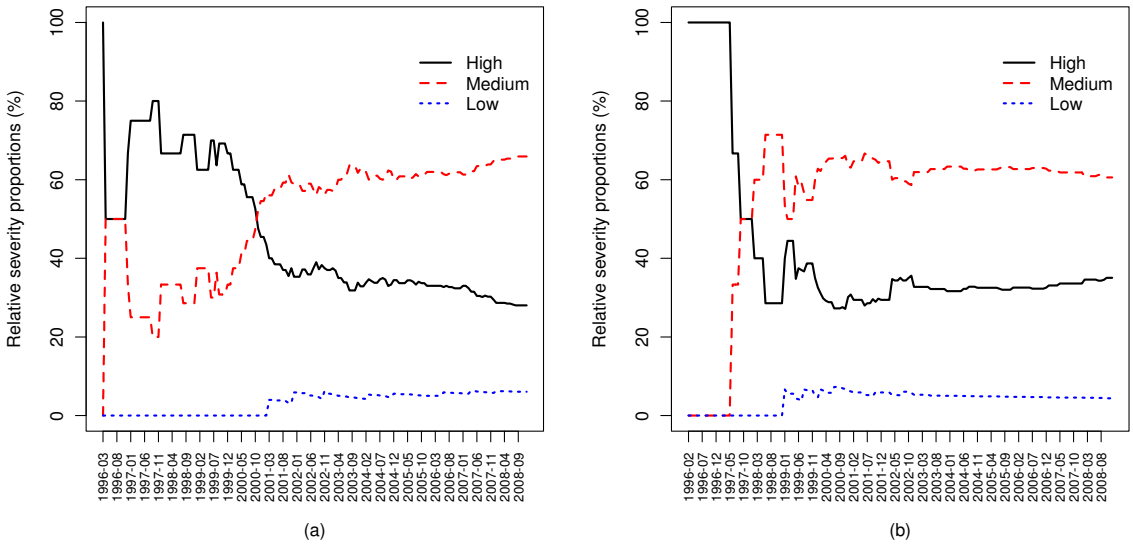


Figure 4.7: Vulnerability proportions by severity are shown for Apache and IIS in (a) and (b) respectively

levels. Penta et al. (2009) have empirically shown that buffer overflows are patched significantly faster than other types of vulnerabilities because they represent the kind to which the developers tend to respond faster.

4.5 Predictive capability

Even when a VDM shows the nice goodness of fit during the period covered, it is possible that the model may not be able to predict well in the future if the model does not anticipate changes in the trend that is actually encountered. In the software reliability engineering field, the predictive capability for a number of reliability growth models has been investigated in the past (Musa and Okumoto, 1984; Malaiya et al., 1992). A VDM having a better predictive capability should be able to estimate the future behavior better, for example, the total number of vulnerabilities using currently available datasets. It can be used to estimate the resources needed for maintenance and the risk estimation.

Here, the starting point for comparing the two models is chosen to be when cumulative installations exceed 100 million and 50 million for Apache and IIS Web

Table 4.9: Prediction error

	Effort-based	Time-based
Apache	-0.0749	-0.1665
IIS	-0.1177	-0.1153

servers respectively since only after some significant time, the effort-based model can project the future trend. The predictive capabilities for the two models can be comparable only when the two models have data points for the same specific time points. Because the time-based model has data points for every single month, but the effort based model does not, the models are applied for the calendar time with somewhat unequal interval of months during the estimations. For each time point, the available partial data at the point are fitted to the models using regression analysis to estimate model parameters. Then the models with the estimated parameters for each time point are used to predict the final number of vulnerabilities at the end of the time period. The estimated final values for each time point are compared with the actual number of vulnerabilities to calculate normalized estimation errors.

We use prediction error (PE) as a metric for comparison (Malaiya et al., 1992). PE is a measure of how well a model predicts throughout the test phase. Prediction error PE is given by:

$$PE = \frac{1}{n} \sum_{n}^{t=1} \frac{\Omega_t - \Omega}{\Omega} \tag{4.1}$$

where n is the total number of data points during the prediction period, and Ω is the actual number of vulnerabilities whereas Ω_t is the estimated final number of vulnerabilities at time t. The normalized errors $((\Omega_t - \Omega)/\Omega)$ of the estimated values for the two Web servers are shown in Figure 4.8. The PE values are given in Table 4.9, which suggests that the VDMs tend to underestimate Ω.

In all cases, prediction errors approaches the zero line as more and more of the data becomes available. For the Apache Web server, in Figure 4.8 (a), the effort-based model yields a lower prediction error than the time-based model. Also the

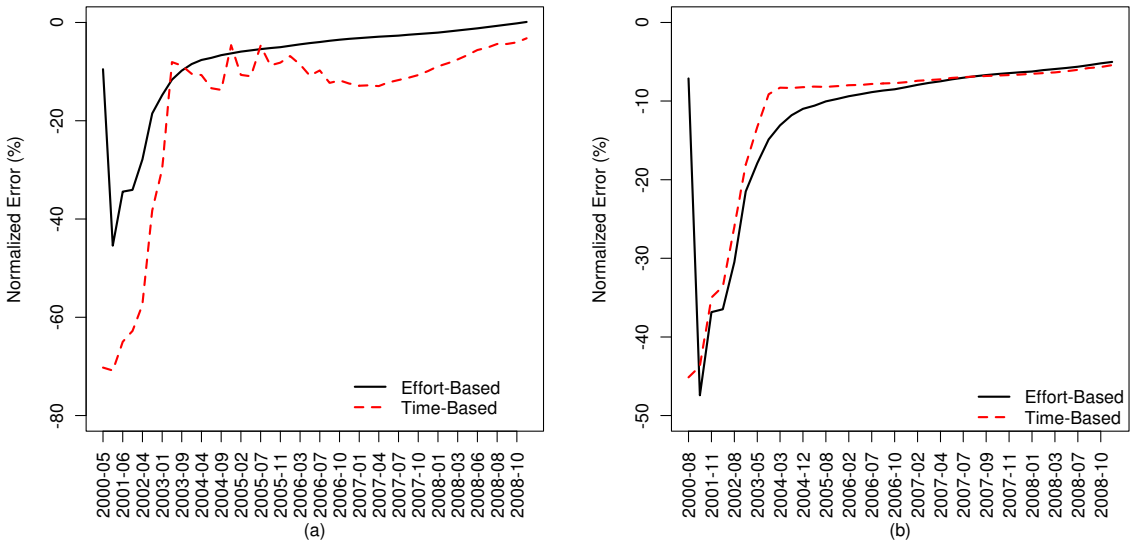


Figure 4.8: Prediction errors are shown for Apache and IIS in (a) and (b) respectivly

effort-based model stabilizes to the 0% error value line faster. The time-based model shows a somewhat similar pattern with the effort-based model but a bit less stable. For the IIS, the effort-based model and the time-based model both yield generally similar prediction patterns each other in Figure 4.8 (b). However, it needs to be kept in mind that the effort based model requires the use of market share data which may not always be readily available. In some cases, the effort variation assumed by the AML model is consistent with real data usage. In other cases, it is possible that usage may vary in a different manner or unpredictably. In such situations, explicitly use of the effort variation, as measured by the installed base, may provide more accurate predictions.

4.6 Discussion

When the total number of vulnerabilities is examined, both the time-based and effort-based models fit the datasets well, even when the vulnerabilities are categorized by type. This suggests that the models can be used to estimate the number of vulnerabilities expected to be discovered in a given period, and which types are likely to dominate.

Table 4.10: Known D_{KD} vs. Known V_{KD}

Application	Ksloc	Known Defects	D_{KD}	Known Vulnerabilities	V_{KD}	Ratio (V_{KD}/D_{KD})
Apache	373(Unix)	1380	3.699	132	0.353	0.0954
IIS	NA	NA	NA	137	NA	NA
Win 98	16,000	10,000	0.625	91	0.0057	0.0091
WinNT 4.0	18,000	10,000	0.556	197	0.0109	0.0197

The results of model fitting for the vulnerabilities classified by type are shown in Table 4.5. The fitting was done for the most common types of vulnerabilities for which the available data points are enough to be statistically significant.

The effort-based model requires the number of systems for target products in market share, which may be difficult to obtain. The time-based model does not require this data. It can, therefore, be a feasible alternative when market share data is unavailable.

Static analysis has been used in software reliability engineering, where some of the systems' attributes are estimated empirically even before testing begins. Similar static analysis can be carried out by utilizing metrics such as software size and estimated number of total defects. These methods can potentially be used to estimate Defect density and Vulnerability density as follows:

$$D_{KD} = \frac{KnownDefects}{Ksloc} \tag{4.2}$$

$$V_{KD} = \frac{KnownVulnerabilities}{Ksloc} \tag{4.3}$$

D_{KD}, defects per thousand lines of code, and V_{KD}, vulnerabilities per thousand lines of code, can then be used to estimate the total number of vulnerabilities of a comparable system. Table 4.10 shows the major attributes of the Apache server and two other legendary operating systems for comparison. Unfortunately, some of the important metrics for the Microsoft IIS server are not available. For proprietary systems, such data can be hard to obtain outside of the developing organization.

The sizes of IIS and Apache may be comparable in terms of Source Lines Of Code (SLOC) numbers since both offer the same features. The code size for Apache was determined using the SLOCCount tool[4]. Source code size of Windows 98 and NT 4.0 are given by McGraw (2003). The Apache 2.2.11 source code size for Windows is 240 Ksloc, smaller than for Unix version of Apache 2.2.11 of 373 Ksloc. A few of the Apache vulnerabilities may be applicable to only a specific platform. In Table 4.10, vulnerability density values for the Windows operating systems are significantly less than for Apache. This may be due to the fact that Windows operating systems have large segments that do not play a role in accessibility, while servers are smaller and therefore vulnerabilities are more concentrated in the code. This assumption is supported by the fact that the defect density to the vulnerability density ratio is higher in Windows NT 4.0, a server operating system, than in Windows 98, a client operating system. Note that V_{KD}/D_{KD} ratios are within a narrow range.

The data for IIS suggests that the vulnerability discovery rate has slowed down significantly since 2004. However, several vulnerabilities were found in IIS in 2008. This may be caused by a new version of IIS being released and the expansion of IIS market share. Factors such as patch releases, number of remaining vulnerabilities, economic aspects etc., also need to be considered when evaluating Web servers.

One interesting fact is that Apache, IIS and SUN Web servers share one common vulnerability (CVE-2008-2579), even though the three software systems do not share any source code. The vulnerability is unspecified in the Oracle WebLogic Server[5] Plug-in for the Web servers' component in some Oracle BEA[6] product suite. The vulnerability allows unauthorized disclosure of information, modification, and disruption of service.

The results show that the two VDMs are found to be applicable even though the two software systems have gone through a number of successive versions. Sharing of

[4] `http://dwheeler.com/sloccount`
[5] `http://www.oracle.com/technology/products/Weblogic/index.html`
[6] `http://www.bea.com`

the code among successive versions for some software systems have been examined by Kim et al. (2007). They have suggested that, for evolving software, the overall affect may be explained by a superposition of the trends for vulnerabilities for each individual version.

It should also be noted that the number of vulnerabilities, either found or estimated as remaining, should not be the only measurement of a security threat. Factors such as patch development and patch application delays and vulnerabilities' exploitation rates also need to be considered.

4.7 Summary

In this chapter, the applicability of quantitative models for the number of vulnerabilities and vulnerability discovery rates for the two most popular HTTP servers are explored. Results demonstrate that the vulnerability discovery in the Web servers follows certain patterns, which can be modeled. The results show that when the all the vulnerabilities are examined, both models fit the datasets well. The models were found provide significant fit even when vulnerabilities are categorized by cause or severity levels. This suggests that the models can be used to estimate not only the number of vulnerabilities expected to be discovered but also the likely distribution in terms of categories by origin and severity levels.

It was observed that a number of input validation error vulnerabilities can be large which would constitute a significant risk. The results can be used to optimize the distribution of testing and patch development effort by allocating more effort to vulnerabilities from classes that represent a higher risk, thereby reducing the overall risk due to vulnerabilities. The results indicate that the models originally proposed for operating systems are also applicable to HTTP servers.

Chapter 5

MODELING VULNERABILITY DISCOVERY PROCESS IN WEB BROWSERS

New vulnerabilities discovered in a Web browser put millions of users at risk, requiring urgent attention from developers to address these vulnerabilities. This chapter analyzes vulnerabilities presented in the six Web browsers quantitatively, which can be used to project the number of vulnerabilities to plan, test and allocate development resources more efficiently. They are Internet Explorer, Netscape, Firefox, Opera, Safari, and Chrome. End-users for the Internet browsers can build better idea about the risks associated with the use of a particular Web browser.

Here, the six Web browsers' vulnerability datasets are fitted to three vulnerability discovery models and their goodness of fit tests are examined. For the discovery models, two time-based models, Alhazmi-Malaiya Logistic and the linear models, and one effort-based model are applied (See Chapter 3 for models). The results show that both time-based and effort-based models generally provide the good fittings with most of the vulnerability datasets, opening the way for the applicability of the models to Web browsers' data. The datasets refined by severity is also shown to have fit the models, indicating that it is possible to predict the number of vulnerabilities belonging to a specific severity level.

5.1 Introduction

The Web browser provides variety and dynamic information from simple text to high qualities of multimedia contents for the Internet users. To display the information, the Web browsers use certain mechanisms and techniques, and need to follow international standards.

With respect to the end-users' point of view, the Web browsers now represent the single most important Internet software systems; one of its many roles, the Web browser serves as a client platform for security-critical applications such as the Internet banking, e-commerce and on-line trading in everyday life. There has been a great concern about potential insecurities in Web browsers due to their vulnerabilities, and the concern is getting more attentions. While the vulnerabilities and exploitations in the Microsoft Internet Explorer (IE) have been frequently discussed[1], its alternatives have not been debated actively even though the alternatives also are not immune to the serious vulnerability issues[2]. So far, many studies on the Web browser security have been in a qualitative manner, focused on detection and prevention of vulnerabilities in Web browsers. Several issues in the security problems related to the Web browsers are currently have been studied (Woo et al., 2006a), such as spyware (Moshchuk et al.), phishing (Dormann and Rafail, 2006; Kumar, 2005; Hallaraker and Vigna, 2005), Web page filtering (mat; Greco et al., 2004) and DNS rebinding (Jackson et al., 2009), malicious pop-up windows[3], and e-commercial fraud (Leung et al., 2004; Grazioli and Jarvenpaa, 2000).

The first public version of IE was released in August 1995. In 1999, Mozilla Web browser was born originated from Netscape Navigator which was introduced in October 1994, and Netscape emerged as the popular client Web browser during

[1] http://voices.washingtonpost.com/securityfix/2006/04/real_world_impact_of_internet_1.html

[2] http://www.eweek.com/c/a/Security/ZeroDay-Firefox-Exploit-Sends-Mozilla-Scrambling/

[3] http://www.pcworld.com/article/118781/new_ad_attacks.html

the 1990s. In 2004, Firefox 1.0 was announced, as the successor of Mozilla Web browser. Firefox occupies about 23% of the overall Web browser market share while IE conquers about 60% of the market in October 2010. Even though Opera's market share is less than 3% presently, it is one of the oldest Web browsers, and it is provided under various divides, from hand-held to desktop. Meanwhile, as users of Apple Mac computer are increasing, Safari Web browser is also getting popular having about 5% of market share now. Chrome, introduced by Google in September 2008, is widening its market share aggressively backed by mainly its speedy performance. Its market share now is about 8%, and it is expected the browser's market share will upsurge for a while.

For the Internet service providers, servers such as Web (HTTP), ftp, mail, database, and streaming servers, are the first and primary gate in contacting clients. For the clients, a Web browser is the main application, which connects for the clients to the Internet. However, Web browsers have been suffered from numerous security holes. Some of the vulnerabilities caused by those functions provide attackers or malicious users opportunities to exploit security holes since these processes require downloading, uploading and executing files having security holes. Web browser vulnerabilities represent one of the main sources of the spread for the viruses and worms. However, the convenience and dynamic technical functionalities offered by Web browsers make them currently indispensable.

5.2 Related works and background

Frei et al. (2008) show that a significant number of the Internet users are exposed at risk because people do not use the latest most secure Web browsers and plug-ins. In the paper, they quantified some of the insecurities in browser usages due to not updating to the most secure versions. In their later work, Duebendorfer and Frei (2010) have further investigated about effectiveness of the Web browser

updates on four different browsers and concluded that silent updates[4] and little dependency on operating systems are most effective for users to surf the Web with the latest browser versions; Google Chrome's silent auto update mechanism is the most effective compared to those of Firefox, Safari, and Opera.

Acer and Jackson (2010) challenged the fact that browsers receiving infrequent security patches are safer than those receiving frequent patches, that lower bugs count are safer, and that reducing vulnerabilities is the only way for vendors to improve security. They proposed methods for evaluating browser security that takes into account new industry practices such as the silent patch deployment. Grosskurth and Godfrey (2007) used a semi-automated analysis method to investigate the architecture and evolution of Web browsers. They observed some interesting phenomena such as different strategies for code reuse, emergent domain boundaries, convergent evolution, and tension between open and closed source development approaches.

Comparing Web browsers with other software systems such as operating systems and office software products, new versions of Web browsers are released faster than others. Releasing new versions of software systems means that defects are fixed and/or new functions are implemented. However, new versions do not indicate that the pool of vulnerability is reduced. New codes can cause other vulnerabilities.

5.3 Preliminaries to modeling vulnerabilities in Web browsers

Before examining each Web browser's vulnerability, discussion is needed about how and where we obtained vulnerability dataset, market share, the number of user dataset and relations to the significant factors which might effect on the discovery process.

First, Table 5.1 shows overall information of Web browsers that presents data obtained the number of vulnerabilities from the NVD and the market share data

[4]One of the software update mechanisms which does not require user's intervention.

Table 5.1: Web browsers' market share and vulnerabilities found (2010 October)

	IE	Mozilla/Firefox	Netscape
Market Share	59.18%	22.86%	0.63%
Vulnerability	589	600	44
Lines of code	2.30 MLOC (7.0.6)	3.2 MLOC (4.0)	N/A
Init. Release	Aug. 1995	Nov. 2004	Dec. 1994
Latest Version	8	3.6.10	9
	Opera	Safari	Chrome
Market Share	2.29%	5.36%	8.50%
Vulnerability	177	279	156
Lines of code	N/A	N/A	4 MLOC (Chromium8)
Init. Release	Dec. 1996	Jan. 2003	Sep. 2008
Latest Version	10.62	5.0.2	6.0.474

from Engineering Work Station (1996~1998)[5], Websidestory[6] (1999 for IE), the counter[7] (2000~2007), hitlink[8] (2008~2010). The web browser market shares and the total number of vulnerabilities are found in October 2010. Other sources of the browsers' usage data show similar values even though there are minor differences among the monitoring Web sites because they depend on the number of page hits. The IE share is about 60% and Firefox share is approximately 23%. As we can see from Table 5.1, for Web browsers with a lower percentage of the market share, such as Netscape, Opera, Safari and Chrome, the total number of vulnerabilities found is lower than the leading two Web browsers, IE and Firefox. This does not mean that those Web browsers are more secure, but merely that only a limited effort has gone into finding their vulnerabilities.

From the table, we observe that the vulnerability discovery rate is related more to the market share than to the period of usage. Even though the Mozilla Web browser was released earlier than Firefox and both source code size are similar,

[5] `http://ews.uiuc.edu/bstats/latest.html`
[6] `http://websidestory.com`
[7] `http://www.thecounter.com/stats`
[8] `http://marketshare.hitslink.com/browser-market-share.aspx?qprid=0`

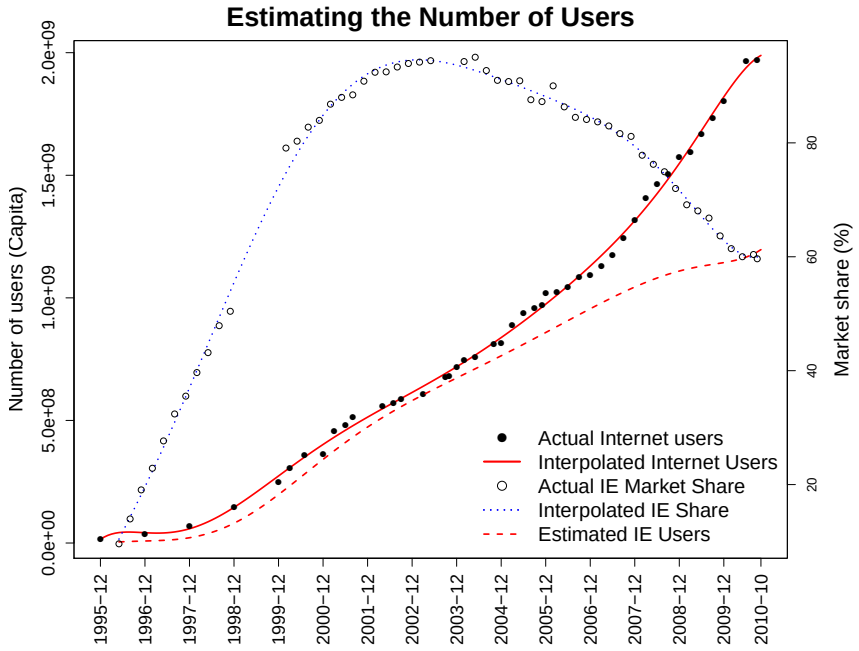


Figure 5.1: How to estimate the number of Web browser users

Firefox has a greater number of vulnerabilities than Mozilla because of its greater popularity. And Netscape, Mozilla's former and the first popular commercial Web browser, has only forty four vulnerabilities (See Figure 5.5). This shows that market share is more important factor than software age. Note that we combine the datasets for Mozilla Web browser and Firefox in the paper and use the term Firefox instead of Mozilla and Firefox. The release date in Table 5.1 for the Mozilla/Firefox column is for Firefox.

Figure 5.1 shows the process of how to estimate the number of users in each month for the effort-based model. Here, only estimating process for the IE user is shown for an example. In the figure, the left y-axis represents the number of users whereas the right y-axis indicates the market share. To determine the number of each Web browser users, the entire number of Internet users[9] and each browser's market share for each month are multiplied. Since the actual datasets for the number of Internet users and the each browser's market share are not available for every single

[9] `http://internetworldstats.com/emarketing.htm`

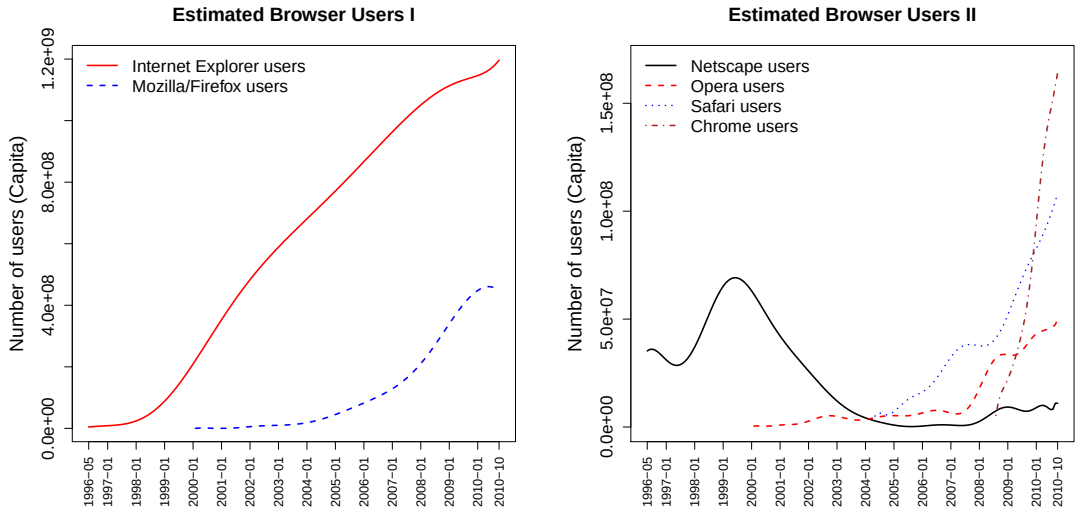


Figure 5.2: Estimating the number of users

month, the data have been interpolated. Figure 5.2 shows the estimated number of users for the six Web browsers. The plots are split due to the scales to be compared.

5.4 Aggregate vulnerabilities in Web browsers

In this section, we use vulnerability datasets for the six Web browsers to determine whether the vulnerability discovery trends are well described by the three discovery models.

IE has been the most popular Web browser, utilized by approximately 60% of the Internet users currently. It has dominated the Web browser market. Its market share had been over 90% from 2001 to 2005. This has made it a very attractive target for exploration and exploitation by malicious users. The problem was exacerbated by the integration of IE into Windows, unlike other Web browsers. This integration provided more functions, such as easier interface with other components. However, security analysts and experts consider the integration as a security disadvantage since IE is connected with a variety of Windows core components including Windows periodic update process. Another weakness of IE is using the non-standard features,

which did not follow the W3C standard[10]. 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 OSes are marketed with IE pre-installed.

Although Firefox 1.0 was released in November 2004, it did not gain significant recognition early. Its popularity increased because of its perceived better security, intuitive design and multi-tap features. Market share of IE have been declining since 2005 because of Firefox. Currently Firefox's market share is about 23%. Firefox had expended its market share rapidly. However, its popularity has led to a rising number of newly discovered vulnerabilities. Six hundred vulnerabilities have been reported in Mozilla/Firefox according to NVD, and is the highest number of vulnerabilities among the six Web browsers.

Figure 5.3 shows the cumulative number of vulnerabilities by month for time-based vulnerability discovery models. In the figure, the dashed lines indicate the fitted AML model and the solid line represents LVD fittings for the linear phase only.

In the beginning of the Time-based model, the slope of the curve for IE rose gently until 2000, after which the slope has generally remained steady. As we mentioned in Chapter 3, when LVD is introduced, we apply the model from October 2003 to October 2010 for IE. During this period, LVD provides a good fitting. However, there are problems to apply LVD to whole time period because the vulnerability found trend or rate is not a linear throughout the entire lifecycle as discussed in Chapter 3.2. From the view of the three phases of the vulnerability discovery process, IE does not appear to have yet entered the saturation phase. Rather, IE currently

[10] `http://www.w3.org/TR/`

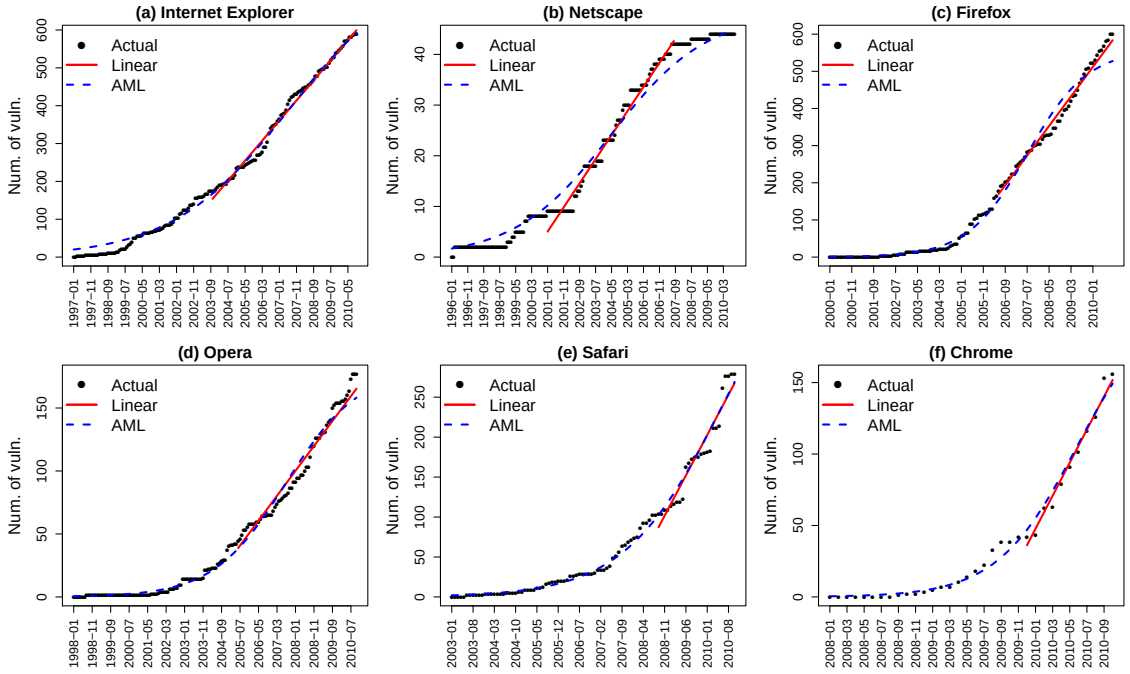


Figure 5.3: Fitting; Time-based model to browser vulnerabilities

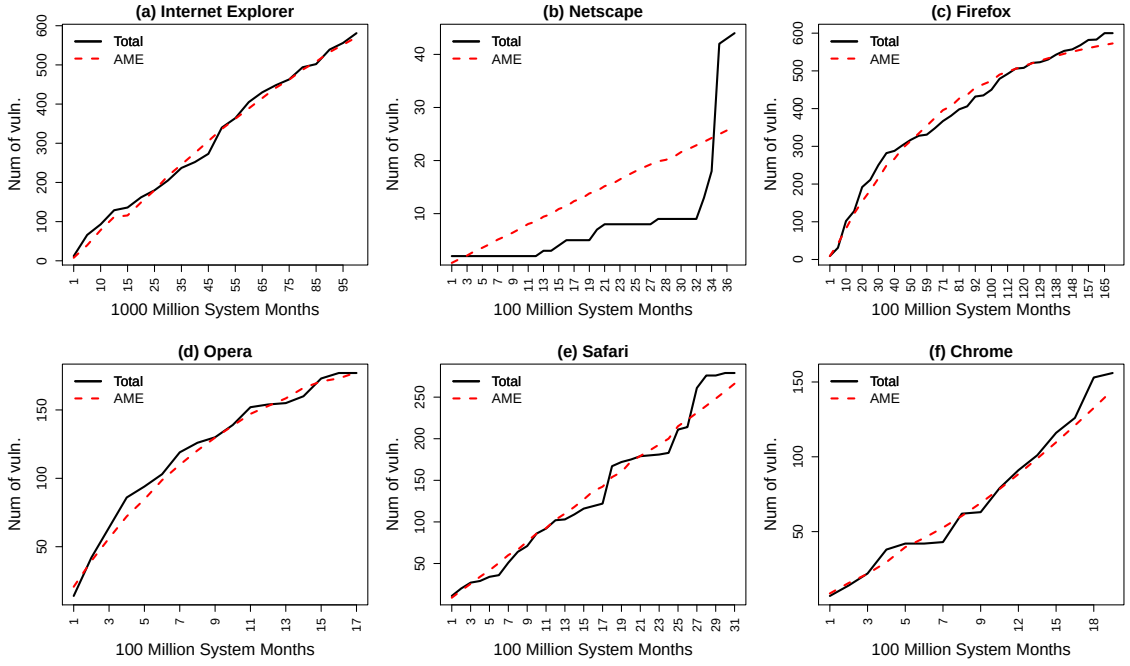


Figure 5.4: Fitting; AME Effort-based model to browser vulnerabilities

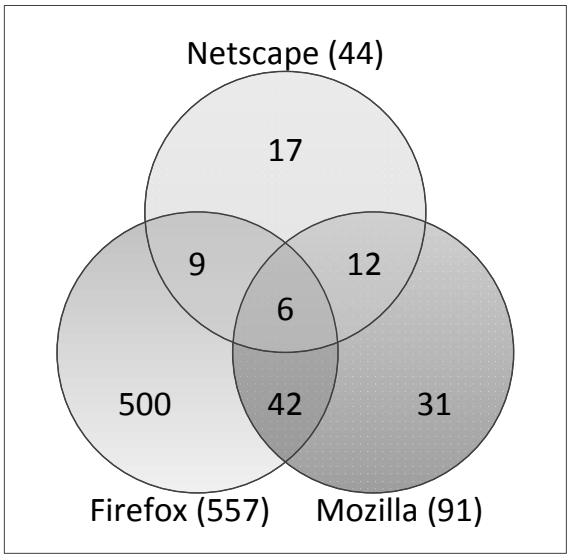


Figure 5.5: Shared vulnerabilities among Netscape, Mozilla and Firefox

still appears to be in the linear phase since the number of vulnerabilities is growing linearly in spite of having been on the market for years. The second transition point occurs on April 2011 based on the calculation of the second derivative of AML (Table 5.2). This may be because of its larger market share and possibly higher number of potential vulnerabilities. This suggests that vulnerability discovery for IE may continue at a significant pace in the near future as long as it is evolved.

Netscape Navigator originated from Mosaic Web browser dominated the middle of 1990s when the Internet usage had become popular. However, Netscape lost its market share from over 90% in middle of 1990s to less than 1% in October 2010 after IE was introduced which was pre-installed in Windows operating systems. Many of Netscape's vulnerabilities found in recent time are linked with code sharing with Mozilla/Firefox.

Figure 5.5 shows the number of vulnerabilities shared among the three browsers. The only Web browser reached the saturation phase is Netscape because its final version 9 was released on October 2007 and its developing line has been disconnected currently. Moreover, the browser's market share is also very tiny which helps for the vulnerability discovery pattern enters in the saturation phase. In Figure 5.3, AML

model's S-shape pattern matches with Netscape's vulnerability discovery trend, and LVD also fit well during the linear phase.

Firefox is the second most popular Web browser based on the market share. While there is still a considerable market share gap between IE and Firefox, this gap is shrinking slowly. Although Firefox is just seven years old, and its market share is about one-third of the market, the actual trend of Firefox's vulnerability discovery rate is still in the linear phase. Consequently, we can expect that more vulnerability will be found in the near future, the saturation phase is not likely to be reached soon in spite of the AML model shows that Firefox is entering the saturation phase as shown in Table 5.2. In Figure 5.3 (c), it is clearly showing that AML fitting line tends to saturate at the end of the examined period while the actual discovery trend keeps rising.

Opera Web browser is almost the same age with Netscape. It was initially released in December 1996. It is still being developed. It can be used not only in a desktop environment but also in mobile device such as a smart phone. Despite of its age, it has low market share and 177 vulnerabilities. The lower number of vulnerabilities is caused by lower number of users. As shown in Figure 5.3 (d), Opera's vulnerability trend follows the time-based models well. The partially applied LVD shows that Opera is in the linear phase.

Safari's market share is steadily increasing as the number of Mac OSX's users is growing. As a result, the number of vulnerabilities is also rapidly increasing from 2007. 122 vulnerabilities have been found in Safari from April 2007 to October 2010. In the same time period, Safari has gained over 3% of market share. Figure 5.3 (e) illustrates the movement of safari's vulnerability discovery pattern for the time-based models. Visually, AML model fits very well for the number of Safari's vulnerability. LVD is applied from September 2008 to October 2010.

In September 2008, Google introduced their Web browser Chrome. Currently Chrome's population is after Firefox, and many Firefox users tend to migrate to

Table 5.2: Phase test for aggregate vulnerability in Web browser

	LVD + σ at Data End	LVD - σ at Data End	# of Vuln. at Data End	Trans. Point 1	Trans. Point 2	Phase
IE	612.2743	586.6513	589	2003-10	2011-04	Linear
Netscape	75.0226	46.4334	44	2001-01	2007-08	Saturation
Firefox	597.9114	569.1612	600	2006-06	2009-01	Linear
Opera	172.6634	158.0932	177	2005-06	2009-11	Linear
Safari	282.7641	252.0902	279	2008-09	2012-04	Linear
Chrome	157.5507	146.2024	156	2009-12	2010-12	Linear

Table 5.3: χ^2 goodness of fit test - AML - Web browsers

	A	B	C	χ^2	$\chi^2_{critical}$	P-value	R^2
IE	0.00004	784.9644	0.0502	420.4922	183.9586	0.0000	0.9958
Netscape	0.00069	48.2233	0.5899	36.4841	159.8135	1.0000	0.9890
Firefox	0.00013	558.7199	1.3793	218.2364	123.2252	0.0000	0.9877
Opera	0.00028	180.9692	1.9801	68.5087	124.3421	0.9945	0.9879
Safari	0.00013	486.4047	0.5299	54.6760	92.8082	0.9242	0.9857
Chrome	0.00101	210.9290	2.7886	18.1839	32.6705	0.5752	0.9874

Table 5.4: χ^2 goodness of fit test - LVD - Web browsers

	S	k	χ^2	$\chi^2_{critical}$	P-value	R^2
IE	5.30463	-281.1064	47.6841	106.3948	0.9995	0.9907
Netscape	0.41772	-16.4810	30.6905	100.7486	1.0000	0.9816
Firefox	7.95682	-450.8506	27.7863	69.8321	0.9976	0.9876
Opera	1.97741	-139.1435	35.1977	83.6752	0.9987	0.9702
Safari	7.20844	-410.1666	32.7340	37.6524	0.1378	0.9334
Chrome	11.57370	-241.6291	4.1805	18.3070	0.9388	0.9802

Table 5.5: χ^2 goodness of fit test - AME - Web browsers

	B	λ_{VU}	χ^2	$\chi^2_{critical}$	P-value	R^2
IE	1151.5431	0.0000	15.4690	30.1435	0.6295	0.9939
Netscape	1151.5431	0.0000	175.7205	48.6023	0.0000	0.4988
Firefox	620.0800	0.0001	41.8256	46.1942	0.0552	0.9827
Opera	196.4820	0.0012	1.7070	27.5871	0.9999	0.9921
Safari	99682.8749	0.0000	35.5588	41.3371	0.1252	0.9781
Chrome	65994.0114	0.0000	9.9185	22.3620	0.6231	0.9858

Chrome. Its market share has 9.26% in November 2010. Respecting its market share growth, discovered vulnerabilities are speedily raising. With visual inspection as shown in Figure 5.3 (f), AML and LVD are fitted well for the Chrome's vulnerability datasets.

Table 5.3 shows the Chi-square goodness of fit tests for AML model on the six datasets along with the model parameters. The table is corresponding to the Figure 5.3. Here, for the all Chi-square goodness of fit tests, 0.05 has been chosen for the alpha level. This means that a model fitting is statistically acceptable when P-value is higher than 0.05 or a Chi-square statistic value is smaller than its corresponding critical value. P-values closed to 1 indicate a better fit. For more information about the Chi-square goodness of fit test, see Chapter 7.4.

Along with the hypothesis test, R^2 values are also given in the table. The Chi-square statistics for AML models in Table 5.3 are less than the critical values except for IE and Firefox. The results could be related to the fact that the two Web browsers do not have a significant number of vulnerabilities in their early time period which distorts logistic growth pattern causing low P-values. In fact, Chi-square test is very sensitive with small number of datasets applied. R^2 values, however, for AML models are significant range for all Web browsers. In all tables, values are rounded up at the proper decimal point in each case, so that value zero does not necessarily means zero, but a very small number.

Table 5.2 shows the results from the phase test for total vulnerability datasets for justification of applying LVD. It decides whether a vulnerability discovery process is one of the three phases in Figure 3.2. The standard deviation σ is calculated between the actual datasets and LVD fitting lines. It is assumed that a Web browser is in the linear phase if a number of vulnerability is higher than LVD - σ at the end data point as we discussed in Chapter 3.2. Furthermore, the table displays values associated with LVD model. Transition point 1 and 2 are the starting and end

points for the LVD model fittings. When the second transition point appears after the data end point month, October 2010, as shown in Table 5.2, the last available time point has been used instead. Only Netscape is in the saturation phase. Other five Web browsers are still in the linear phase. We note that the current number of Firefox's and Opera's vulnerability is higher than LVD + σ. This indicates that the vulnerability discovery rate of Firefox and Opera is higher than their other periods.

For Firefox, even though the calculated transition point 2 is January 2009 based on AML model, the last available time point has been used instead of the second transition point since the visual observation clearly shows that the browser is in the linear phase now. Unlike other Web browsers, Firefox has two somewhat distinctive linear phases. The first linear phase is from June 2006 to January 2008 and the second linear phase start around October 2008. The slope of second linear phase is steeper than the first phase suggesting that the constant vulnerability discovery rate has been increased. Opera also shows a similar pattern with Firefox; its second transition point is occurring before the data end point month while currently the browser displays the visually linear phase. For the entire Web browser described here, except Netscape, are expecting linear phase after the calculated second transition points in the table because of their new releases which will introduce new vulnerabilities, and that will extend the linear phase parts. Because of this reason, LVD might be a good model when a quick future estimation is needed.

Table 5.4 shows the fitting results for LVD model on the six datasets along with the model parameters, the two transition points from Table 5.2, and R^2 values. The table is corresponding to the Figure 5.3. For the LVD fittings, the Chi-square goodness of fit test and R^2 are significant for all the Web browsers because LVD is only applied to the linear phase, and as shown in Table 5.2, all the vulnerability datasets have clear linear phases. The noticeable thing in the table is the slope parameter value of S which represents the vulnerability discovery rate. As shown in

the table, Chrome claims the highest discovery rate followed by Firefox, Safari, IE, Opera, and Netscape, in the sequence of highest vulnerability discovery rate. With only this information, Firefox is more risky than IE. The constant k in LVD does not have a clear interpretation at the moment. All the datasets are well represented with LVD, but Safari, having several big sudden raises in the discovery processes, has low P-value although that still is statistically significant.

Figure 5.4 shows the model fitting for AME effort-based on the Web browsers. The y-axes represent the number of users while the x-axes are Million System Month, except IE. IE is based on 10 Million System Month due to the huge number of users. In the plots, the dashed lines represents AME model fittings while the other lines are showing the actual data based on the estimated number of browser users. Table 5.5 shows the corresponding goodness of fit test results with model parameters and R^2 values.

For the entire vulnerability datasets, the model fitting claims statistically significant results except Netscape. In Figure 5.4 (b), we suspect the big gap, between 32 million system months and 37 million system months, was caused very small number of user, but there is a significant number of vulnerabilities have been found which is reasoned by shared vulnerabilities between Netscape and Firefox. While producing good fitting shapes, IE, Safari, and Chrome project linear growth patterns rather an exponential shape. Especially the vulnerability data growth in Chrome appears to be superlinear as shown in Figure 5.4 (f). Only Firefox and Opera display general exponential growth pattern.

5.5 Vulnerability severity levels

Figure 5.6 displays the relative percentages for the cumulative number of vulnerabilities for the three severity classes in each month. It should be noted that the early part of the plot represents very few number of vulnerabilities, and thus, are

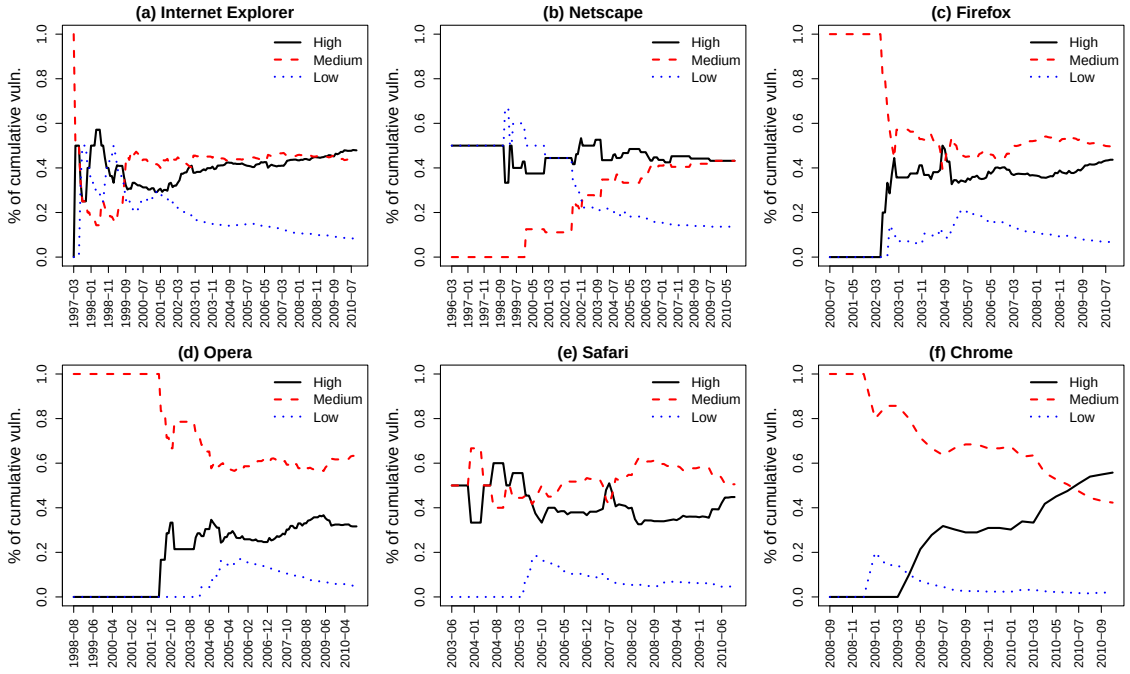


Figure 5.6: Web browsers' severity variation

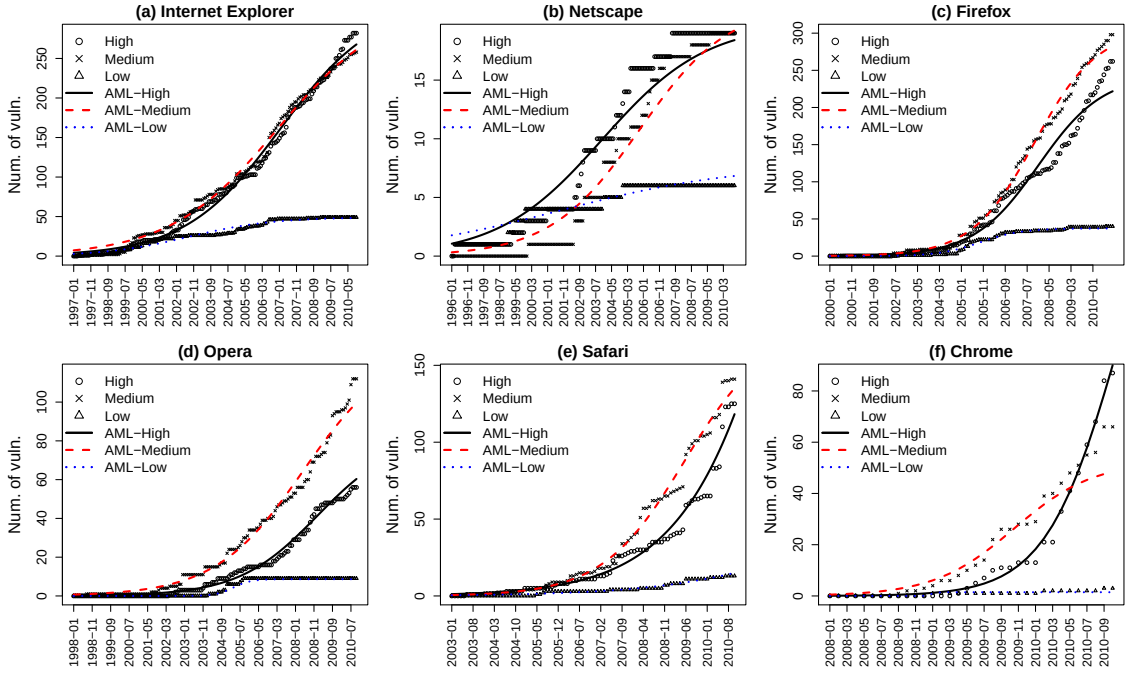


Figure 5.7: AML model fitting by severity - Web browsers

not significant to be analyzed as mentioned in Chapter 4.4. For example, the plots for Firefox up to July 2002 represent only five vulnerabilities total. The distribution of the severities for each Web browser's vulnerabilities shows its respective shapes as shown in Figure 5.6.

However, these six Web browsers have a slightly similar severity order. Generally, the largest portion of the vulnerabilities has been found in medium severity, followed by high severity vulnerabilities and then low severity level vulnerabilities. In general, the medium severity vulnerabilities tend to be detected earlier. Then high and medium vulnerabilities are converged. This would suggest a shift in vulnerability detection priorities. For all the datasets, the proportions of low severity vulnerabilities are getting decreased. Vulnerability finders may have it more rewarding to seek higher severity vulnerabilities. Unlike the other browsers, IE and Chrome have more number of high severity vulnerability now as of October 2010. Especially, the fraction of Chrome's high severity discovery rate is increasing and medium severity discovery rate is declining. In this section, we apply the three vulnerability discovery models to the six Web browsers categorized by the three severity levels.

Figure 5.7 shows the result of AML model fittings for the Web browsers' vulnerability datasets by severity. In general, the severity patterns of Web browsers follow the aggregate vulnerability discovery patterns except low severity vulnerability. With a visual inspection, high and medium severities are in linear phases whereas all the web browsers' low severity vulnerabilities already reached the saturation phase since low severity vulnerabilities have been rarely found as shown in Figure 5.7. This observation implies that the size of the high and medium severity level vulnerability pool is potentially larger than the low severity's one.

Table 5.6 represents the results of AML model fittings from Figure 5.7. For the IE model fitting, in Figure 5.7 (a), it is hard to distinguish between the high severity

Table 5.6: χ^2 goodness of fit test by severity - AML - Web browsers

	Severity	A	B	C	χ^2	$\chi^2_{critical}$	P-value	R^2
IE	High	0.0001	321.8864	0.2467	83.6089	167.5143	0.9999	0.9934
	Medium	0.0001	322.1045	0.1385	54.8313	163.1161	1.0000	0.9941
	Low	0.0008	49.0637	0.3410	48.6714	169.7113	1.0000	0.9683
Netscape	High	0.0015	19.8671	0.9238	21.3407	123.2252	1.0000	0.9789
	Medium	0.0017	21.5521	3.1536	13.4823	105.2672	1.0000	0.9829
	Low	0.0023	7.8000	0.4438	2.7827	91.6702	1.0000	0.8973
Firefox	High	0.0003	243.9349	2.8243	143.2500	112.0220	0.0002	0.9758
	Medium	0.0002	305.8589	2.3849	53.4783	118.7516	0.9998	0.9962
	Low	0.0029	37.2734	45.1934	13.0953	91.6702	1.0000	0.9880
Opera	High	0.0006	82.8344	6.0303	26.4327	177.3897	1.0000	0.9891
	Medium	0.0003	129.4416	1.4785	65.1859	103.0095	1.0000	0.9889
	Low	0.0243	8.6450	1.0588E+07	13.2575	89.3912	1.0000	0.9859
Safari	High	0.0001	835.9413	0.8918	45.1128	81.3810	0.9474	0.9683
	Medium	0.0004	174.7620	1.7617	30.3773	81.3810	0.9998	0.9883
	Low	0.0006	70.6205	2.1926	2.2477	36.4150	1.0000	0.9585
Chrome	High	0.0012	199.5697	16.9998	8.8385	24.9958	0.8613	0.9884
	Medium	0.0041	52.5900	2.3635	11.6600	30.1435	0.1965	0.9837
	Low	2.4818	1.4549	1.00E+20	6.4148	33.9244	0.9990	0.6582

Table 5.7: Phase test by severity in Web browser

	Severity	LVD + σ at Data End	LVD - σ at Data End	# of Vuln. at Data End	Trans. Point 1	Trans. Point 2	Phase
IE	High	282.2525	268.1453	282	2004-01	2010-02	Linear
	Medium	275.2449	258.6757	258	2003-06	2010-06	Saturation
	Low	58.51033	54.0490	49	1999-12	2005-04	Saturation
Netscape	High	24.93192	22.0913	19	2000-04	2007-06	Saturation
	Medium	24.85609	23.0898	19	2002-09	2008-11	Saturation
	Low	7.922938	6.6669	6	1996-03	2007-10	Saturation
Firefox	High	250.9451	223.4095	262	2006-06	2009-10	Linear
	Medium	228.0334	194.6153	298	2006-04	2009-05	Linear
	Low	97.44538	92.5295	40	2004-09	2006-10	Saturation
Opera	High	59.51561	54.1214	56	2006-09	2010-10	Linear
	Medium	106.8485	96.1103	112	2005-10	2010-10	Linear
	Low	38.28619	36.6850	9	2004-09	2005-10	Saturation
Safari	High	136.1285	119.6669	125	2009-10	2015-11	Linear
	Medium	143.1713	129.4177	141	2007-12	2010-11	Linear
	Low	12.68087	9.8171	13	2005-05	2016-04	Linear
Chrome	High	91.59483	86.8765	87	2010-05	2010-10	Linear
	Medium	66.10696	60.9693	66	2009-04	2010-05	Linear
	Low	2.808094	1.8542	3	2009-01	2010-10	Linear

vulnerability discovery process and medium severity discovery process. As a result, the model fitting parameter is fairly similar as shown in the table, especially, the second parameter B which represents the total number of vulnerabilities eventually found. Netscape model fitting is shown in Figure 5.7 (b), and high and medium severity levels in this browser shows relatively clear S-shaped growth patterns unlike other datasets. Subsequently, it claims P-values of 1 for all three severity levels.

It is observed that the high severity in Firefox is the only vulnerability discovery pattern which is producing an insignificant AML model fitting. The Chi-square statistic value is greater than the corresponding critical value or P-value is less than 0.05. Figure 5.7 (c) clearly shows why the fitting is rejected. The growth trend in Firefox high severity has been changed around in the middle of 2007, and the logistic growth behavior could not adjust on the dataset properly. Opera, in Figure 5.7 (d), is the only web browser claims all the P-values of 1 and its growth status is not in the saturation phase like Netscape. For Safari and Chrome, in Figure 5.7 (e) and Figure 5.7 (f), the vulnerability discovery rates for the high severity are surging up. Consequently, their estimated eventual numbers of vulnerabilities are higher than other browsers (See parameter B in Table 5.6).

Table 5.7 shows whether each severity of Web browsers is situated among the three phases. It shows the two transition points which are providing the start and end points for LVD model fittings. The calculations for the transition points are using the same method as that of the aggregate vulnerability as shown in Table 5.2. It points to the entire Web browsers' high severity are in the linear phase except Netscape. These results are the same as the trend of total number of each Web browser's vulnerabilities. For the medium severity, IE and Netscape claim saturation phases. However, with a visual observation, it is hard to say that the medium severity for IE is entered in the saturation phase. Safari and Chrome are the only two browsers having low severity vulnerability growths in the linear phase,

and it reflects that they are the two youngest Web browsers. Those interpretations show that the number of high and medium severity vulnerabilities reflect the entire recent trend of Web browsers.

Figure 5.8 is showing the LVD model fittings whereas Table 5.8 presents the corresponding model fitting results which shows model parameters, Chi-square statistic, critical values, and R^2. By and large, the model fittings are all statistically acceptable since the LVD model is only applied during the customized practical ranges. For the slope parameter S, both high and medium severities claim equal number of the highest parameter values across the vulnerability datasets. The high severities of IE, Safari and Chrome have currently steeper slope than other severities whereas the medium severities of Netscape, Firefox and Opera have currently steeper slopes then other severity levels. Hence, at the time being, high and medium severity vulnerabilities tend to be found equally likely. Firefox's high severity vulnerabilities derive two linear phase in the aggregate vulnerability as shown in the previous section.

Figure 5.9 shows AME model fitting for the six Web browsers by categorized in severity, and Table 5.9 provides the model fitting information. The AME model fittings by severity show similar visual patterns as shown in the aggregate AME model fittings. The model fittings are rejected for Netscape high and medium, Safari high, and Chrome high. For those rejected model fittings, the main reason is that unexpected gaps of number of users in the million system months. In Figure 5.9 (b), (e), and (f), it is clearly shown that there are sudden growth in the number of vulnerabilities, but not enough users. Other AME model fittings are all satisfactory with the exponential growth pattern. IE, Safari, and Chrome displays linear growth patterns, a similar pattern from the previous section. Figure 5.9 (f) shows that the origin for the superlinear behavior in Figure 5.3 (f) which is the high severity.

With some exceptions which did not claim significant P-values for the hypothesis test above, in general, model fittings are statistically significant for all other

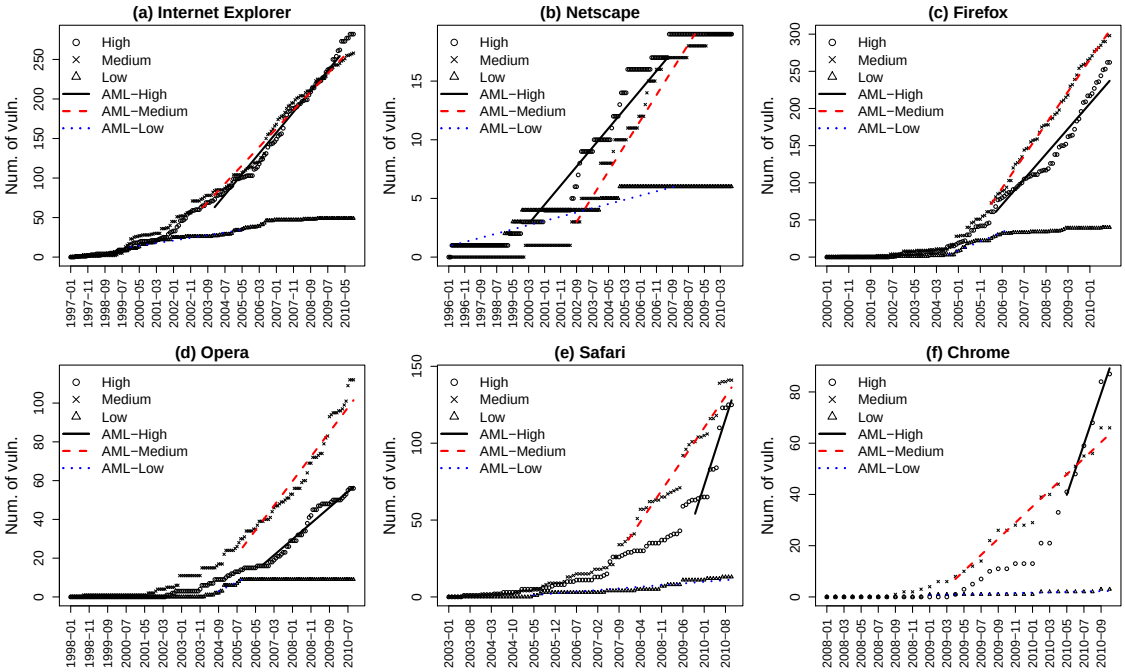


Figure 5.8: Linear model fitting by severity - Web browsers

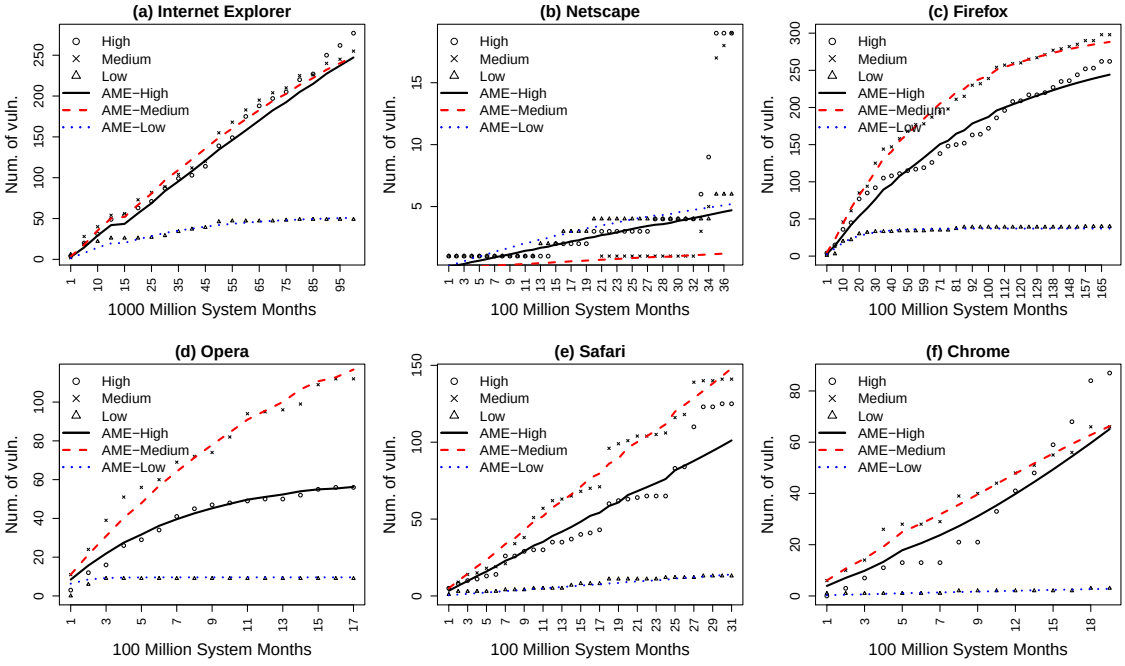


Figure 5.9: AME model fitting by severity - Web browsers

Table 5.8: χ^2 goodness of fit test by severity - LVD - Web browsers

	Severity	S	k	χ^2	$\chi^2_{critical}$	P-value	R^2
IE	High	2.6186	-159.4859	31.8807	93.9451	1.0000	0.9853
	Medium	2.3207	-118.2694	44.7641	106.3948	1.0000	0.9808
	Low	0.3362	0.4782	15.2769	83.6754	1.0000	0.8814
Netscape	High	0.1628	-5.4651	20.4447	108.6483	1.0000	0.9601
	Medium	0.2162	-14.5135	15.8435	95.0811	1.0000	0.9654
	Low	0.0360	0.8921	22.5349	167.5139	1.0000	0.9059
Firefox	High	3.3948	-204.1503	62.7041	69.8323	0.1471	0.9465
	Medium	4.2973	-150.5676	5.9074	52.1916	1.0000	0.9922
	Low	1.2522	-67.7928	10.0706	37.6524	0.9957	0.9317
Opera	High	0.8205	-69.5365	10.7654	66.3388	1.0000	0.9692
	Medium	1.2653	-93.3741	25.5848	79.0821	1.0000	0.9545
	Low	0.4735	-35.4354	1.9181	22.3616	1.0000	0.8433
Safari	High	6.1488	-450.0941	10.1592	21.0255	0.6024	0.8981
	Medium	2.9158	-137.7914	18.8125	48.6023	0.9835	0.9529
	Low	0.1541	-3.2405	21.5911	84.8208	1.0000	0.8691
Chrome	High	9.8793	-246.6599	0.4583	11.0700	0.9944	0.9807
	Medium	3.1422	-43.2963	3.8209	28.8689	1.0000	0.9802
	Low	0.0757	-0.2434	1.6291	32.6713	1.0000	0.7664

Table 5.9: χ^2 goodness of fit test by severity - AME - Web browsers

	Severity	B	λ_{VU}	χ^2	$\chi^2_{critical}$	P-value	R^2
IE	High	928.40941	0.00000	21.57271	30.14353	0.25151	0.99390
	Medium	460.00000	0.00001	8.95417	30.14353	0.96079	0.99200
	Low	53.33545	0.00003	5.24946	30.14353	0.99842	0.96051
Netscape	High	500.00000	0.00000	143.90840	48.60237	0.00000	0.48410
	Medium	278.65927	0.00000	728.89152	48.60237	0.00000	0.33233
	Low	9.15298	0.00023	3.88619	48.60237	1.00000	0.87051
Firefox	High	299.10249	0.00010	42.25998	46.19426	0.10603	0.97067
	Medium	315.59998	0.00015	7.56368	46.19426	1.00000	0.99427
	Low	37.04550	0.00066	3.79820	46.19426	1.00000	0.95253
Opera	High	62.17602	0.00140	0.84058	26.29623	1.00000	0.98789
	Medium	133.79102	0.00103	2.49422	27.58711	0.99923	0.98617
	Low	8.92502	0.01071	0.01382	26.29623	1.00000	0.98628
Safari	High	19911.35341	0.00000	54.92395	41.33714	0.00117	0.92250
	Medium	5498.08186	0.00001	18.47033	41.33714	0.88849	0.98281
	Low	541.06205	0.00001	1.99010	28.86930	1.00000	0.94959
Chrome	High	19400.71590	0.00000	37.02251	22.36203	0.00022	0.95314
	Medium	86.21377	0.00070	2.74885	22.36203	0.99707	0.98188
	Low	4.28516	0.00054	0.62181	22.36203	1.00000	0.79635

cases. Nevertheless, the R^2 values and visual observations indicate that the fits are practically acceptable.

5.6 Prediction capability

In this section, prediction capabilities have been examined. Figure 5.10 shows the prediction errors for the three VDMs on the six Web browsers, and Table 5.10 shows the initial parameters used for the three models. As we discussed in the previous chapter, while time-based models, AML and LVD, have data points in each month, the effort-based model, AME, has the data values only after every significant number of installations had occurred. In the figure, the starting points for comparing prediction errors on the three models are chosen when cumulative users exceed every ten millions and one million for Internet Explorer and others respectively. Since the prediction errors among the models can be comparable only when the models have data points for the equivalent time points all together, and since the time-based models have more data points than the effort-based model, the comparisons are conducted on top of the effort time which is a proportion representing the number of current users over the number of users on the second transition point or the data end point which is October 2010. Because of that, when the effort time axis is mapping to the calendar time axis, unequal intervals are shown between the adjacent calendar time points. Entire data points in effort time axis have corresponding calendar time data points whereas the vice versa is not true. The linear model has data points only between the first transition point to the second transition point or end of the data point as shown in Table 5.2.

To predict the number of vulnerabilities for the target point in each data point in Figure 5.10, all the available partial records from the beginning upto the second transition point or the data end point are regressed by the models to estimate model parameters on the current point. Then each model with the estimated parameters

for each time point is used to predict the final number of vulnerabilities at the end of the time period. The estimated numbers for each time point are compared with the actual numbers of vulnerabilities to calculate normalized estimations.

In Figure 5.10, prediction errors approach zero percent error lines as more data becomes available in general. For IE, in Figure 5.10 (a), AML produces big error margins at the beginning. The behavior is related to the fact that the Web browser does not have a significant number of vulnerabilities by year 2000 which distorts logistic growth pattern as shown in Figure 5.3 (a). After some time later, AML has been calm down with the other two models. AME claims relatively well predictions from the beginning due to the exponential growth pattern which agrees with Equation 3.7. Linear model also performs well because Internet Explorer shows a clear linear phase after the first transition point.

Prediction errors for Netscape are shown in Figure 5.10 (b). Across the prediction period, AML swings several times along with the sudden increases as shown in Figure 5.3 (b). Also it has been observed that AML performs well during the saturation phase. AME shows a bit monotonic error rates. That is probably because the number of user growth rate had followed exponential growth model well until right before end of the inspection period in Figure 5.4 (b), then suddenly, at the end of the examined period, there was a huge jump which crashes the expectation of the exponential growth pattern. At the beginning of the linear phase, there is no vulnerability found at all. As a result, the linear model does not work because the linear regression will not work under that circumstance. However, as soon as some vulnerabilities had started to be detected, the model performs getting better.

The prediction capability for AML, in Figure 5.10 (c) performs poorly at the beginning, and the reason for that seems the same with IE case. As shown in Figure 5.3 (c) and Table 5.2, AML claims that Firefox had escaped from the linear phase on January 2009 in spite of the clear visual linear trend. This probably prevents for

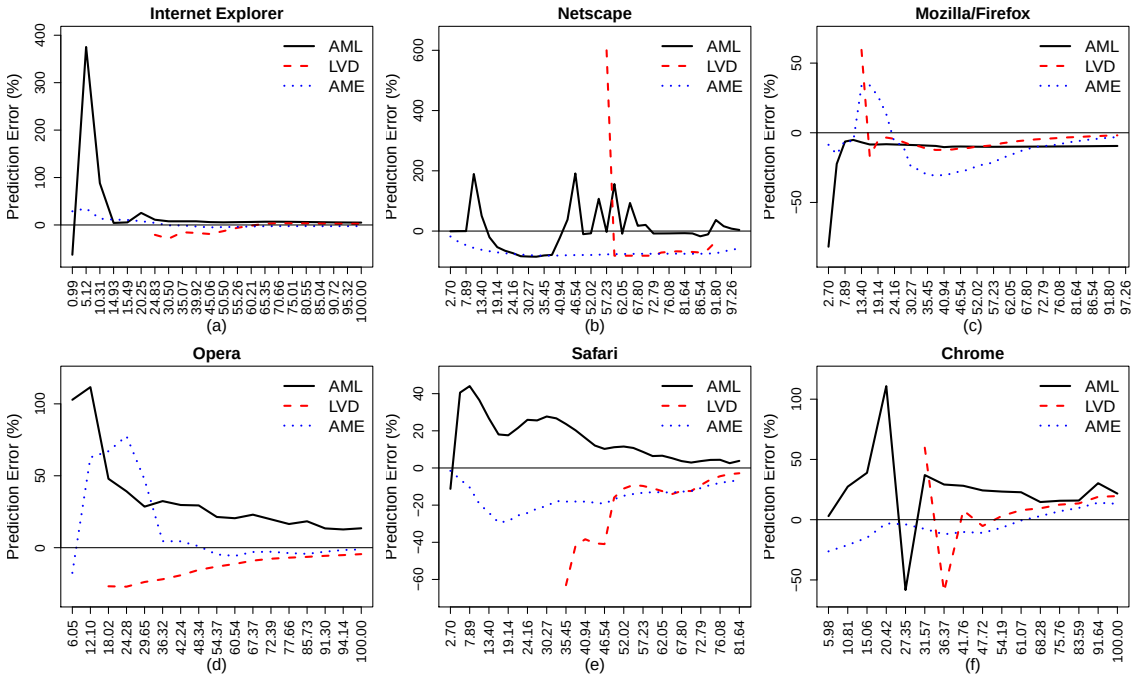


Figure 5.10: Prediction errors; x-axis represents effort time (%)

Table 5.10: Initial parameter values for prediction

	AML			LVD		AME	
	A	B	C	S	k	B	λ_{vu}
IE	3.68E-05	883.5	0.050222	5.36744	145.58818	1178	6.8647E-06
Netscape	0.00069	44	0.589954	0.431477	1.838506	44	6.2583E-06
Firefox	0.00009	600	0.7	8.172146	155.7482	600	0.00013894
Opera	0.000284	354	2.034972	2.10153	32.48029	354	0.00098227
Safari	0.000126	837	0.52993	7.550766	74.75691	837	0.00000085
Chrome	0.001013	258	2.788681	12.09091	21.27271172	258	1.1181E-06

AML to approach to the zero percent error line at the end of the period in Figure 5.10 (c) unlike other models. In the AME prediction error line, top and bottom instabilities are generated by the corresponding up and down exponential behavior residuals shown in Figure 5.4 (c). The linear model performs relatively well across the prediction period except the beginning part which has a sudden raise in Figure 5.3 (c).

With Opera, AML also shows a big error line at the beginning of the prediction phase which apparently caused by the same issue in IE or Firefox. AME shows quite a big error line at the beginning as well, and it seems due to the gap between

the model behavior and actual number of users as described in Figure 5.4 (d). Meanwhile, even though there are some sudden raises during the linear phase in the actual data when the linear regression is performed, residuals caused by the raises are canceled each other for the regression. As a result, the model performs getting better as the number of data points goes up.

Safari has a more odd growth behavior than other browsers in both time and effort based models as shown in Figure 5.3 (e) and Figure 5.4 (e). There are several relatively big sudden raises during the linear phase. The biggest victim, due to the peculiar behavior, is the linear model because the model simply cannot decide the future trend under the situation. On the other hand, saturation speeds to the zero percent error line, for the other two models, relatively stable.

In Figure 5.3 (f), there are some gaps between the fitting and the actual growth line around at the end of 2009 to the beginning of 2010, which implies that AML should not estimate well the future trend around at that time period. The two time-based models suffer around the time shown in Figure 5.10 (f) while AME predicts better than the other two models. Notice when the effort time approaches to the 100 percent effort time, the prediction errors a bit deviate from the zero percent prediction error line. That is because at the end of the observation period, there is a sudden growth from the actual data in both Figure 5.3 (f) and Figure 5.4 (f).

Here, prediction capabilities among the models are compared using average error (AE) and average bias (AB) (Malaiya et al., 1992) which measure how well a model predicts during different phases of the time period, and quantifies the general tendency to overestimate or underestimate the number of vulnerabilities respectively. The comparison among the three models is somewhat iniquitous since AML and the linear model are based on time whereas AME is founded on the number of users. Also the linear model is only applied during the linear phase. However, we consider comparing the models' prediction capabilities is meaningful despite of the unfairness.

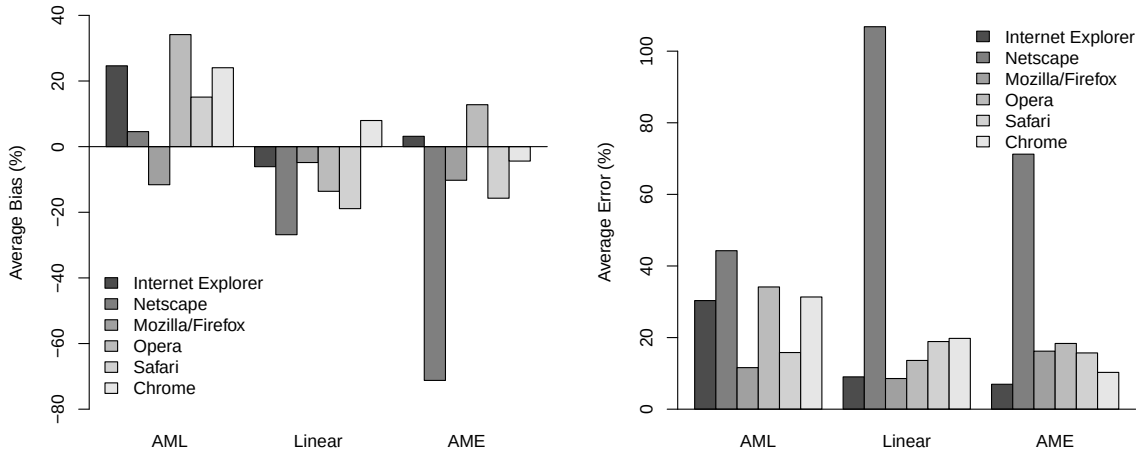


Figure 5.11: (a) Average Bias and (b) Average Error

Table 5.11: Average bias & average error - Web browsers

	AB			AE		
	AML	LVD	AME	AML	LVD	AME
IE	0.24627	-0.06106	0.031485	0.303382	0.090232	0.069638
Netscape	0.045764	-0.26835	-0.71233	0.442659	1.068288	0.712328
Firefox	-0.11605	-0.04842	-0.10246	0.116055	0.085611	0.16207
Opera	0.341385	-0.13613	0.127768	0.341385	0.136127	0.183577
Safari	0.15096	-0.18893	-0.15714	0.158216	0.188928	0.157142
Chrome	0.240448	0.079625	-0.04399	0.313397	0.197736	0.102786

Equation 5.1 and 5.2 are AB and AE, where n is the total number of data points during the prediction period, and Ω is the actual number of vulnerabilities at target point whereas Ω_t is the estimated number of vulnerabilities at time t.

$$AB = \frac{1}{n}\sum_{t=1}^{n}\frac{\Omega_t - \Omega}{\Omega} \tag{5.1}$$

$$AE = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{\Omega_t - \Omega}{\Omega}\right| \tag{5.2}$$

Figure 5.11 plots the calculated AB and AE, and Table 5.11 shows the corresponding exact values. Figure 5.11 (a) shows that the linear and AME tend to underestimate while AML is likely to overestimate. Since the models have strong bias, they could be good candidates for recalibration just like the Web server cases

in the previous chapter. The reason why Netscape has a quite different look between AB and AE is that the two time-based models have big swings across the zero percent error line. In Figure 5.11 (b), AML, LVD, and AME perform the best once, twice, and three times respectively although some of them look no significant difference each other.

5.7 Discussion

Secure Science Corp.[11] reports a single phishing group collecting access information for 13,677 accounts by installing malicious code by exploiting an unpatched vulnerability. The exploitation techniques and tools utilized are no longer the exclusive possession of experts, since many of them are now widely available and can be relatively easy to use.

Web browsers' vulnerabilities are used as a medium of spreading viruses and worms. For example, Nimda, which use the buffer overflow vulnerability, affects the Microsoft Internet Explorer harshly. The vulnerability discovery 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, quantitative projections can be used to allocate resources needed for fast patch development.

In the chapter, we have examined the six Web browsers' vulnerability datasets to determine whether the vulnerability discovery process tends to follow specific patterns and if these patterns can be modeled. The results show that when the aggregate number of vulnerabilities is examined, the AML model fit the datasets well for most of Web browsers, as shown in Figure 5.3 and Table 5.3. The model was found to fit even when vulnerabilities are partitioned by severity levels (Figure 5.7 and Table 5.6). This suggests that the model can potentially be used to estimate

[11] `http://voices.washingtonpost.com/securityfix/2006/04/real_world_impact_of_internet_1.html`

the number of vulnerabilities expected to be discovered in a given future period, and what severity level distributions is likely.

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

We note that there are sufficient data for high and medium severity vulnerabilities, and the fit is quite good. This suggests that the model can be used to project the expected number of high severity vulnerabilities, which may be much higher interest than others.

LVD model can be applied only to the specific period since vulnerability discovery rate should not be linear for the entire software lifecycle. Our result shows the software vulnerability discovery rate based on calendar time is close to the logistic distribution, in general.

The AML and LVD model used here does not require the use of market share data. The effort-based AME model can potentially generate more stable projections because it can remove fluctuations due to variability in the discovery effort with time. Found vulnerabilities are proportion to the number of users. Thus, the market share is the main factor to affect the number of detected vulnerabilities. In this chapter, we notice the exceptional case, Netscape, which had been found more vulnerabilities after losing large market share because of shared vulnerabilities with Firefox.

Examining the current vulnerability discovery trends for the six Web browsers, five appear to be in the linear phase except Netscape (a new version of Netscape is no longer available in the market). It can be expected that more vulnerabilities will be discovered in all five.

When comparing browser security, we need to keep in mind that the vulnerability discovery rate in the near future may be more important than the vulnerabilities

already discovered in the past. Other factors should be considered including severity levels and quick availability of patch releases. Some of security experts have considered IE to be less secure. For the reason, they are pointing to the integration of IE into Windows and Active X. Secunia[12] reports that IE has more unpatched vulnerabilities than Firefox. However, when Firefox's popularity continues to increase, it attracts more attempts to discover its vulnerabilities. Now, Mozilla/Firefox has more number of vulnerabilities than IE.

5.8 Summary

This chapter examines the trends in vulnerability discovery in Web browsers and explores the applicability of quantitative models for the number of vulnerabilities. These models give us an insight into the vulnerability discovery process. The results show that the AML model generally tracks the available data well. We found that the fit is significant when aggregate vulnerabilities are divided into classes (for example, vulnerabilities arising due to high and medium severity), provided there are sufficient vulnerabilities in a class. The model can, thus, be used to project the classes of vulnerabilities that are more likely to be encountered, and consequently can be used to make testing more effective. It is also possible to project the high severity vulnerabilities that may be expected in the near future.

The results indicate that the models originally proposed and validated for operating systems (Alhazmi and Malaiya, 2005) are applicable not only to Web servers but also to Web browsers.

[12] `http://Secunia.com/gfx/Secunia2008Report.pdf`

Chapter 6

WEB BROWSER SECURENESS WITH RESPECT TO CVSS SCORE

In this chapter, we will examine and compare the secureness of Web browsers with respect to the CVSS base score. Consequently, this chapter could be considered as an extended chapter from the previous chapter.

Table 6.1 shows that the current market share is way more evened compare to back in the days, although Internet Explorer still occupies more than half of the market. Meanwhile, the browser vendors' patch works are somewhat in inverse proportion to their market shares except Safari. Apple seems not work hard for patching compare to the other vendors.

When organizations' security or individuals' privacy is a top priority, users are frequently asking themselves questions such as what is their Web browsers' historical reputations related to the security, how much their Web browsers are inherently more vulnerability-free than others, is the vendor provides patch update quickly

Table 6.1: Number of cumulative vulnerabilities with initial release date, market share, and patch rate

	IE	FF	OP	SF
# of Vulnerabilities*	575	525	159	213
Release date	Aug. 1995	Nov. 2004	Dec. 1996	Jan. 2003
Market Share†	60.65%	24.52%	4.65%	2.37%
Patch rate‡	80.21%	96.73%	98.07%	77.06%

*Total number of vulnerabilities. Datasets are minded at NVD on April 2010

†`http://marketshare.hitslink.com/report.aspx?qprid=0` on March 2010

‡Average value; `http://secunia.com` on March 2010; Secunia data starts from February 2003

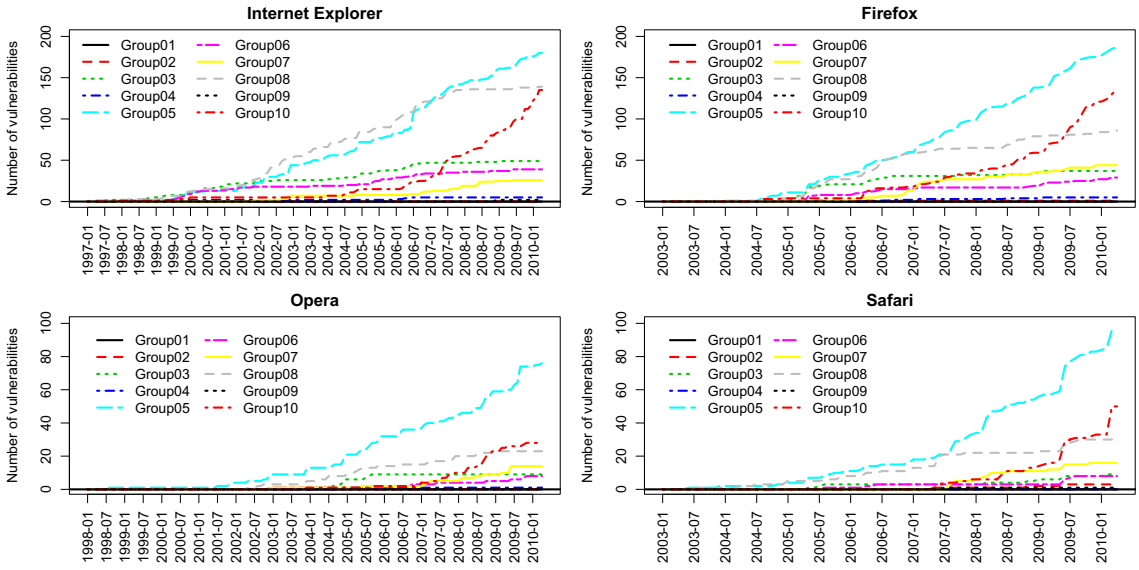


Figure 6.1: Grouped CVSS base scores; vulnerabilities having their scores of [0.0, 1.0] belong to Group 1, having their scores of (1.0, 2.0] belong to Group 2, ..., having their scores of (9.0, 10.0] belong to Group 10.

and properly, and if the system is compromised what could be the total cost they need to pay.

Common Vulnerability Scoring System (Mell et al., 2007) provides some of those answers directly and indirectly. For more detail descriptions of CVSS, see the Chapter 2.2.3. You might want to review the subsection for the later part of this chapter due to the abbreviations used.

This chapter quantitatively analyzes all the elements in CVSS base score for the four Web browsers of Internet Explorer (IE) , Firefox (FF), Opera (OP) and Safari (SF). The results show that exploitation aftermath is getting worse and when there are extra complexities and authentications, most of the exploitation would be avoidable.

6.1 Comparing trends of grouped CVSS scores

According to the NVD, the score range from 0 to 3.9 corresponds to low severity, 4.0 to 6.9 to medium severity and 7.0 to 10.0 to high severity as we mentioned in

Table 6.2: Number of cumulative vulnerabilities – Grouped CVSS base score

	Group number									
	1	2	3	4	5	6	7	8	9	10
IE	0	0	49	5	180	39	26	139	2	135
FF	0	1	37	5	187	29	44	86	0	136
OP	0	0	9	1	76	8	14	23	0	28
SF	0	3	9	0	96	8	16	30	1	50

Table 6.3: Number of cumulative vulnerabilities – Grouped CVSS by exploitability sub-scores

	Exploitabilty Group number									
	1	2	3	4	5	6	7	8	9	10
IE	0	0	0	11	88	0	3	0	193	280
FF	0	1	0	6	62	0	2	0	222	232
OP	0	0	0	1	12	0	0	0	68	78
SF	0	1	0	6	10	1	0	0	133	62

Table 6.4: Number of cumulative vulnerabilities – Grouped CVSS by impact sub-scores

	Impact Group number									
	1	2	3	4	5	6	7	8	9	10
IE	0	0	225	0	19	0	186	0	0	145
FF	0	0	224	0	20	0	140	0	0	141
OP	0	0	85	0	10	0	35	0	0	29
SF	0	0	108	0	5	0	47	1	0	52

Chapter 4.4. Figure 6.1 shows the grouped CVSS base scores for the Web browsers. In the figure, the CVSS base score groups are split into the ten groups evenly as follows:

- Group 1: $0.0 \leq$ CVSS base score ≤ 1.0
- Group 2: $1.0 <$ CVSS base score ≤ 2.0
- $\vdots$
- Group 10: $9.0 <$ CVSS base score ≤ 10.0

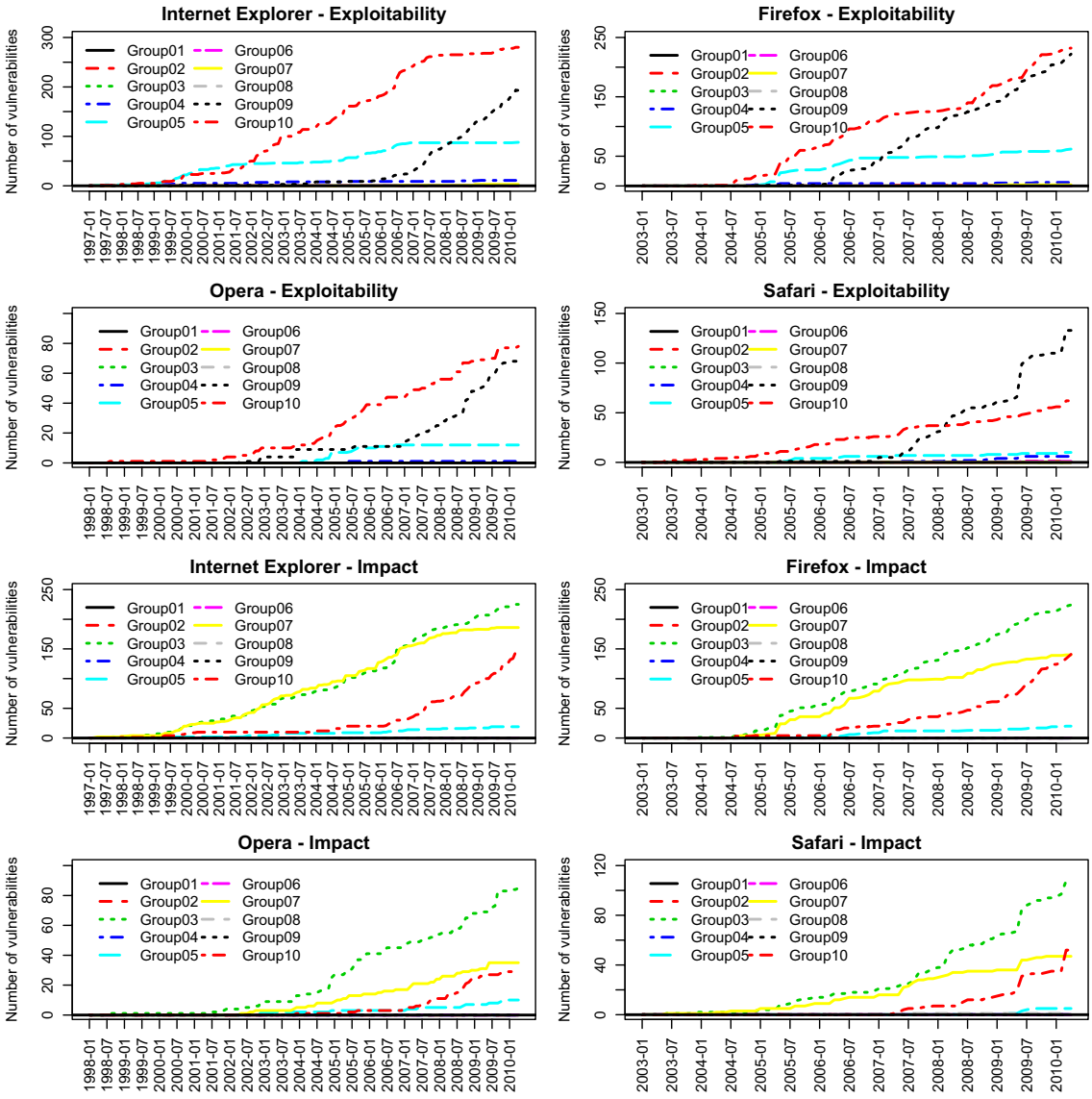


Figure 6.2: Grouped exploitability and impact sub-scores; vulnerabilities having their scores of [0.0, 1.0] belong to Group 1, having their scores of (1.0, 2.0] belong to Group 2, ..., having their scores of (9.0, 10.0] belong to Group 10.

Table 6.2 shows the number of vulnerabilities of the grouped CVSS base scores. For the four Web browsers, Group 5s have the biggest portion of the number of vulnerabilities. For Firefox, Opera, and Safari, the second biggest group is Group 10 followed by Group 8. Rankings of Group 10 and Group 8 had been switched around 2009. For Internet explorer, Group 8 and Group 10 are about to change their position sooner or later. The figure shows that the growth rate of Group 10 is

steeper than other groups, which indicates the vulnerability severity level is getting worse. No vulnerability is falling into Group 1.

Figure 6.2 shows growth rates of grouped scores for the exploitability and impact sub-scores, and Table 6.3 and 6.4 show the number of vulnerabilities for the explitability and impact sub-scores respectively. For the exploitability plots, in Internet Explorer, Firefox, and Opera Web browsers, Group 10s have the biggest number of vulnerabilities followed by Group 9 whereas Group 9 is the biggest group followed by Group 10 for Safari Web browser. This trend indicates that vulnerabilities are compromised remotely with very little or none of authentication processes. On the other hand, in Group 1, Group 2, Group 3, Group 6, Group 7, and Group 8, there is very small number of vulnerabilities or none of them are found.

For the impact sub-scores, all the Web browsers show the very similar pattern; Group 3 is the highest followed by Group 7 or Group 10, and Group 10 is about to catch Group 7 up or Group 10 already overtook Group 7 in some cases. No vulnerabilities are falling into Group 1, Group 2, Group 4, Group 6, Group 8, or Group 9. There is one exception in Safari which is CVE-2007-3514 having impact score of 7.8 which is Group 8.

6.2 Exploitability and impact sub-metrics

Figure 6.3 and Figure 6.4 show the growth rate of the three elements in the exploitability metric group and the three elements in impact metric group respectively for the four Web browsers, and Table 6.5 and 6.6 show the corresponding number of vulnerabilities for exploitability and impact sub-scores respectively. In Figure 6.3, in general, methods how the vulnerability is accessed are very similar across the four browsers. For the access vector, most of the time, vulnerabilities are compromised from the remote networks, and some are attacked locally. The observation suggests that organizations need to improve their network security related equipment.

Table 6.5: Number of cumulative vulnerabilities – Exploitability metrics

	Exploitability metric								
	AV			AC			Au		
	L	A	N	H	M	L	M	S	N
IE	10	0	565	89	196	290	0	4	571
FF	6	1	518	64	224	237	0	3	522
OP	1	0	158	12	68	79	0	0	159
SF	6	1	206	12	136	65	0	1	212

Table 6.6: Number of cumulative vulnerabilities – Impact metrics

	Impact metric								
	C			I			A		
	N	P	C	N	P	C	N	P	C
IE	169	257	149	185	245	145	149	268	158
FF	190	186	149	125	259	141	185	191	149
OP	78	49	32	41	89	29	71	57	31
SF	91	62	60	73	87	53	90	63	60

It is shown that the access complexities are not complex enough to protect systems from attackers. Most of the time, vulnerabilities are compromised by low access complexity followed by medium complexity. However, vulnerabilities with highly complex accesses are also frequently observed. For the authentication, almost every time, the vulnerabilities do not requires any authentication process. Some are asking a single authentication process, but vulnerabilities with multiple authentications never found.

In Figure 6.4, for Internet Explorer, when vulnerabilities are compromised, most of the time, confidentiality, integrity, and availability tend to be partially impacted. Even though the number of vulnerabilities leading to the category of *complete* is still the lowest, the growth rate is the highest, so that in the near future, *complete* seems will overtake the *none*. For Firefox, in the confidentiality and availability impact cases, *partial* and *none* categories seems occurred with the similar frequencies. However, like Internet Explorer, the growth rate of *complete*

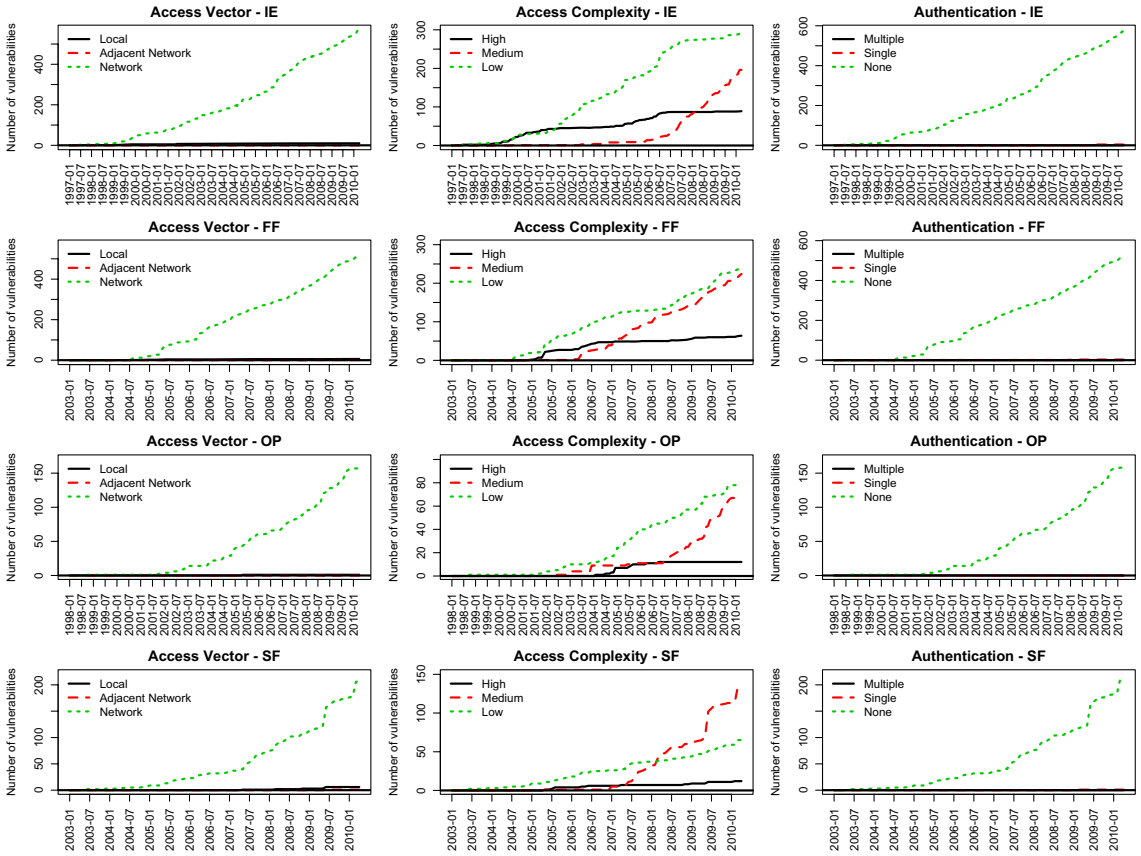


Figure 6.3: Exploitability matric group; Access Vector, Access Complexity, and Authentication.

category is higher than others, so that it seems like in the near future the category will catch up the other two. For the integrity impact, the *partial* is dominant.

Opera and Safari shares somewhat similar pattern across the three impact elements. In many cases, vulnerabilities are not affecting the confidentiality impact (*none*). Then they affect partially followed by *complete*. For the integrity impact, vulnerabilities are falling into the *partial* category followed by *none*; the *complete* positions the third. For the availability impact, *none* and *partial* occurs neck and neck followed by *complete*. In Opera, the growth rate of *complete* seems to be slowdown whereas the growth rate of *complete* in Safari tends to be accelerated.

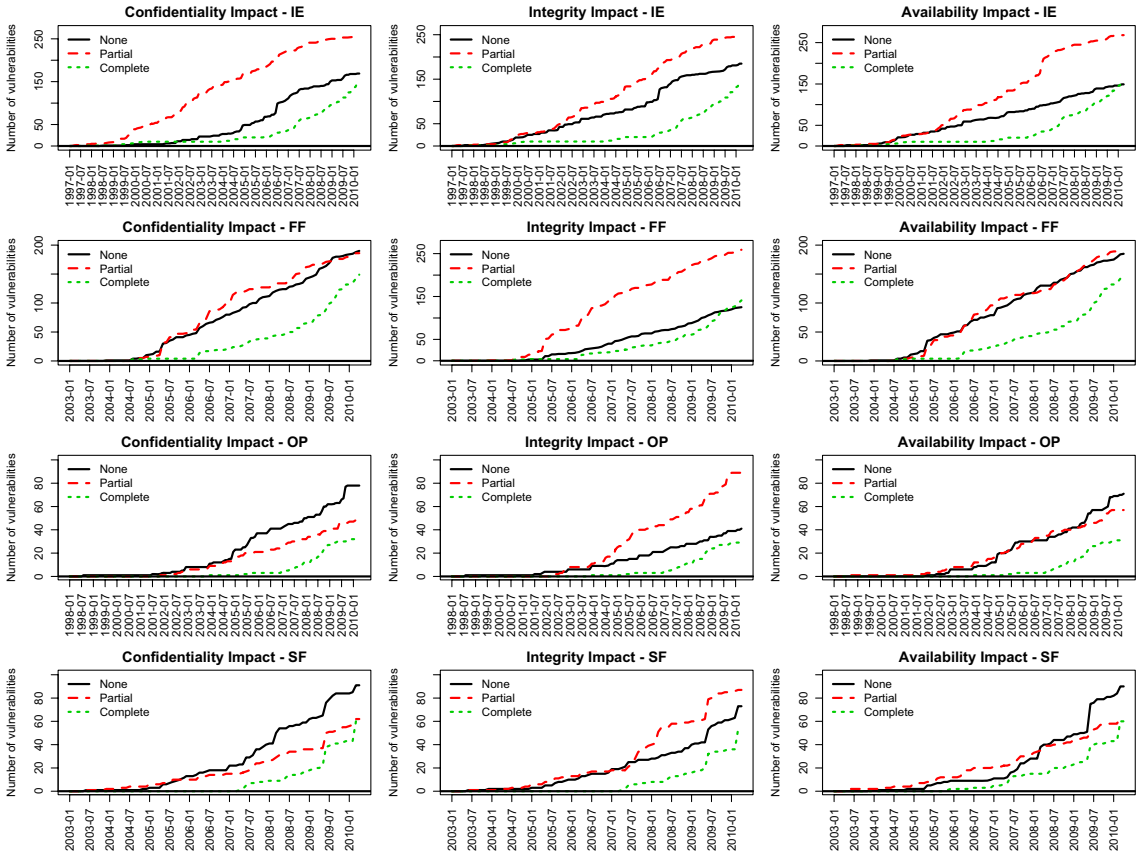


Figure 6.4: Impact metric group; Confidentiality Impact, Integrity Impact, and Availability Impact.

6.3 Combination of exploitability and impact

Figure 6.5 shows all the possible combinations between the elements from the exploitability and impact metric groups. The reason why some Web browsers have more categories than others at the graphs is that simply some Web browsers do not have any vulnerability for those categories. It is clearly observed that some combinations have more incidents than others.

For the all Web browsers, vulnerabilities are occurred most of the time at the categories of (AV_N, AC_L, Au_N), (AV_N, AC_M, Au_N), or (AV_L, AC_L, Au_N) which means majority of vulnerabilities are compromised (remotely with low access complexity and no authentication process), (remotely with medium access

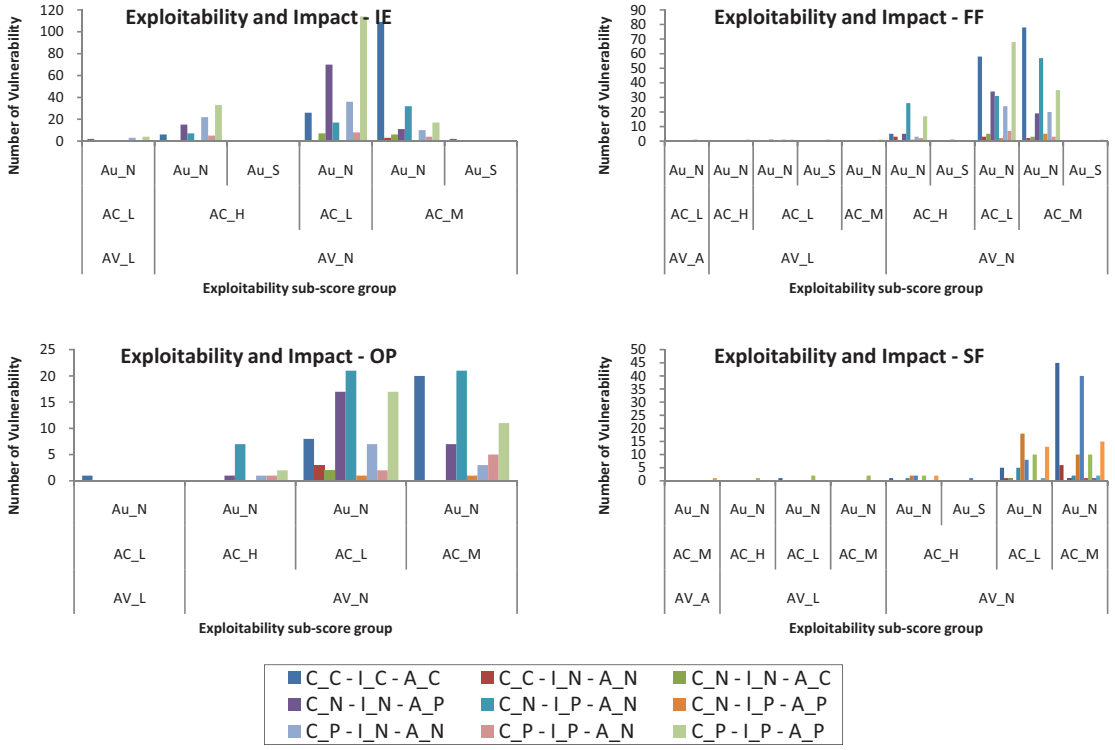


Figure 6.5: Relationship between exploitability and impact sub-score groups; all the possible combinations.

complexity and no authentication process), or (remotely with low access complexity and no authentication process) respectively.

Table 6.7 shows the top three individual combinations having the biggest number of vulnerabilities for each Web browsers. It is evidently shown that when access vector is network (N), and there is no authentication process (N), majority of vulnerabilities are compromised. Also, there is no vulnerability having a high access complexity. Frequently, a compromised vulnerability let attackers completely gain IT asset in terms of confidentiality, integrity, and availability: the triple Cs in the table.

6.4 Summary

The chapter analyzed the base scores from Common Vulnerability Scoring System for the four popular Web browsers quantitatively: Internet Explorer, Firefox,

Table 6.7: Top three high frequency combinations

	Freq.	Exploitabilty			Impact		
		AV	AC	Au	C	I	A
IE	114	N	L	N	P	P	P
	109	N	M	N	C	C	C
	70	N	L	N	N	N	P
FF	78	N	M	N	C	C	C
	68	N	L	N	P	P	P
	58	N	L	N	C	C	C
OP	21	N	L	N	N	P	N
	21	N	M	N	N	P	N
	20	N	L	N	C	C	C
SF	45	N	M	N	C	C	C
	40	N	M	N	N	P	N
	18	N	L	N	N	N	P

Table 6.8: Summary of CVSS version 1 and version 2 base scores for the experimental data (Scarfone and Mell, 2009)

	mean	median	s.d.	skew	below 5.0	exactly 5.0	above 5.0
V1	5.1	5.6	2.62	0.11	45%	0%	55%
V2	6.6	6.8	1.91	-0.05	25%	10%	65%

Table 6.9: Most common CVSS version 2 vectors (Scarfone and Mell, 2009)

Freq. (%)	Exploitabilty			Impact		
	AV	AC	Au	C	I	A
2662 (24.2)	N	L	N	P	P	P
1527 (13.9)	N	M	N	N	P	N
999 (9.1)	N	M	N	P	P	P
896 (8.1)	N	M	N	C	C	C
743 (6.7)	N	L	N	C	C	C
577 (5.2)	N	L	N	P	N	N
443 (4.0)	N	L	N	N	N	P
251 (2.3)	L	L	N	C	C	C
240 (2.2)	N	L	N	N	N	C
217 (2.0)	L	M	N	C	C	C

Opera, and Safari. The base score in the scoring system measures how a vulnerability will directly affect an IT asset as the degree of losses in confidentiality, integrity, and availability. Furthermore, the score captures how the vulnerability is accessed and whether or not extra conditions are required to exploit it.

The results show that, almost all the time, vulnerabilities are compromised from remote area at no authentication required systems. This suggests for organizations to enhance their network security related facilities, and also to add authentication process in their Web sites. The result also reveals that exploitation aftermath is getting worse.

An analogous study had been conducted by Scarfone and Mell (2009) in 2009 based on 11,012 CVEs between June 20^{th}, 2007 and April 30^{th}, 2009. They examined CVSS version 2 scoring system in depth without software categorizations. Their main goal was to determine how effectively version 2 had addressed the version 1 problems focused on base metrics. The problem was there were not enough diversities among the produced scores. The comparisons between the versions are shown in Table 6.8. It indicates that version 2 should generally produce higher scores than version 1. Moreover, they analyze the percentages of combinations in the base metrics which shown in Table 6.9. The top 10 most common vectors comprised over 77% of all vulnerabilities. The combinations are pretty much the same pattern from those of Web browsers' in Table 6.7.

Chapter 7

MODELING SKEWNESS IN VULNERABILITY DISCOVERY

Recent studies have shown that the S-shaped AML vulnerability discovery model often fits better than other models and demonstrates superior prediction capabilities for several major software systems (Alhazmi and Malaiya, 2006b; Alhazmi et al., 2007; Alhazmi and Malaiya, 2008). However, the AML model is based on the logistic distribution, which assumes a symmetrical discovery process with a peak in the center of discovery process. Hence, it could be expected that when the discovery process does not follow a symmetrical pattern, an asymmetrical distribution based discovery model might perform better.

In the chapter, the relationship between performance of S-shaped vulnerability discovery models and skewness in target vulnerability datasets is examined. To study the possible dependence on the skew, alternative S-shaped models based on the Weibull, Beta, Gamma and Normal distributions are introduced and evaluated. The models are applied to datasets from the eight major software systems: Red Hat Linux, MAC OSX, Windows XP, Windows Server 2003, Apache Web server, IIS, Firefox and Internet Explorer.

The applicability of the models is examined using two separate approaches: goodness of fit test to see how well the models track the data, and prediction capability using average error and average bias measures. It is observed that an excellent goodness of fit does not necessarily result in a superior prediction capability. The

results show that when the prediction capability is considered, all the right skewed datasets are represented better with the Gamma distribution based model. The symmetrical models tend to predict better for left skewed datasets, the AML model is found to be the best among them.

7.1 Motivation

Voit and Schwacke (2000) have found that the S-distribution, which is one of the S-shaped distributions, provides accurate representations for frequency datasets regardless their skewness, so this flexibility makes the distribution an ideal candidate for Monte Carlo analyses. A comparison of several S-shaped growth models in biomedical engineering has been conducted by Albaiceta et al. (2007). Among the vulnerability discovery models, which have emerged recently, the AML model is the only S-shaped model proposed thus far.

The AML model uses, as shown in Chapter 3.1, a logistic vulnerability discovery process which assumes a symmetric shape around the peak discovery rate value. However, for many datasets, the upper and lower percentiles may not be equidistant from its median and, thus, the assumption of a symmetrical discovery may not hold precisely. Moreover, Barua and Srinivasan (1987) have shown that, in risk analysis, ignoring the skewness can lead an analysis to be inaccurate. Thus, the AML model needs to be evaluated for cases with asymmetrical discovery patterns.

This chapter, first, introduces four more S-shaped models, and examines whether the performance of S-shaped VDMs is related to their underlying probability density functions (PDF) and skewness in target vulnerability datasets. For example, a VDM based on the Gamma distribution (Figure 7.4 (e)), which belongs to the family of right-skewed distributions, might represent the right skewed datasets (Figure 7.2 (c)) better than the S-shaped models whose underlying PDF is symmetrical such as the AML model or the Normal distribution based VDM (Joh and Malaiya, 2010b).

7.2 Skewness

Table 7.1: Skewness and total number of vulnerabilities from Figure 7.1. Datasets from NVD on 2009

	RHL	OSX	Windows XP	Server 2003
Skewness	0.4539	-7.064	-0.1394	0.2636
Number of Vulnerabilities	228	512	302	219
	Apache	IIS	IE	Firefox
Skewness	-0.3420	1.0034	-0.5436	0.0236
Number of Vulnerabilities	133	140	501	388

The Skewness static measures the magnitude and the direction of the skew. It characterizes the degree of asymmetry of a distribution around its mean value (Kenney and Keeping, 1962). The skewness values are calculated using the following equation.

$$Skewness = \frac{n}{(n-1)(n-2)} \sum (\frac{x_i - \bar{x}}{s})^3 \tag{7.1}$$

where n is the number of data points, x_i is the i^{th} data value, $\bar{x}$ represents the mean value, and s is the standard deviation for the datasets examined. As Figure 7.2 shows, a positive or right skewness indicates a distribution with a tail extending toward more positive values, and a negative or left skewness indicates a distribution with a tail extending toward more negative values. As the skewness statistic departs further from zero, a positive value indicates the possibility of a positively skewed distribution or a negative value indicates the possibility of a negatively skewed distribution.

Figure 7.1 shows the number of vulnerabilities for each month for the eight software systems. The starting points of the x-axes are January of the year when the first vulnerability was reported for each dataset. Skew values calculated by Equation 7.1 are not affected by time points having no data value at the starting

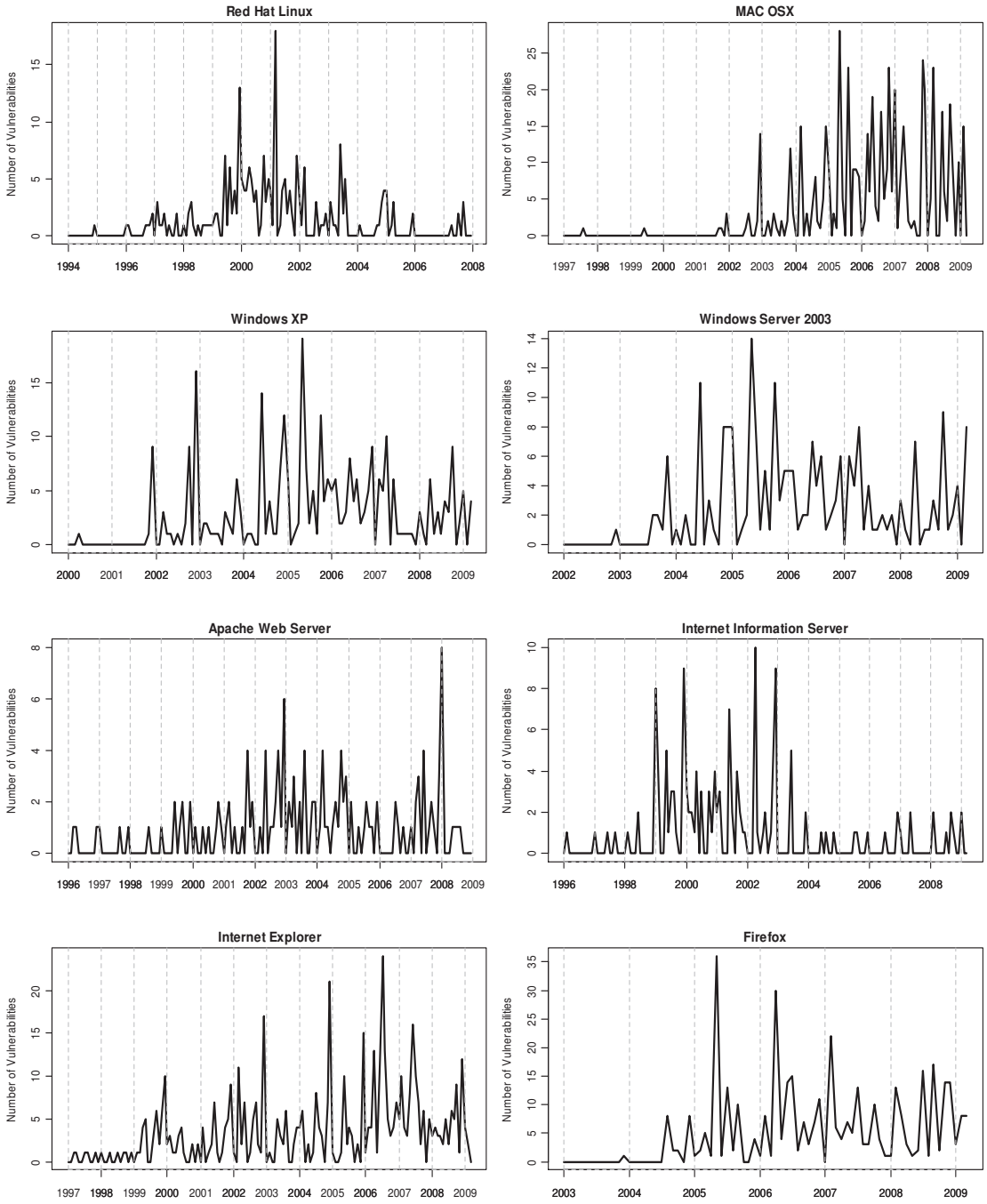


Figure 7.1: Run chart for the number of vulnerabilities in each month.

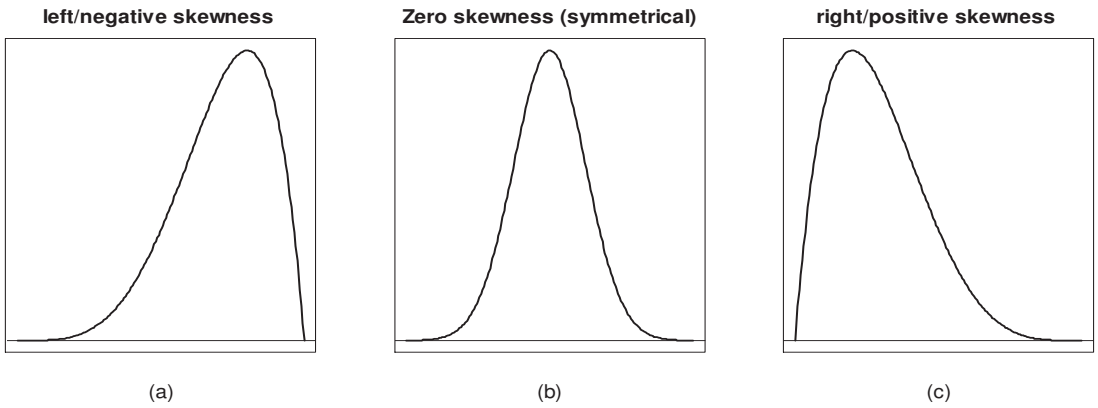


Figure 7.2: Skewness for S-shaped probability density functions. When a tail is located at the left (right) side, it is called skewed left (right). Left (right) skewness can be said also negative (positive) skewness.

period in Figure 7.1. Table 7.1 gives the corresponding skewness values for each of the software systems along with the cumulative number of vulnerabilities.

It should be noted that the calculated skew values tend to change only a little when the data, near the tail sides with only a few data points, is truncated. For example, the skew is -0.3532 for OSX if we skip the first five years and start from year 2002, thus the dataset is still skewed left.

Often skewness cannot be clearly observed visually when the data is noisy and not continuous. This would cause the visual observation to be misleading. In Figure 7.1, visually, the data for Red Hat Linux may appear symmetrical, due to the peaks around year 2000 and 2001, however, with significant number of vulnerabilities on the right side of the plot, the calculated value finds the data to be skewed right. Thus, skewness should be evaluated using a formal mathematical expression to avoid any visual illusions

Bulmer (Bulmer, 1965) has suggested a rule of thumb for skewness: i) if an absolute value of skewness is greater than 1, the distribution is highly skewed ii) if an absolute value of skewness is between 1/2 and 1, the distribution is moderately skewed iii) and if an absolute value of skewness is less than 1/2, then the distribution is approximately symmetric. According to the rule, in Table 7.1, IIS is highly skewed

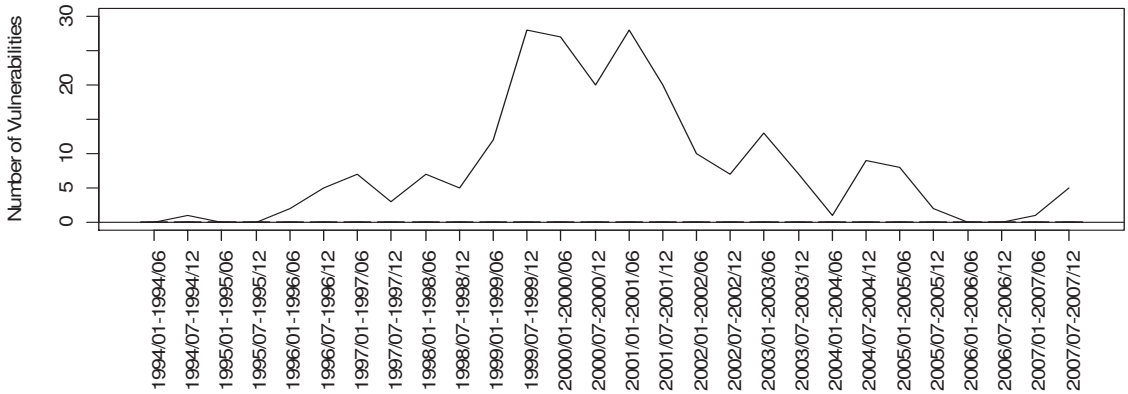


Figure 7.3: Number of vulnerabilities grouped by every six month for RHL; bell shape is clearly observed.

right, OSX and IE is moderately skewed left, and the others are approximately symmetric.

7.3 S-shaped vulnerability discovery models

In this section, the four new S-shaped vulnerability discovery models are introduced: Weibull, normal, Beta and Gamma distribution based models. The Weibull distribution based VDM model is proposed by Kim (2007) who pointed out that the asymmetrical S-shaped VDM might be alternative to the symmetrical AML model. The other three S-shaped models are introduced here for the first time [1].

Choosing S-shaped models is natural when saturation is inevitable. Sir Richard Stone (Stone, 1980), a Nobel laureate, wrote about the S-curve growth models in economics as "whatever the future may hold, growth must at some stage be faced by limit beyond which it cannot go". Figure 7.4 (a) shows the three phases of the S-shaped models. The learning phase is from the introduction of the system until the onset of sustained growth as a consequence of increasing popularity. It is followed by the linear phase, and this is when most of the vulnerabilities are to be discovered. Finally, saturation phase will eventually occur. The last phase might not

[1]Okamura et al. (2009) briefly mention about Gamma and Log-Normal distributions tested for Windows XP in the context of patch release time.

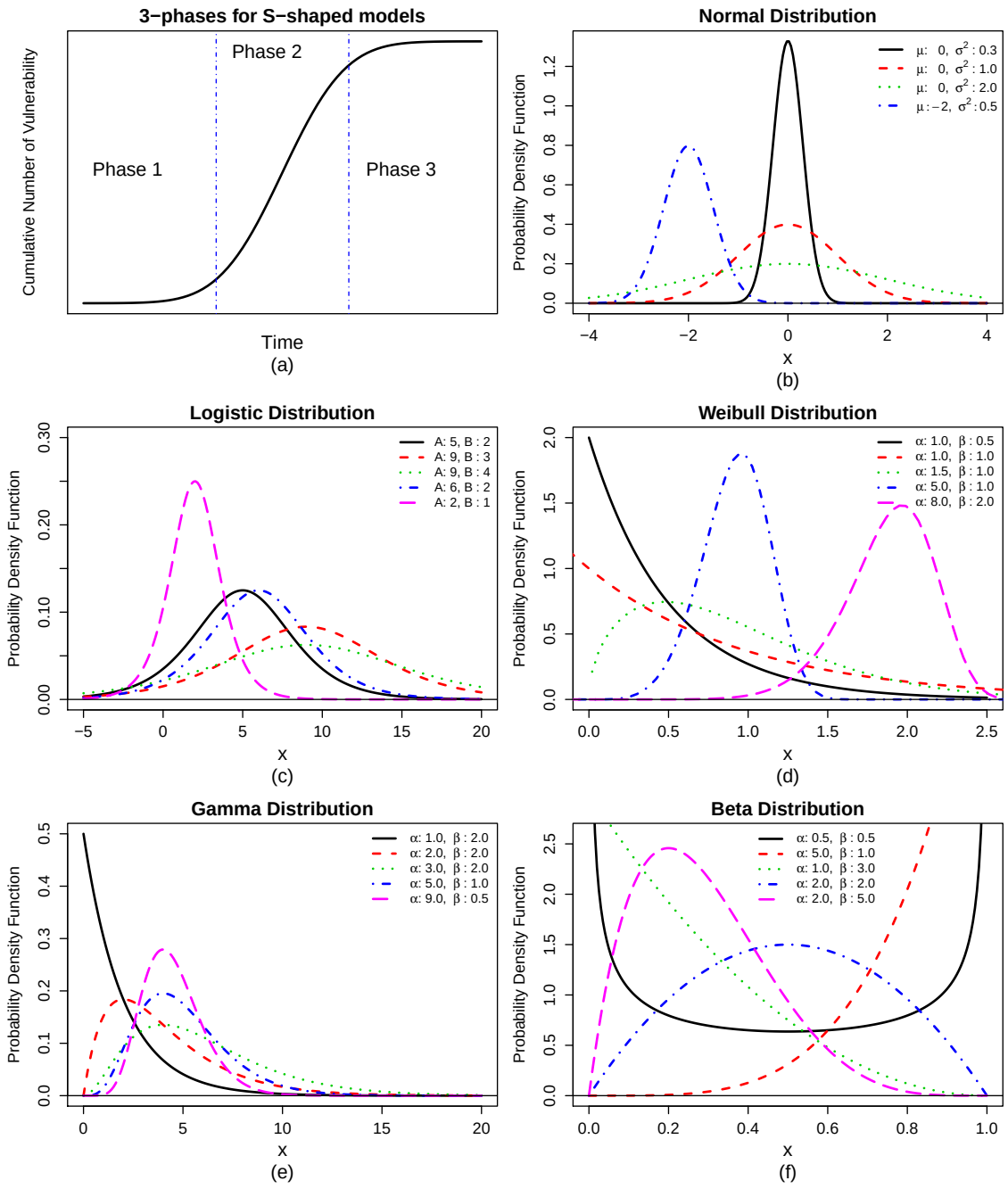


Figure 7.4: Three phases for S-shaped models (a) and the five probability density functions (b, c, d, e, and f) for normal, logistic, Weibull, Gamma, and Beta probability distributions respectively. The five probability density functions are applied for vulnerability discovery models here.

be observed as long as a significant number of vulnerabilities are still undiscovered. The transition points between the phases are defined mathematically by Alhazmi and Malaiya (Alhazmi and Malaiya, 2006b) for the AML model. See Chapter 3 for the detail about AML.

Figure 7.3 illustrates why S-shaped models should be more accurate than non-S-shaped models for the vulnerability discovery process, provided the time period involved is sufficiently long. In the figure, the vulnerability discovery behavior is approximately bell shaped, and thus, the cumulative number of vulnerabilities found will form an S-curve like 7.3 (a). Since the AML model is based on the logistic probability distribution, it has a symmetrical PDF as shown in 7.4 (c) with a skewness value of zero. In the figure, A is a location parameter which determines the location or shift associated with the distribution and B is a scale parameter; the larger the scale parameter, the more spread out the distribution.

7.3.1 Normal Distribution based model

In statistics, the Normal distribution (or Gaussian distribution) is a continuous probability distribution that describes a dataset that clusters around an expected value. It is a symmetrical distribution with zero skewness like the AML model. Figure 7.4 (b) represents the PDF for the Normal distribution. In the figure, μ is a location parameter and σ is a scale parameter. It is similar to the logistic distribution used by AML model in shape but has lighter tails in both sides. Hence, if a vulnerability dataset has fewer incidences at the beginning and at the end of a discovery process, then the Normal distribution based VDM might perform better than the logistic distribution based model.

Equation 7.2 gives vulnerability discovery rate for the normal distribution based VDM and Equation 7.3 gives the corresponding cumulative model. μ is a location parameter and s is a scale parameter. When t approaches infinity in Equation

7.3, the right hand side converges to γ. Hence, γ represents the total number of vulnerabilities that will eventually be discovered.

Like the AML model, the time t varies from the negative infinity to the positive infinity. Hence, unlike asymmetrical distributions, AML or Normal distribution based VDM could possibly characterize the pre-release vulnerability discovery, represented using the folded VDM (Kim, 2007; Younis et al., 2011) if we take time $t = 0$ as the software release time. This effect cannot be modeled by the asymmetrical distributions which are generally defined for the duration from $t = 0$ to the plus infinity.

$$\omega_{Normal}(t) = \frac{\gamma e^{-\left(\frac{(t-\mu)^2}{2s^2}\right)}}{\sqrt{2\pi}s} \tag{7.2}$$

$$\Omega_{Normal}(t) = \frac{\gamma}{2}\left[1 + erf\left(\frac{(t-\mu)^2}{2s^2}\right)\right] \tag{7.3}$$

$$where\ erf(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$$

7.3.2 Gamma distribution based model

The Gamma distribution is a two-parameter continuous probability distribution with a shape parameter α and a scale parameter β as shown in 7.4 (e). This distribution is only defined for the t values from 0 to the positive infinity. The Gamma distribution is often used for a probabilistic model for waiting times, it arises naturally in processes for which the waiting times between Poisson distributed events are relevant[2]. For example, in life testing, the waiting time until death is a random variable which is frequently modeled using the Gamma distribution (Hogg and Craig, 1978).

Equation 7.4 is the vulnerability discovery rate for the Gamma distribution based VDM. Equation 7.5 is its corresponding cumulative model. It has the shape parameter $\alpha > 0$, the scale parameter $\beta > 0$ and a parameter γ which signifies the

[2]http://mathworld.wolfram.com/GammaDistribution.html

total number of vulnerabilities that would eventually be found. As α increases, the Gamma distribution becomes more symmetrical. The distribution is defined for $t > 0$. Since the Gamma distribution belongs to the family of right-skewed distributions it might perform better when a vulnerability dataset is right-skewed. When time t is sufficiently large, many vulnerability discovery datasets should appear right-skewed because the number of vulnerabilities found will drop gradually, which would cause a tail on the right side.

$$\omega_\gamma(t) = \frac{\gamma}{\Gamma(\alpha)\beta^\alpha} t^{\alpha-1} e^{-\frac{t}{\beta}} \tag{7.4}$$

$$where \quad \Gamma(\alpha) = \int_0^\infty t^{\alpha-1} e^t dt$$

$$\Omega_\gamma(t_0) = \gamma \int_{t=0}^{t_0} \frac{1}{\Gamma(\alpha)\beta^\alpha} t^{\alpha-1} e^{-\frac{t}{\beta}} dt \tag{7.5}$$

7.3.3 Weibull distribution based model

The Weibull distribution is one of the most widely used lifetime distributions in reliability engineering. Li et al. (2004) have empirically shown that Weibull model is better in comparison to other reliability models for defect occurrence across a wide range of software systems. Zhou and Davis (2005) have shown the applicability of Weibull distribution to explain the failure process data from the several open source projects by fitting the distribution to the time related bug reporting patterns. Figure 7.4 (d) gives its PDF where the parameter α represents the shape parameter and β is the scale parameter. The Weibull distribution based VDM model was proposed by Kim (2007) recently.

Equation 7.6 represents the vulnerability discovery rate for the Weibull distribution based VDM, and Equation 7.7 gives the number of cumulative vulnerabilities. Here, α determines the shape of the software vulnerability discovery rate. When α is approximately 3, the shape is symmetrical. The curve has negative skewness when $\alpha > 3$ and positive skewness when $\alpha < 3$. The parameter β controls the time

scale. The parameter γ represents the total number of vulnerabilities that would eventually be discovered. The calendar time t is defined from 0 to the positive infinity just like Gamma distribution based VDM.

$$\omega_{Weibull}(t) = \frac{\alpha\gamma}{\beta}\left(\frac{t}{\beta}\right)^{\alpha-1} e^{-\left(\frac{t}{\beta}\right)^{\alpha}} \tag{7.6}$$

$$\Omega_{Weibull}(t) = \gamma e^{-\left(\frac{t}{\beta}\right)^{\alpha}} \tag{7.7}$$

7.3.4 Beta distribution based model

Figure 7.4 (f) shows the behavior of probability density function for the Beta distribution. Since the x-axis is fixed from 0 to 1, it only has two positive shape parameters α and β and no scale parameter. The Beta distribution can accommodate both forms of skewness, positive and negative. Thus, it is frequently used when skewness serves a core decision maker (Moitra, 1990).

$$\omega_{\beta}(t) = \frac{\gamma t^{\alpha-1}(1-t)^{\beta-1}}{B(\alpha,\beta)} \tag{7.8}$$

$$where \quad B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$$

$$\Omega_{\beta}(t_0) = \gamma \int_{t=0}^{t_0} \frac{t^{\alpha-1}(1-t)^{\beta-1}}{B(\alpha,\beta)} dt \tag{7.9}$$

Equations 7.8 and 7.9 give the discovery rate and the cumulative number of vulnerabilities for Beta distribution based VDM. Both α and β are positive shape parameters. The parameter γ represents the total number of vulnerabilities. The vulnerability discovery rate has negative skewness when $\alpha > \beta$ and positive skewness when $\alpha < \beta$. When $\alpha = \beta$, the distribution represents a symmetrical model. Since the time value t is defined only within a fixed interval, practitioners need to know the start and end points of the vulnerability discovery process which may be unrealistic in many cases. However, when people could estimate the starting and the ending points of the discovery process, it might provide better performance in some cases because it accommodates the both types of skew.

7.4 Goodness of fit analysis

In this section, the goodness of fit analysis has been evaluated mainly using Chi-square (χ^2) goodness of fit test. The Chi-square statistic is calculated as:

$$\chi_s^2 = \sum_{i=1}^{n} \frac{(o_i - e_i)^2}{e_i} \tag{7.10}$$

where o_i and e_i are the observed and expected values at i^{th} time point respectively. For the fit to be acceptable, the χ^2 statistic value should be less than the corresponding critical χ^2 value for the given alpha level and the degrees of freedom. The P-value represents the probability that a value of the statistic at least as high as the value calculated by Equation 7.10 could have occurred by chance. In the dissertation, level of α chosen is 0.05. Hence, if the P-value of the χ^2 test is below 0.05, then the fit will be considered to be unsatisfactory. In general, for most Chi-square goodness of fit test, including ours, the research hypothesis is the null hypothesis. Hence, a P-value closer to 1 indicates a better fit for the test. The P-value is calculated by using the number of degrees of freedom for the given dataset. The rule of thumb for the χ^2 goodness of fit test is that there should be at least 5 data points, so the analysis is initiated when there are 5 or more data points. Goonatilake et al. (2007) provides the material how to apply Chi-square test in detail.

Table 7.2 shows the parameter values for each S-shaped VDM and the results of the Chi-square goodness of fit test along with the R^2 value for the eight datasets, for the four operating systems and the four Web related software systems. In the table, fits for some of Internet Explorer and Firefox are not accepted since P-values are less than 0.05. However, they can be misleading because visually the plots in Figure 7.5 fit the datasets well. Also all the R^2 values are close to 1, which means very good fittings. Values in the tables are rounded to fourth decimal places.

The results show that the five S-shaped models fit the given datasets very well with most of the P-values in Table 7.2 being 1 or very close to 1, except for the

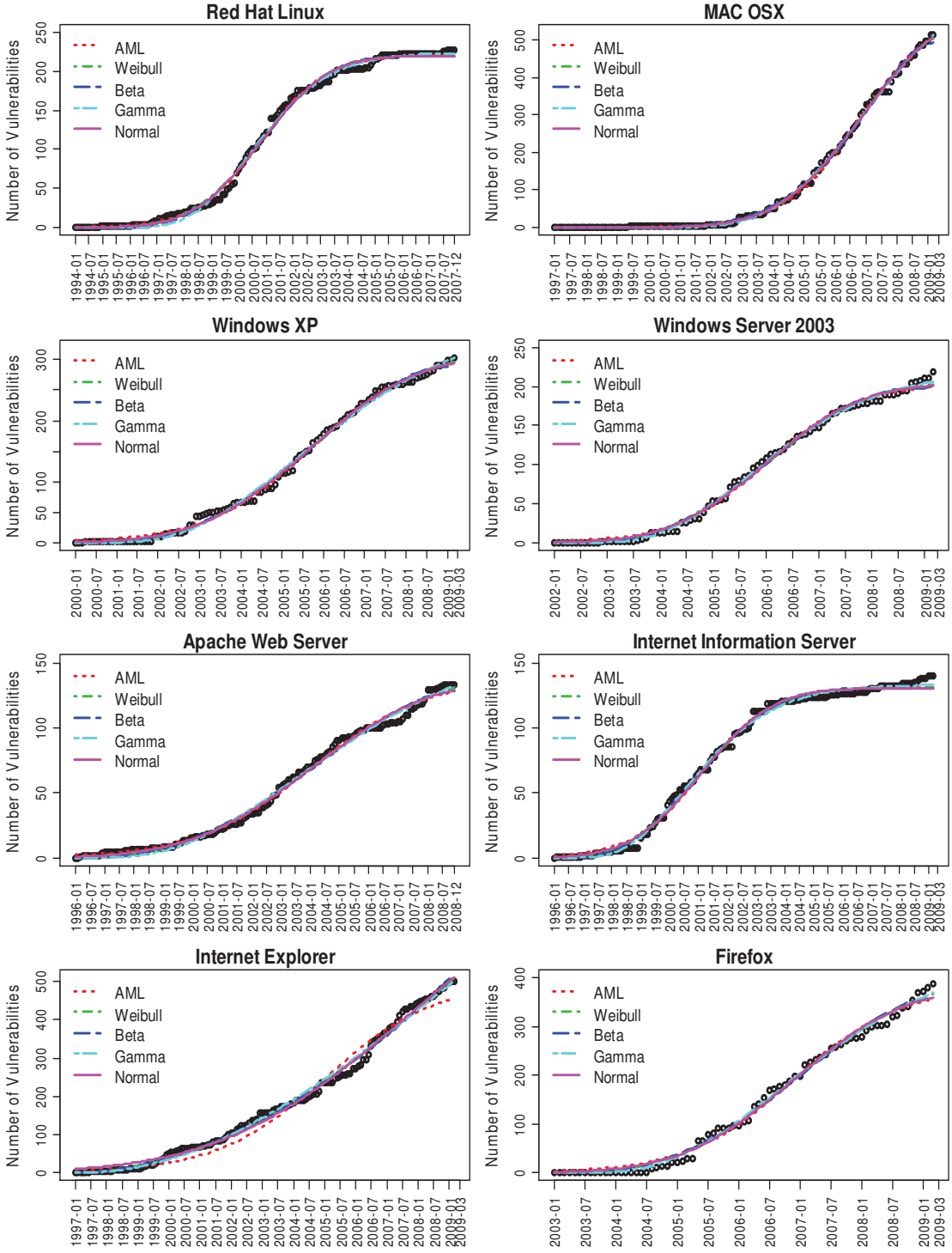


Figure 7.5: Model fitting for the five VDM models. It is observed that all the five S-shaped VDMs fit very well for most of the datasets.

Table 7.2: Model parameters and Chi-square (χ^2) goodness of fit test for model fitting

	Para. 1	Para. 2	Para. 3	χ_s^2	χ_c^2	P-value	d.f.	R^2
	Red Hat Linux							
AML	0.0003	221.1418	1.7409	50.2572	165.3159	1.0000	137	0.9972
Weibull	3.9873	90.9009	218.9963	79.9610	159.8135	0.9999	132	0.9958
Beta	6.1904	6.4613	219.0680	81.9306	153.1979	0.9992	126	0.9960
Gamma	11.8933	7.1397	223.7918	82.2759	148.7793	0.9978	122	0.9969
Normal	82.7842	23.4449	219.7616	81.7450	159.8135	0.9998	132	0.9965
	MAC OSX							
AML	0.0001	572.6486	7.9195	81.3571	118.7516	0.8395	95	0.9980
Weibull	5.8473	124.6684	542.9656	49.6092	114.2679	0.9999	91	0.9986
Beta	6.3442	2.0363	491.5072	61.9089	114.2679	0.9916	91	0.9980
Gamma	16.0513	7.9699	691.7948	46.0737	105.2672	0.9997	83	0.9988
Normal	119.6110	25.6563	592.7040	35.1902	110.8980	1.0000	88	0.9988
	Windows XP							
AML	0.0002	312.2884	0.3316	43.9617	127.6893	1.0000	103	0.9970
Weibull	3.0788	78.2036	312.5689	58.2086	109.7733	0.9925	87	0.9970
Beta	3.1459	2.1794	289.5698	58.9773	112.0220	0.9941	89	0.9969
Gamma	5.3065	15.1303	367.2040	57.5546	95.0815	0.9209	74	0.9960
Normal	69.1542	24.9641	309.0597	64.1183	112.0220	0.9785	89	0.9972
	Windows Server 2003							
AML	0.0005	204.4337	0.6340	52.2160	88.2502	0.9217	68	0.9948
Weibull	3.1638	55.4057	205.8716	42.4827	89.3912	0.9950	69	0.9962
Beta	3.5940	2.9589	198.5265	42.3072	89.3912	0.9953	69	0.9951
Gamma	6.7291	7.8783	220.1135	19.8005	95.0815	1.0000	74	0.9979
Normal	49.0744	16.9582	203.3947	52.5957	91.6702	0.9500	71	0.9957
	Apache Web Server							
AML	0.0003	137.7257	0.5002	24.4280	162.0156	1.0000	134	0.9943
Weibull	2.6975	114.8528	145.4950	23.3024	148.7793	1.0000	122	0.9945
Beta	2.6047	1.8249	127.8919	29.1381	150.9894	1.0000	124	0.9931
Gamma	4.2942	27.8625	172.2811	26.9338	145.4607	1.0000	119	0.9947
Normal	97.6709	38.7897	138.1927	21.9189	155.4047	1.0000	128	0.9946
	Internet Information Server							
AML	0.0005	130.8490	0.6588	70.3371	169.7113	1.0000	141	0.9950
Weibull	2.9219	70.4692	130.5435	47.5040	163.1161	1.0000	135	0.9954
Beta	4.0871	6.3368	130.3883	36.0432	160.9148	1.0000	133	0.9956
Gamma	6.4370	10.1136	133.0643	23.7104	158.7119	1.0000	131	0.9968
Normal	62.3076	23.3523	130.0286	55.2632	166.4153	1.0000	138	0.9949
	Internet Explorer							
AML	0.0001	501.0000	0.2000	728.3713	164.2162	0.0000	136	0.9806
Weibull	1.9656	320.8684	2635.3384	119.5330	160.9148	0.7922	133	0.9950
Beta	1.8291	0.9805	520.7295	109.8756	162.0156	0.9372	134	0.9954
Gamma	2.3513	123.0781	2085.3758	141.1310	158.7119	0.2574	131	0.9934
Normal	129.5461	57.2559	826.6074	348.4672	164.2162	0.0000	136	0.9943
	Firefox							
AML	0.0004	303.4813	0.7132	258.7119	83.6753	0.0000	64	0.9798
Weibull	2.9433	55.5245	401.7704	147.2615	81.3810	0.0000	62	0.9941
Beta	2.9059	1.9255	357.6044	163.5747	81.3810	0.0000	62	0.9923
Gamma	5.2684	10.5932	459.0048	88.4143	77.9305	0.0079	59	0.9956
Normal	47.7852	17.2930	382.2693	189.7606	84.8206	0.0000	65	0.9926

Parameters are in the equations from 3.1 to 7.9: for AML, Parameter 1, 2 and 3 are A, B and C respectively. For Normal, Parameter 1, 2 and 3 are μ, σ and γ respectively. For the rest of them, Parameter 1, 2 and 3 are α, β and γ respectively. Here α level is 0.05 for the Chi-square Goodness of fit test. All the values in the tables are rounded off with four decimal places.

two software systems. The five VDMs produce very similar fitted curves in most cases. For Internet Explorer, the AML and the Normal VDMs produce P-values less than 0.05, so the two symmetrical VMDs are rejected by the goodness of fit test. In Figure 7.1, for Internet Explorer, there is an observable skew to the left since the tail is on the left side. This is probably why Weibull and Beta VDMs, which can model left skewness, fits are relatively better than other VDMs for this dataset. For Firefox, all the five VDMs result in very small P-values with no specific observable cause. It is common that the Chi-square goodness of fit test is inaccurate for small values of the expected numbers because the distribution of the test statistics does not fit the Chi-square distribution very well. However, the R^2 values and visual observations suggest that the fits are reasonably good. Thus, a rejection of the fit based on P-value alone can be questionable.

In Figure 7.5, some of the datasets do not display the three distinct phases that are expected for an S-shaped vulnerability discovery process (Figure 7.4 (a)). A possible explanation for this is provided by superposition effect when the data involves successive versions with changes and additions to the code, as discussed by Kim et al. (2007). This is especially true for several of the software systems examined here. The vendors keep releasing the successive versions with additional functionality time to time using additional code, and it causes the linear phase to be extended.

With the goodness of fit test alone, we do not observe a significant relationship between the underlying assumptions of the VDMs and skewness in datasets. We next compare the models using their prediction capabilities. We examine how well the five S-shaped VDMs can predict the number of vulnerabilities in advance since a superior result of goodness of fit test does not necessarily imply a good prediction capability. The prediction capability is more important than model fitting since the main use of a model is to predict the future trends rather than reviewing the past

behavior. In the following section, each model's prediction capability is evaluated to assess how well the underlying assumptions for the VDMs yield projections for future.

7.5 Prediction capabilities of S-shaped models

The core task for a vulnerability discovery model is to predict the vulnerability detection rate that the software users would encounter during a specific future period. This is similar to the main problem for software reliability estimation after a specific period of testing. A vulnerability discovery model with a better prediction capability should be able to estimate the number of future vulnerabilities more accurately using only currently available data. Such a prediction is needed to estimate the resources needed for maintenance and for the risk estimation.

When the available number of data points is small, there is not enough information to do a meaningful projection. In this study, the analysis of the prediction capability for the five S-shaped VDMs is initiated after about two third of the time period has been elapsed from the beginning of the vulnerability discovery process. For each month, the available data upto that month is fitted to the models using regression analysis to estimate model parameters. Then, using the estimated parameters, the models are used to predict the number of vulnerabilities at the end of the time period, which would eventually represent the complete vulnerability dataset. The estimated final values for each time point produced by the five models are compared with the actual number of vulnerabilities to calculate the prediction error values.

The two predictability measures will be used, Average Error (AE) and Average Bias (AB), as shown in Chapter 5.6. AE is a measure of how well a model predicts throughout the test phase, and AB indicates the general bias of the model which assesses the tendency of the model to overestimate or underestimate.

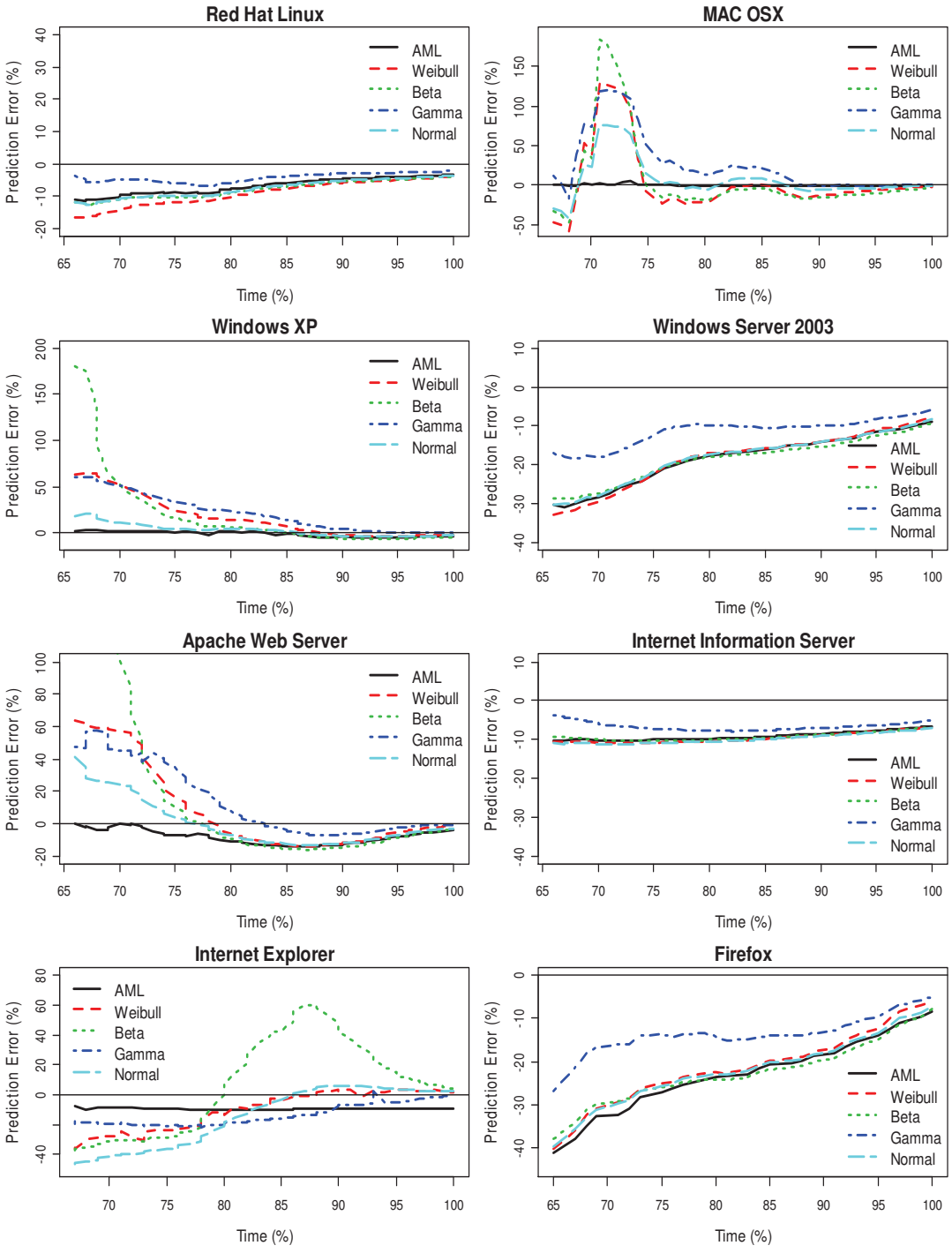


Figure 7.6: Prediction errors for the S-shaped models. Starting points of x-axes are at 66% of time point toward the target month, which means after two third of a time period has been elapsed from the starting point of a vulnerability discovery process. It is clearly shown that Gamma VDM performs the best with all the skewed right datasets of RHL, Server2003, IIS and Firefox.

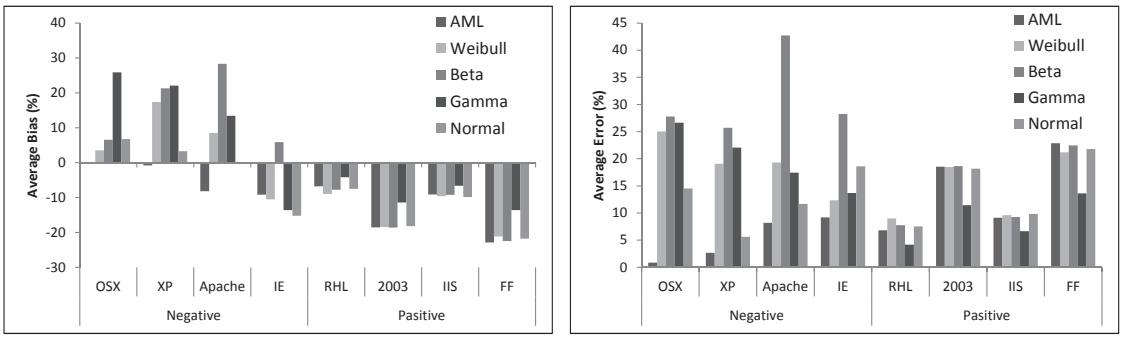
Table 7.3: Average bias and average error (Unit: %)

	AML		Weibull		Beta	
	AB	AE	AB	AE	AB	AE
RHL	-6.8062	6.8062	-9.0041	9.0041	-7.7660	7.7660
OSX	-0.1760	0.8560	3.5248	25.0304	6.5530	27.7730
Win XP	-0.8144	2.6533	17.3505	19.0813	21.2705	25.6956
Server 2003	-18.5416	18.5416	-18.4665	18.4665	-18.6358	18.6358
Apache	-8.2019	8.2019	8.4881	19.3185	28.2841	42.7233
IIS	-9.1149	9.1149	-9.6016	9.6016	-9.2471	9.2471
IE	-9.1961	9.1961	-10.5673	12.3535	5.8406	28.2559
Firefox	-22.8593	22.8593	-21.1867	21.1867	-22.4604	22.4604

	Gamma		Normal		*Empty cell*
	AB	AE	AB	AE	
RHL	-4.1747	4.1747	-7.5176	7.5176	
OSX	25.8446	26.6234	6.7460	14.5324	
Win XP	22.0362	22.0362	3.2513	5.6087	
Server 2003	-11.4468	11.4468	-18.1736	18.1736	
Apache	13.3948	17.4270	-0.0146	11.6706	
IIS	-6.6553	6.6553	-9.8402	9.8402	
IE	-13.5888	13.6924	-15.2435	18.6043	
Firefox	-13.6256	13.6256	-21.7896	21.7896	

The normalized error values $(\Omega_t - \Omega)/(\Omega)$ for each dataset are plotted in Figure 7.6, and the values for AE and AB are given in Table 7.3, where Ω is the actual number of vulnerabilities at target point whereas Ω_t is the estimated number of vulnerabilities at time t. Figure 7.6 clearly shows that, in spite of the rather similar fitted values in Figure 7.5, the models produce significantly different predictions compared with each other. It is observed that one of the two VDMs, AML or Gamma, always results in the best performance for all the datasets.

As one would expect from the skewness properties of the models shown in Table 7.1, the Gamma VDM demonstrates superior prediction capabilities for all the four positively skewed datasets – RHL, Server 2003, IIS and Firefox. A noticeable observation is that AML performs better than other VDMs with the all left skewed datasets. Surprisingly Normal VDM does not behave similar to the AML model even though Normal VDM yields the second best performance for the negatively skewed



(a) Average bias **(b)** Average error

Figure 7.7: It is shown that AML performs the best for the skewed negative dataset whereas Gamma VDM performs the best for the skewed positive datasets.

datasets. One would have expected that the two models would show a similar behavior because of their symmetry. The plots for the Normal VDM are rather similar to the asymmetrical models most of the time. This might suggest that the vulnerabilities tend to be detected a little earlier or later than what the Normal VDM predicts since the logistic distribution has heavier tails than the Normal distribution in both sides.

One would expect that when datasets were skewed left or skewed right, Weibull and Beta VDMs would perform better than AML and Normal VDMs which are symmetrical. However, the results using Weibull and Beta VDMs, which can model both left or right skewed probability density functions, did not support a significant relationship between their PDF behavior and the skewness in the vulnerability datasets although both still predict the future behavior reasonably well. The Beta VDM sometimes generates remarkably different predictions in comparison to the other models. This might be because the Beta distribution is defined only for a finite time period.

Figure 7.7 (a) and (b) summarize the observations. As seen in Figure 7.7 (a), Beta VDM has an average bias smaller than other models for IE. However, that does not mean that the model is better than others. In Figure 7.6, for IE, the prediction

errors for the Beta VDM swing across the zero percent error line widely resulting in positive and negative errors cancelling each other. Figure 7.7 (b) confirms the big error swings. For the right skewed datasets, Gamma VDM yields the best results for the both measurements. For all the right skewed datasets, the five VDMs always underestimate the number of vulnerabilities. This suggests that the datasets are very good candidates for the recalibration approach which relies on the consistency of the bias for adjusting the future predictions (Brocklehurst et al., 1990).

To evaluate the statistical significance of the differences among the AE values, ANOVA (analysis of variance) test is conducted for the negative and positive skewed datasets separately. The ANOVA test only can tell whether the performances among the models are the same or not. It cannot identify which one is better. Hence, Fisher's Least Significant Difference (LSD) (Ott and Longnecker, 2000) test is conducted after identifying the inequality performance from the ANOVA test (LSD equation is shown in Chapter 9.3). For a fair comparison, AE values are normalized by setting the biggest value as 1 in each software group.

Table 7.4 and 7.5 show ANOVA tables for the negatively skewed and the positively skewed datasets respectively for AE. Here, the alpha level has been chosen to be 0.05. To be statistically significant, the F values in the ANOVA tables, need to be greater than the corresponding F critical values with small enough P-value (smaller than 0.05). In the two tables, F values and P-values show that not all the models perform equally.

Table 7.6 and 7.7 show the absolute values of differences between mean values of AE and the significance of pairwise comparisons with the shaded cells representing statistically significant differences. For the shaded cells, differences between two compared models' mean values are greater than calculated LSD value. The calculated LSD values for each table is $\text{LSD}_{negative} = 0.2896$ and $\text{LSD}_{positive} = 0.1189$ respectively. Table 7.6 confirms that AML performs better than Weibull, Beta, and

Table 7.4: ANOVA table for the skewed negative datasets in AE

Negative	SS	df	MS	F	P-value	F_{crit}
Between Groups	1.555252	4	0.388813	10.52039	0.00029	3.055568
Within Groups	0.55437	15	0.036958			
Total	2.109622	19				

Table 7.5: ANOVA table for the skewed positive datasets in AE

Positive	SS	df	MS	F	P-value	F_{crit}
Between Groups	0.414582	4	0.103646	16.62514	2.18493E-05	3.055568
Within Groups	0.093514	15	0.006234			
Total	0.508096	19				

Table 7.6: LSD. Difference between mean values from negatively skewed datasets in AE

Negative		AML	Weibull	Beta	Gamma	Normal
	Mean	0.1629	0.6333	1.0000	0.6772	0.4183
AML	0.1629	0	0.4704	0.8371	0.5143	0.2554
Weibull	0.6333		0	0.3667	0.0439	0.2150
Beta	1.0000			0	0.3228	0.5817
Gamma	0.6772				0	0.2589
Normal	0.4183	$LSD_{Negative}$ =0.2896				0

Table 7.7: LSD. Difference between mean values from positively skewed datasets in AE

Positive		AML	Weibull	Beta	Gamma	Normal
	Mean	0.9193	0.9734	0.9462	0.5876	0.9408
AML	0.9193	0	0.0541	0.0269	0.3317	0.0215
Weibull	0.9734		0	0.0272	0.3858	0.0325
Beta	0.9462			0	0.3586	0.0054
Gamma	0.5876				0	0.3533
Normal	0.9408	$LSD_{Positive}$ =0.1189				0

Gamma VDMs in negative datasets, but cannot tell AML is better than Normal VDM. The table also tells that Beta VDM performs worse than others. Table 7.7 confirms that Gamma VDM performs better than others in positive datasets.

7.6 Categorization by vulnerability type

In this section, the five S-shaped VDMs are applied to vulnerability discovery processes on OSX and IIS grouped by vulnerability types. The reason why OSX and IIS are selected here is that they have the most skewed datasets in each skewness (See Table 7.2).

Figure 7.8 shows the number of categorized vulnerabilities for OSX and IIS in each month. Applying VDMs on categorized datasets might be useful if patch developers like to inspect vulnerability behaviors for a certain types of fault only. Here, the classification method is the same with Chapter 4.3 having eight types.

Table 7.8 and 7.9 compare vulnerability distributions in OSX and IIS by type. The categories with the highest numbers are input validation errors, followed by design errors. There is a slight difference in category ordering between OSX and IIS from the third place: OSX has more Exception errors than access validation errors whereas IIS has more access validation errors. It could be interpreted that IIS has been more vulnerable to Access validation errors because the Web servers need to be accessed by Web browsers, so that the servers could more frequently undergone the specific error. OSX, on the other hand, has higher percentages on Other type due to the OSes' own complexities.

To determine whether there is an observable pattern at the level of individual classes, the top four most frequent categories are fitted and predicted as shown in Figure 7.9 and 7.10 respectively.

In Figure 7.9, the upper four are from OSX while the bottom four are from IIS. The most clear observation is that OSX has been in the linear phase whereas IIS

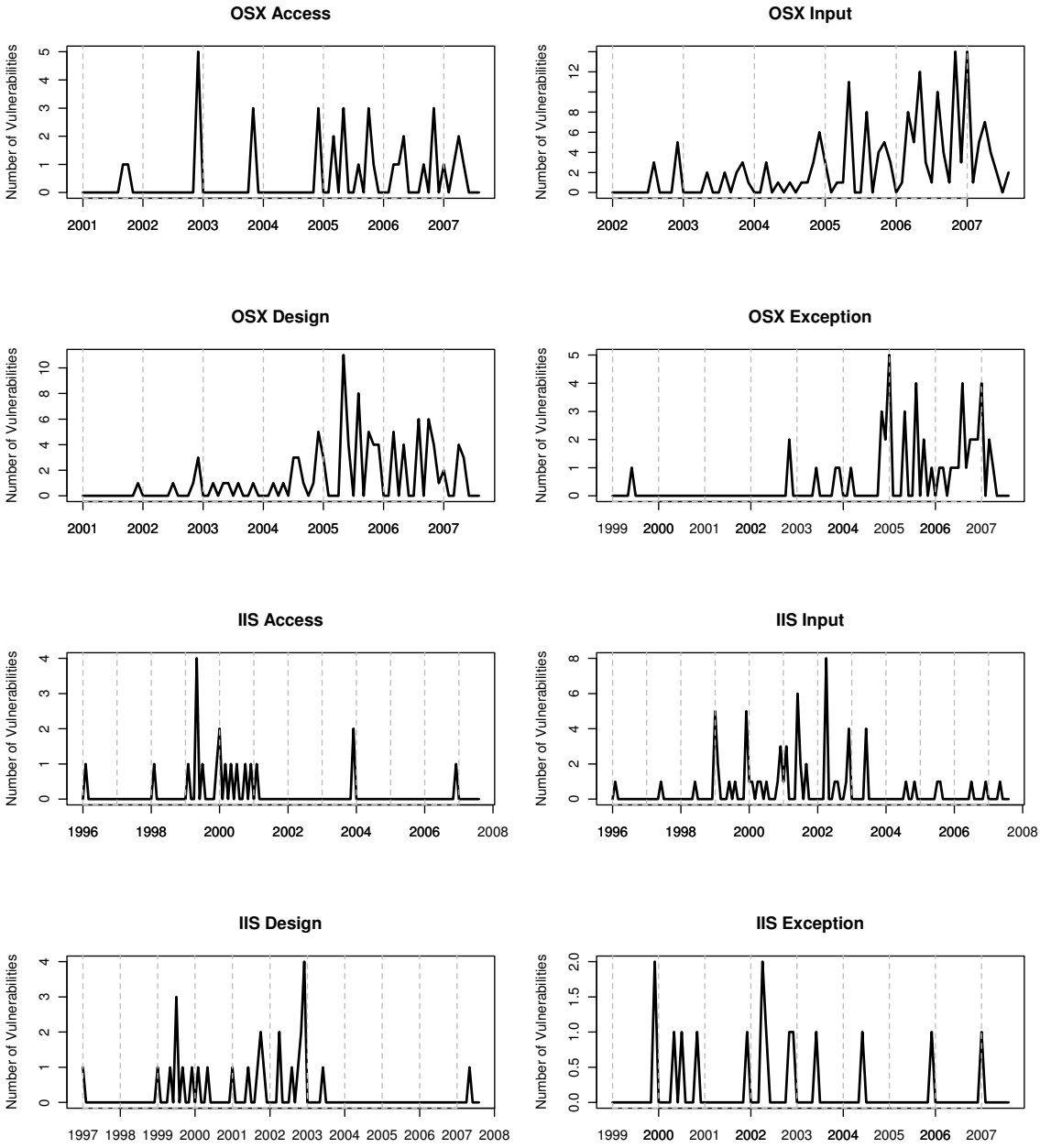


Figure 7.8: Run chart for the number of vulnerabilities grouped by vulnerability types. The upper four plots from OSX, and the bottom four plots from IIS.

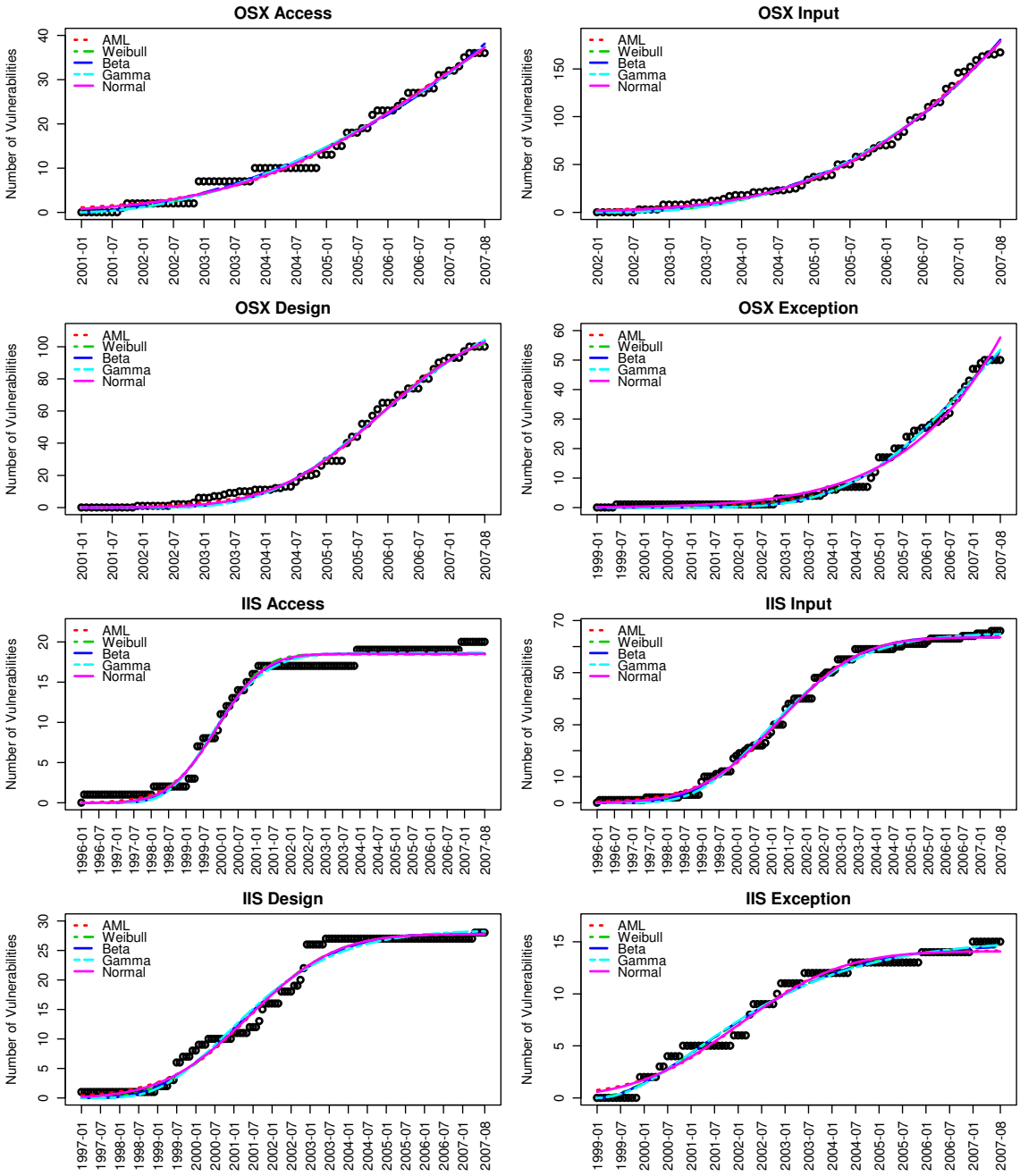


Figure 7.9: Model fitting for the four vulnerability types from OSX and IIS. The upper four plots from OSX, and the bottom four plots from IIS. It is observed that all the five VDMs fit very similarly and very well.

Table 7.8: Skewness and total number of vulnerabilities by type I – Access, Input, Design, & Exceptionion.

Type:	Access	Input	Design	Exception
MAC OSX				
Skewness	-0.6488	-1.0382	-0.8143	-1.8566
Num Of Vuln.	36	167	100	50
Percentage	9.05%	41.96%	25.13%	12.56%
IIS				
Skewness	1.2430	0.4809	0.4080	0.8023
Num Of Vuln.	20	66	28	15
Percentage	14.08%	46.48%	19.72%	10.56%

Table 7.9: Skewness and total number of vulnerabilities by type II – Environment, Configuration, Race & Other.

Type:	Env	Config	Race	Other
MAC OSX				
Skewness	-1.3221	-0.1879	NA	-2.3013
Num Of Vuln.	4	15	1	25
Percentage	1.01%	3.77%	0.25%	6.28%
IIS				
Skewness	1.0325	0.0477	NA	-1.3896
Num Of Vuln.	4	5	1	3
Percentage	2.82%	3.52%	0.70%	2.11%

shows a clear saturation phase, just like what the aggregated vulnerabilities shows in Chapter 7.4. Moreover, the five model fittings also represent very similar pattern in each case. As a result, we are not able to meaningfully compare the models due to the similar behaviors for the model fittings. The corresponding χ^2 values have not been assessed due to the clear visual observations and an expectation of similar results from Chapter 7.4.

Meanwhile, Figure 7.10 shows the prediction errors for the four vulnerability types in each software system (upper four are OSX, and bottom four are IIS). Table 7.10 reveals that there is differences among the normalized mean values in OSX categorizations, but Table 7.11 tells that there is no statistically significant

Table 7.10: ANOVA table for categorized OSX in AE

Positive	SS	df	MS	F	P-value	F_{crit}
Between Groups	0.537452208	4	0.134363052	13.04515488	8.92243E-05	3.055568276
Within Groups	0.15449765	15	0.010299843			
Total	0.691949858	19				

Table 7.11: ANOVA table for categorized IIS in AE

Positive	SS	df	MS	F	P-value	F_{crit}
Between Groups	0.019992705	4	0.004998176	0.074269525	0.988950185	3.055568276
Within Groups	1.009467108	15	0.067297807			
Total	1.029459813	19				

Table 7.12: LSD; Difference between mean values from categorized OSX in AE

OSX		AML	Weibull	Beta	Gamma	Normal
	Mean	0.6440	0.5394	0.6753	0.5746	1
AML	0.6440	0	0.1046	0.0313	0.0695	0.3560
Weibull	0.5394		0	0.1359	0.0351	0.4606
Beta	0.6753			0	0.1007	0.3247
Gamma	0.5746				0	0.4254
Normal	1	LSD_{OSX} =0.1529				0

differences in IIS. Consequently, LSD is only calculated for the OSX as shown in Table 7.12.

While, for the prediction capabilities, Chapter 7.5 shows that negative skewed data was represented better with AML model, here, Table 7.12 only shows that Normal VDM is worse than other models. It has been observed that, for the categorized vulnerabilities, we have not seen any correlations between the models and skewed datasets examined. This might be because of the statistically not significant number of data points.

7.7 Summary

This study focuses on the models based on the S-shaped distributions, which have been successfully used for the prediction of growth rates in many other fields.

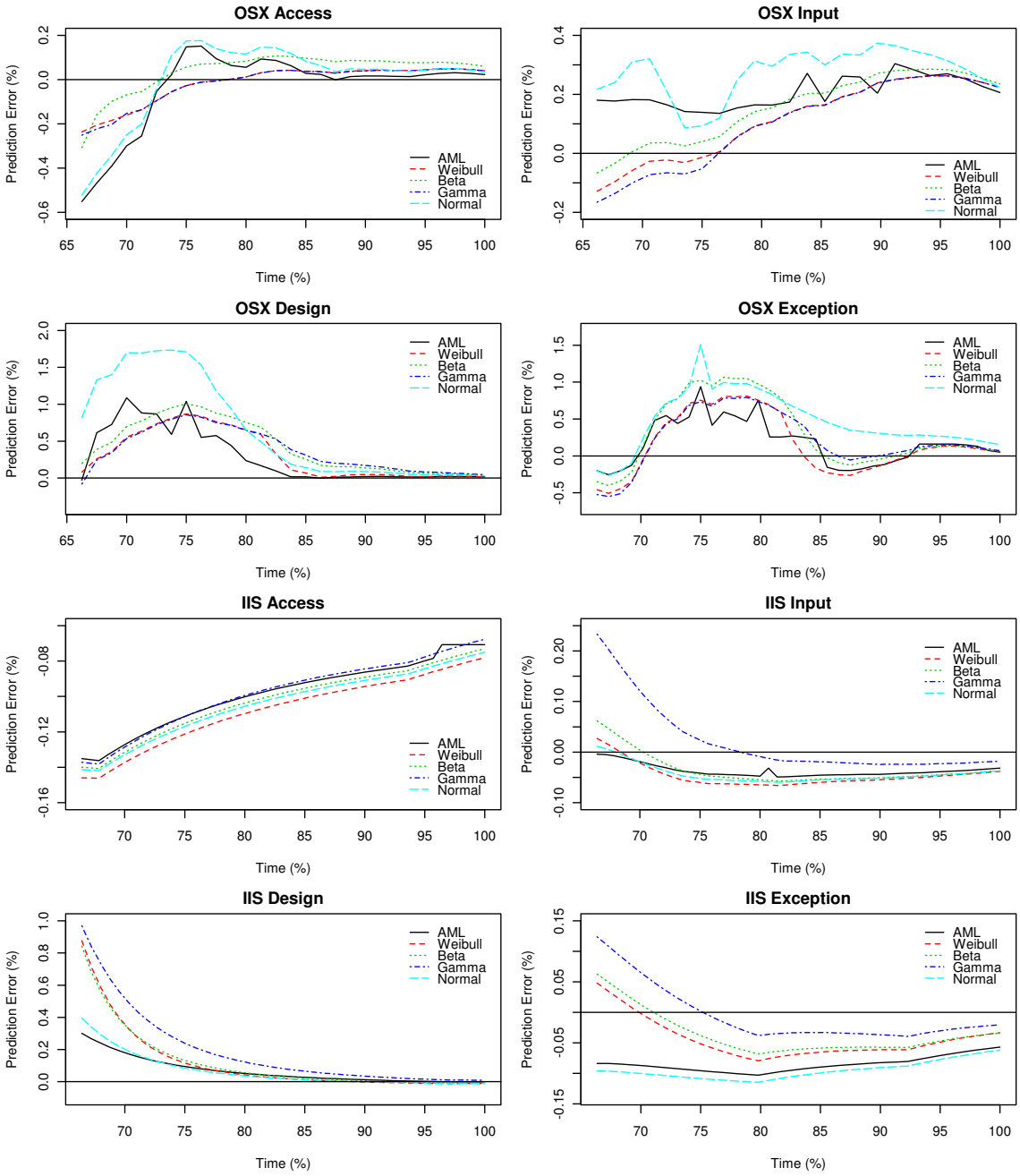


Figure 7.10: Prediction errors by vulnerability types. The upper four plots from OSX, and the bottom four plots from IIS. Starting points of x axes are at 66% of time point toward the target month, which means after two third of a time period has been elapsed from the starting point of a vulnerability discovery process.

Here, the relationship of the performance of an S-shaped VDM with the specific skewness attributes of its underlying probability distribution and in a specific vulnerability discovery dataset has been examined. The prediction capabilities of five S-shaped VDMs (AML, Weibull, Beta, Gamma and Normal) have been examined with the Chi-square goodness of fit tests for the major operating systems (Windows XP, Windows Server 2003, Red Hat Linux, and MAC OSX) and the major Web servers and browsers (Apache Web server, IIS, Internet Explorer, and Firefox). The results of Chi-square goodness of fit test as well as R-squared metric indicate that the five S-shaped models generally fit well, for the most of the datasets examined.

However, even though the five VDMs fit the past datasets well, each model predicts future trends differently at each point in time. The AML model yields the best predictions for the negative skewed datasets followed by Normal VDM whereas Gamma VDM gives the best forecast with positive skewed datasets. The Beta VDM, which is defined only for a finite time period, generally results in the worst forecast even though it can represent both types of skewness. The performance of Weibull and Normal VDMs often tends to be in the middle among the models.

There are some surprising observations. One would expect that the AML model would have weak predication capabilities when the vulnerability discovery datasets are asymmetrical since AML model presumes a symmetrical logistic distribution. In addition, one would expect that the asymmetrical S-shaped models should not only have better goodness of fit results but also better prediction capabilities when the vulnerability discovery data is skewed. The results show, as expected, that the Gamma VDM which assumes a right skewed distribution always provides better performance with the positive skewness datasets than other VDMs. However, the performance of the Weibull and Beta VDMs does not show significant correlations with the skewness in the datasets. They generally perform worse than the two symmetrical models with left skewed datasets, and show a similar performance with

right skewed datasets. The results suggest that Gamma VDM should be preferred to model the vulnerability discovery process with right skewed datasets, and for other datasets, AML is generally a better choice.

Another significant observation is that an excellent goodness of fit does not necessarily mean a superior prediction capability, and thus a quantitative model should not be chosen based on fit alone.

Chapter 8

EXTENDED LINEAR VULNERABILITY DISCOVERY PROCESS

8.1 Motivation

Recently, robust linear behaviors in software vulnerability discovery process have been noticeably observed among the many popular systems having multi-versions released. Schryen (2009) empirically examined vulnerability detection growth processes in seventeen software systems. He found that 14 out of the 17 systems show a significant linear or, at least, piecewise linear correlation between time and the number of cumulative published vulnerabilities, but without a deep investigation why the linear processes are prevalent. While showing the results, the author disproves the S-shape logistic vulnerability discovery pattern proposed by Alhazmi et al. (2005). However, in his article, Schryen fails to mention that, in their later works (Alhazmi and Malaiya, 2006a,b; Alhazmi et al., 2007; Alhazmi and Malaiya, 2008), Alhazmi and Malaiya tried to apply the linear discovery model on their datasets although the results from the linear model are frequently not statistically significant.

In Figure 8.1, the solid S-shaped line shows the shape of the vulnerability discovery process in AML model with the three distinctive phases. In the long run, for a software system, the vulnerability discovery process should look like the S-shape pattern when all the source codes with market effort put on it are reflected as argued in Chapter 7. During the release period, the vulnerability discovery rate

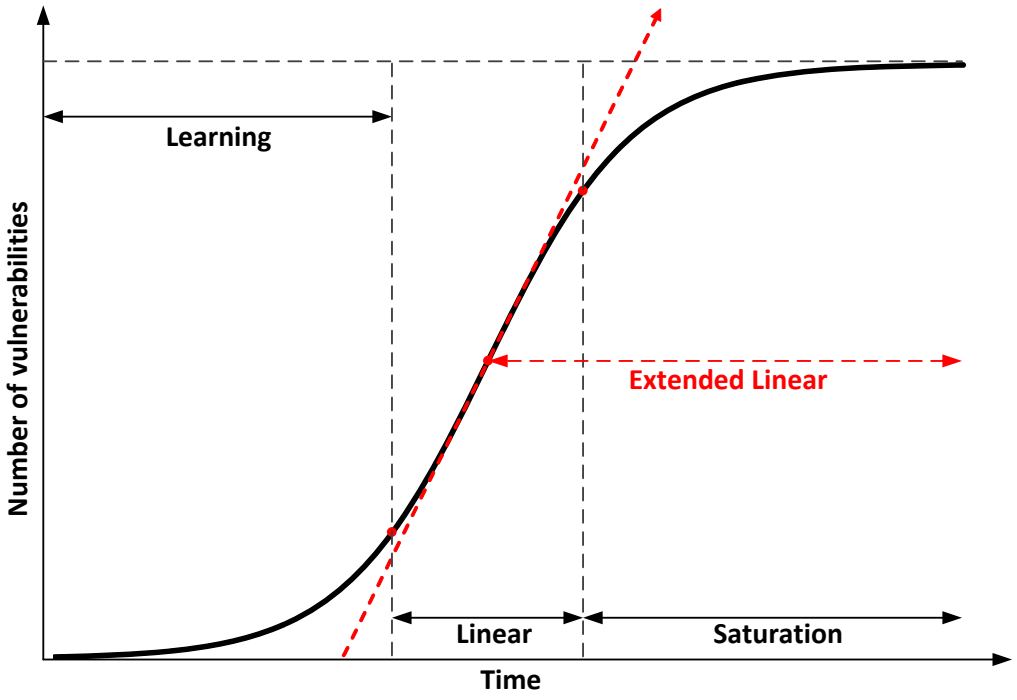


Figure 8.1: S-shaped discovery process and extended linear phase

gradually increases. At this phase, called learning phase, the software is gaining market share gradually and installed bases is small. In the linear phase, the discovery rate reaches the maximum due to the popularity, and finally, in the saturation phase, vulnerability discovery rate slows down.

However, under certain circumstances, the S-shape could be distorted, occasionally, seriously. The length of the second phase could be extended as long as new code is injected with certain levels of popularity lasted among users, so that the final phase tends to appear significantly later. Sometimes, after a clear saturation phase, new vulnerabilities are found. When this happens repeatedly, a discovery process forms a stairway-like pattern. Yet another, the first phase could not be seen at all. It is possible that combinations of above cases are coming out altogether. Among the mutant S-shapes above, in this chapter, mainly, the reason behind the extended linear phase is examined which currently appears notably. Other mutations also can be surmised based on the presentations here.

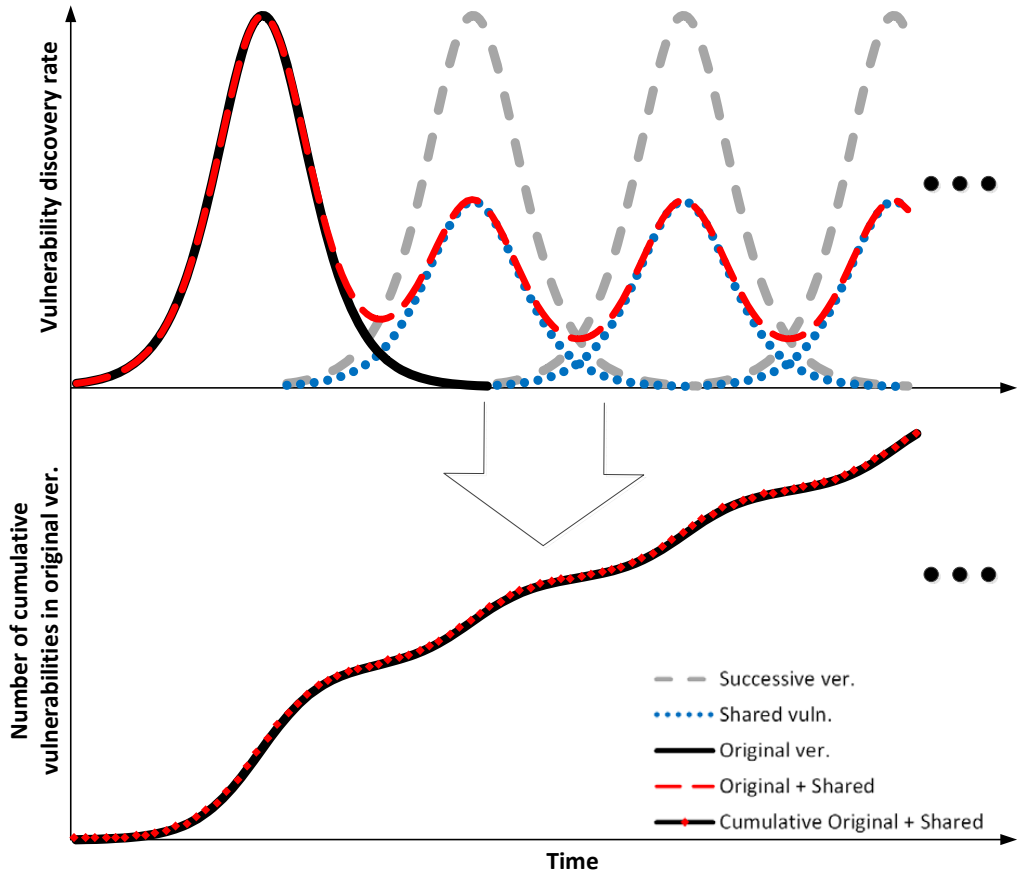


Figure 8.2: The extended linear phase is caused by shared vulnerabilities in successive versions

8.2 Possible causes for the extended linear phase

The red dashed line, in Figure 8.1, highlights an extended linear phase. The first possible reason for this could be code sharing throughout the successive versions. New versions of software systems usually based on the previous version. When the product is getting popular, the number of users is also getting increasing. As a result, vulnerabilities originated from the earlier version starts to be found in the later version.

Moreover, new chunk of codes added into a new version introduces new vulnerabilities. When those software upgrades or patches go on and on, the extended linear phase could be resulted. Figure 8.2 shows this behavior. The original idea of sharing vulnerability is already introduced by Kim et al. (2007). In the figure, the

vulnerability discovery rate for the original software system has been almost hit the saturation phase, marked by the solid black line (the first bell shape hump), but due to the shared vulnerabilities in successive versions (the grey dashed lines), the vulnerability discovery rate for the original product continually rises. The slope in the number of cumulative vulnerabilities is mainly influenced by how many codes are shared between the successive versions. Hence, as long as new versions, sharing codes with the previous version, are released with an enough market share, the extended linear phase will be observed.

The second reason could be, for a software system, the constant number of users with a vulnerability pool having a sheer amount of vulnerabilities which continually discovered with a constant rate due to a balanced effort, not increasing nor decreasing, put on the system, such as number of users. In this case, it will take some time proportional to the size of the vulnerability pool to be exhausted which causes a longer linear phase with a bigger pool. The concept is described in Figure 8.3. If we extend this idea, the linear phase could be caused by the assumption[1] that the majority of vulnerabilities might be detected internally by developers during test phases, or externally by constant number of security professionals interested in a specific software system.

8.3 Observations

The software systems examined for the linear trend here are Windows, OSX, Apache Web server, IIS, Internet Explorer, and Firefox. Further, each system is dissected into the four latest specific versions. The datasets are minded at NVD on January 2011. Table 8.1 shows the release dates for the software systems. It appears that OSX is slightly younger than Windows with only these datasets. Likewise, IIS and Internet Explorer (IE) have been released in their major versions more recently

[1] `http://blog.bit9.com/bid/25781/Reported-Vulnerabilities-Quality-versus-Quantity`

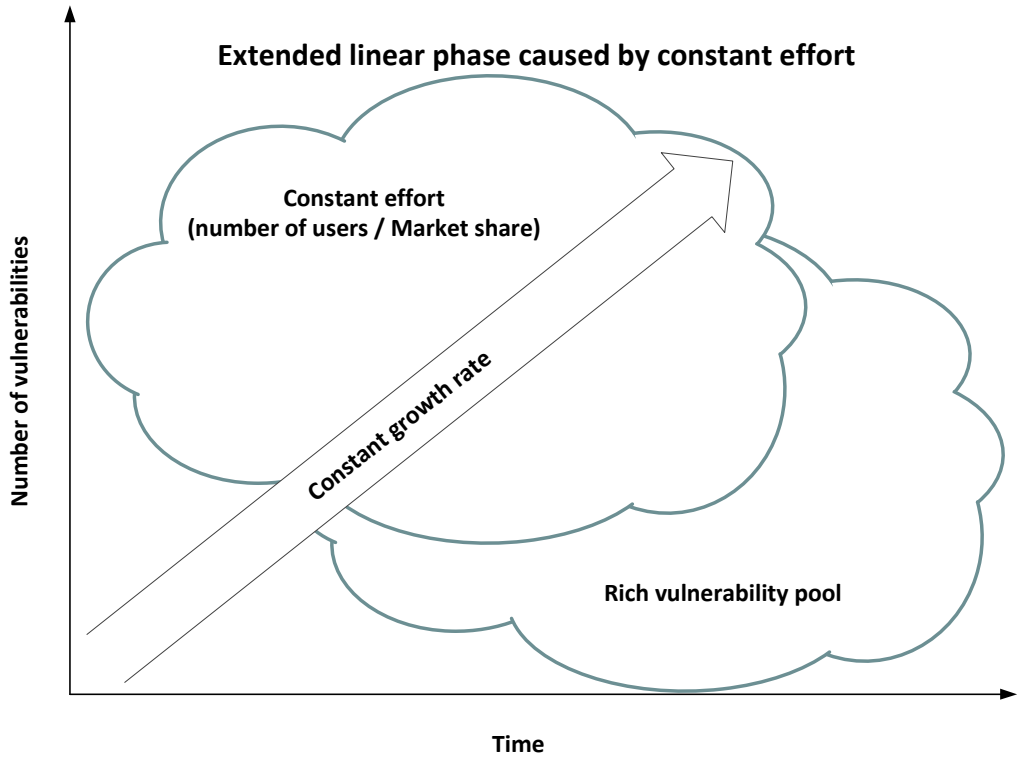


Figure 8.3: The extended linear phase is caused by constant effort put on a system with a rich vulnerability pool

than their counterparts for the past several years. However, Apache still supports and distributes minor versions for 1.3 and 2.0 due to the existing many Web servers having those versions, and releases minor versions frequently for the all three major versions of 1.3, 2.0 and 2.2.

Table 8.2 shows the number of vulnerabilities shared among the successive versions in each software product. In the chapter, the oldest and the latest versions are denoted only based on the table. For example, the oldest Windows OS is 2K and the newest one is Seven. In the table, it could be conjecturable that the code sharing is higher with adjacent versions than others based on the shared number of vulnerabilities. The vulnerability sharing ratios in both OSes from the oldest (Windows 2K & OSX 10.3) to the latest versions (Windows Seven & OSX 10.6) are 5.76% and 5.14 % respectively.

Table 8.1: Software release date

Windows

Version	Release Date
2K	2000-02-17
XP	2001-08-24
Vista	2006-11-08
Seven	2009-07-22

Apache

Version	Release Date
0.2	1995-03-18
1.3	1998-06-06
2.0	2000-03-13
2.2	2005-12-01

Internet Explorer

Version	Release Date
5	1999-03-18
6	2001-08-27
7	2006-10-18
8	2009-03-19

OSX

Version	Release Date
10.3	2003-10-24
10.4	2005-04-29
10.5	2007-10-26
10.6	2009-08-28

IIS

Version	Release Date
4	1997-12-15
5	2000-02-17
6	2003-04-24
7	2008-02-04

Firefox

Version	Release Date
0.1	2002-09-23
1.0	2004-10-27
2.0	2006-03-22
3.0	2006-12-08

For the Web servers, while Apache still shares 40% between the oldest and the latest one, IIS does not shares any between the two, revealing that IIS has been considerably changed from its earlier version. This might be because Apache Web server 1.3 is still evolving, so that the version should have shared some codes with the latest version developed by the same group of developers.

In case of the Web browsers, about 5% of the vulnerabilities from the oldest version is shown up in the newest one for IE while about half of the vulnerabilities have survived in the latest Firefox version from the oldest version. Even with consideration of the longer time gap between the oldest and the newest version in IE, it can be interpreted that the source code reusability in the open source software system tends to be higher than its counterpart. Meanwhile, 90% of vulnerabilities in Windows Seven is sharing with Vista, 70.85% of vulnerabilities in Vista is from XP, and 63.29% of the vulnerabilities in XP is from the previous version, which uncovers that the Windows OSes are continually built on top of its ancestors closely.

Plots in Figure 8.4 shows the linear model fittings and Table 8.3 shows their R^2 values. In those plots, the entire vulnerabilities are considered in each subplot regardless of sharing among the versions. In all cases, the linear patterns are

Table 8.2: Shared number of vulnerabilities and percentages. Numbers in parentheses represent the number of vulnerabilities not shared. For the percentages, it should be read as A is sharing X% with B, where A and B are the row and column respectively as marked below.

Windows

A \ B	2K (191)	XP (113)	Vista (49)	Seven (8)
2K	493 100%	300 60.85%	99 20.08%	28 5.67%
XP	300 63.29%	474 100%	158 33.33%	58 12.23%
Vista	99 44.39%	158 70.85%	223 100%	72 32.28%
Seven	28 35%	58 72.5%	72 90%	80 100%

OSX

A \ B	10.3.x (100)	10.4.x (171)	10.5.x (86)	10.6.x (48)
10.3.x	214 100%	111 51.86%	20 9.34%	11 5.14%
10.4.x	111 31.71%	350 100%	85 24.28%	11 3.14%
10.5.x	20 8.88%	85 37.77%	225 100%	58 25.77%
10.6.x	11 10%	11 10%	58 52.72%	110 100%

Apache

A \ B	0.x (3)	1.3.x (35)	2.0.x (38)	2.2.x (16)
0.x	10 100%	7 70%	4 40%	4 40%
1.3.x	7 11.86%	59 100%	20 33.89%	11 18.64%
2.0.x	4 5.88%	20 29.41%	68 100%	20 29.41%
2.2.x	4 10.81%	11 29.72%	20 54.05%	37 100%

IIS

A \ B	4.x (40)	5.x (31)	6.x (3)	7.x (3)
4.x	85 100%	45 52.94%	2 2.35%	0 0%
5.x	45 50.56%	89 100%	14 15.73%	5 5.61%
6.x	2 10%	14 70%	20 100%	7 35%
7.x	0 0%	5 50%	7 70%	10 100%

Internet Explorer

A \ B	5.x (76)	6.x (136)	7.x (37)	8.x (26)
5.x	296 100%	220 74.32%	73 24.66%	17 5.74%
6.x	220 51.64%	426 100%	128 30.04%	56 13.14%
7.x	73 42.44%	128 74.41%	172 100%	48 27.9%
8.x	17 19.1%	56 62.92%	48 53.93	89 100%

Firefox

A \ B	0.x (10)	1.x (154)	2.x (52)	3.x (22)
0.x	233 100%	221 94.84%	157 67.38%	118 50.64%
1.x	221 45.19%	489 100%	262 53.57%	182 37.21%
2.x	157 48.01%	262 80.12%	327 100%	184 56.26%
3.x	118 54.88%	182 84.65%	184 85.58%	215 100%

significantly observed and the linear fittings are well performed, especially, if we consider the datasets only after the released dates such as shown in OSX, IIS, or IE. We do not see any saturation phase at the end of the data periods in all the six aggregated versions while some of the earlier versions in OSX and IIS show saturation phases. IIS whose sharing rate is relatively small from the oldest to the youngest version, the saturation phases are clearly observed in the first three versions. On the other hand, its counterpart, Apache Web server, the linear growth patterns are dominant because 1.3.x, 2.0.x, and 2.2.x are currently still updated every now and then.

Figure 8.5 shows the number of unique vulnerabilities in each specific version. Their R^2 values can be found at Table 8.3. First, it is observed that the number of vulnerabilities have been dramatically reduced in each sub-plot compared to its entire vulnerability counterpart sub-plot from Figure 8.4 due to the removing the shared vulnerabilities. The noteworthy thing is that the learning phases start to appear more clearly then Figure 8.4. Also, the third phase tends to come out more often when its market share has been encroached by its successive version which proves that the extended linear phenomena is due to the code sharing with the popular successive versions. Especially, Windows 2K, OSX 10.5.x, IE 5.x, IE 6.x, IE 7.x, and Firefox 1.x reveal the saturation phase with unique vulnerabilities while their counterpart sub-plots for the entire vulnerabilities do not.

8.4 Prediction for the extended linear rate

As mentioned in Section 3.2, Equation 3.3 represents the simple linear discovery model, where S represents the slope or discovery rate and k is y-axis intersection which does not have a clear meaning. Now, predicting the exact discovery rate or slope for the extended linear phase is not an easy task. However, we could achieve fairly easily a probable scope of the rate falling into the ranges from the maximum and minimum slopes estimated by AML model fitting.

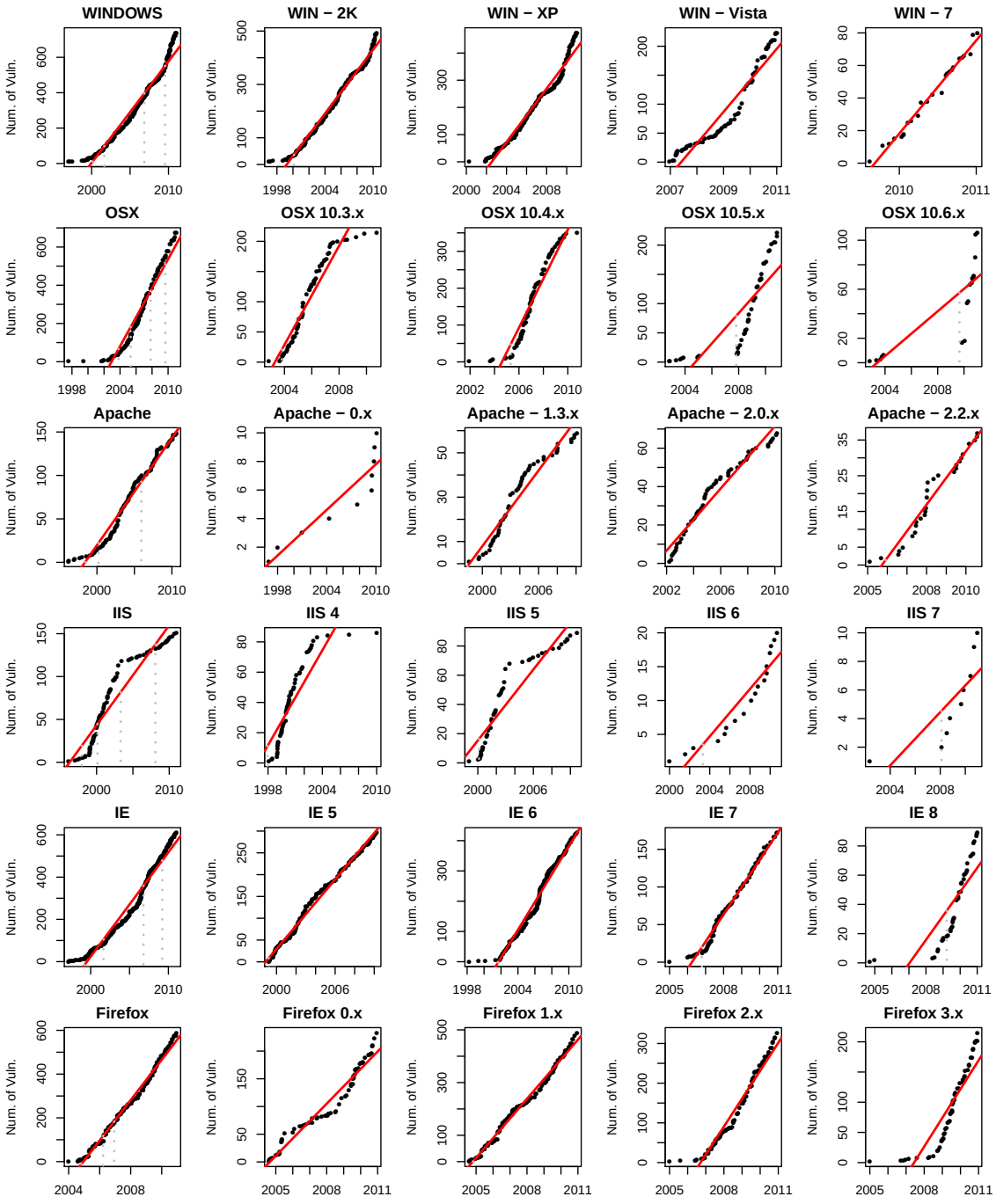


Figure 8.4: Linear vulnerability growth trends in the six software systems. Black dots represent actual data points and the red lines are linear model fittings. Vertical dotted lines are released dates if any. Each row represents the same software group.

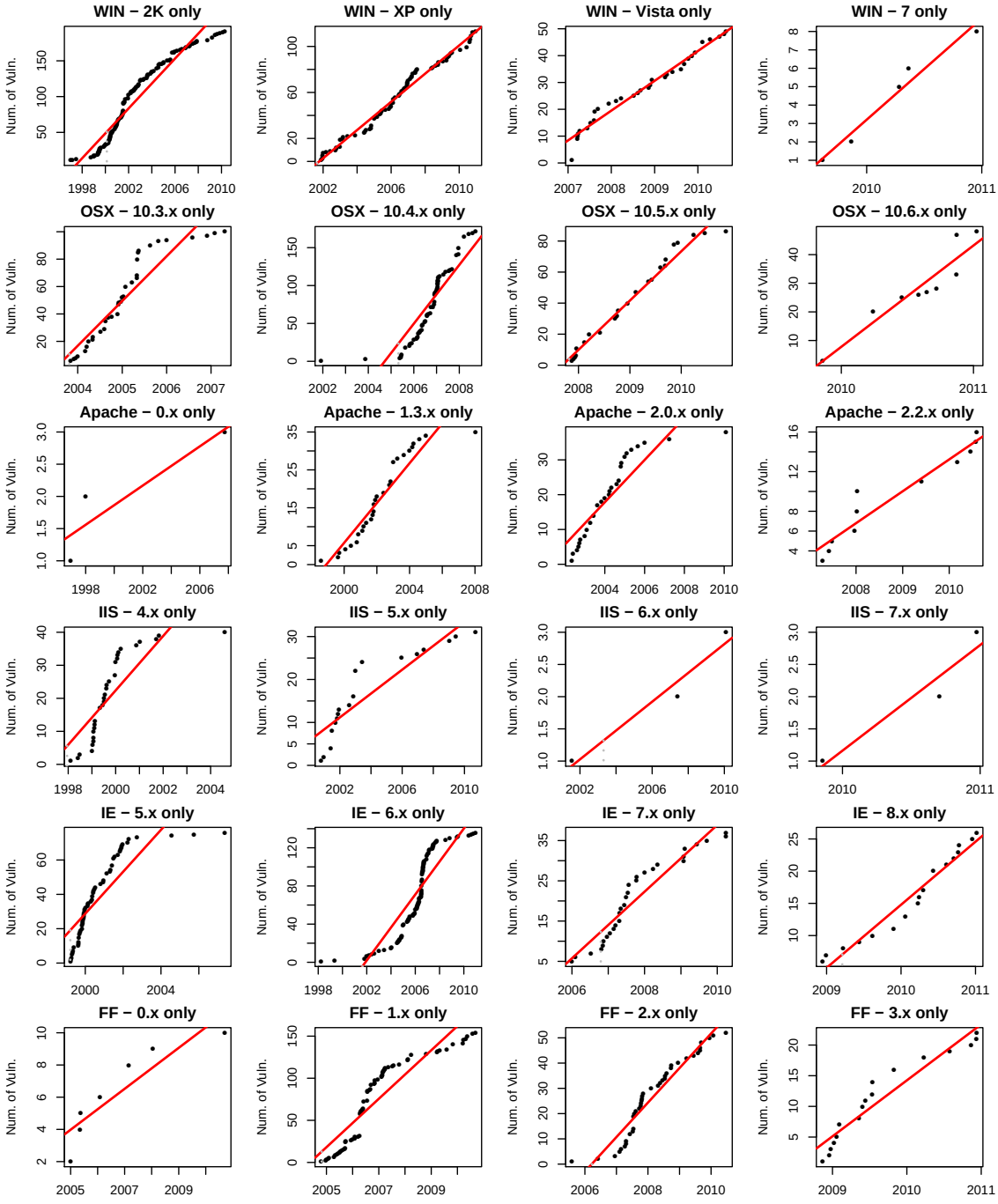


Figure 8.5: Linear vulnerability growth trends in the six software systems by version with unique vulnerabilities in them. Black dots represent actual data points and the red lines are linear model fittings. Vertical dotted lines are released dates if any. Each row represents the same software group.

Table 8.3: R^2 values from Figure 8.4 and 8.5

Windows

Version	Fig. 8.4	Fig. 8.5
All	0.965	NA
2K	0.9849	0.9275
XP	0.9664	0.9827
Vista	0.9139	0.9723
Seven	0.9814	0.9661

Apache

Version	Fig. 8.4	Fig. 8.5
All	0.9699	NA
0.x	0.8480	0.8224
1.3.x	0.9352	0.8861
2.0.x	0.9527	0.7795
2.2.x	0.9441	0.9228

OSX

Version	Fig. 8.4	Fig. 8.5
All	0.9142	NA
10.3.x	0.9022	0.8662
10.4.x	0.9020	0.7720
10.5.x	0.6996	0.9744
10.6.x	0.7325	0.8804

IIS

Version	Fig. 8.4	Fig. 8.5
All	0.9020	NA
4.x	0.7457	0.6666
5.x	0.8685	0.8164
6.x	0.8723	0.9562
7.x	0.6304	0.9185

Internet Explorer

Version	Fig. 8.4	Fig. 8.5
All	0.9664	NA
5.x	0.9937	0.7086
6.x	0.9656	0.7711
7.x	0.9787	0.9032
8.x	0.6238	0.9536

Firefox

Version	Fig. 8.4	Fig. 8.5
All	0.9903	NA
0.x	0.9512	0.8121
1.x	0.9912	0.8327
2.x	0.9541	0.9228
3.x	0.7700	0.9127

Table 8.4: Estimated slopes in Figure 8.4

	Windows	OSX	Apache	IIS	IE	Firefox
Slope	0.1569453	0.2078473	0.03377206	0.03192421	0.1352958	0.2520864

Figure 8.6 demonstrates the maximum ($tan(\beta)$) and the minimum ($tan(\alpha)$) slopes during the linear phase in the AML model. Consequently, the difference (θ) between the two slopes can be achieved. A and B are from the AML parameters. The maximum slope is on the tangent line of the mid-point whereas the minimum slope exists on either of the two transition points. Hence, in some degree, it is possible to estimate the current extended linear vulnerability discovery rate for the multi-version software systems.

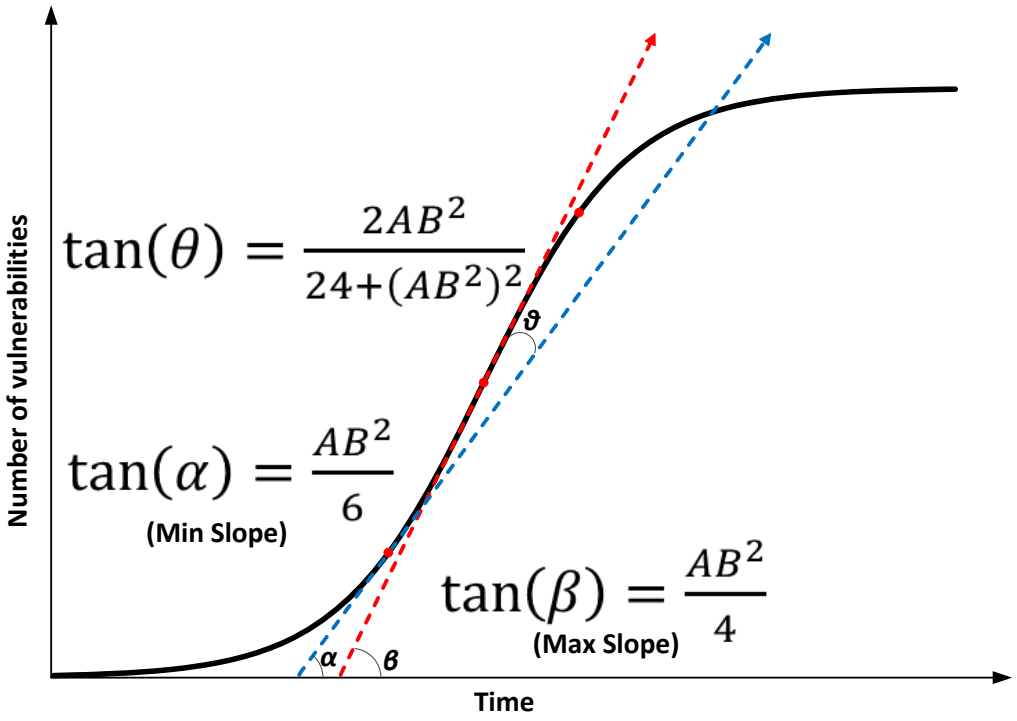


Figure 8.6: Estimated Max/Min slopes by AML

Table 8.4 shows the slopes from the linear fitting for the aggregated versions in Figure 8.4. The first impression from Table 8.4 is that OSX and Firefox have higher slope values against their counterparts. Apache Web server is a bit higher than IIS, but it is not significantly different from IIS. Hence, apparently, vulnerability discovery rate in Microsoft products seems lower than its rival products when only considering the linear model fitting.

Table 8.5 and 8.6 are the estimated information by the AML model fitting on the six aggregated versions in Figure 8.4. Based on the R^2 values, AML fittings reasonably well perform. Transition point 1, 2 and mid-point are produced by the fitting as described in Section 3.2, and the minimum and maximum slopes are measured as mentioned above.

Values for Windows, Apache, IE and Firefox from Table 8.4 are in the boundaries of the minimum and the maximum slopes from Table 8.5. OSX and IIS slope values are lower than their low boundaries because TP1s by AML model fitting are

Table 8.5: Information estimated by AML - Parameters

	A	B	C	R^2	Min Slope	Max Slope
Windows	1.00E-06	932.7873	0.0475	0.9908	0.1455	0.2183
OSX	2.58E-06	709.0172	1.1229	0.9975	0.2164	0.3246
Apache	8.80E-06	148.4182	0.3539	0.9943	0.0323	0.0485
IIS	1.62E-05	133.8943	0.4433	0.9879	0.0483	0.0724
IE	1.39E-06	744.4496	0.0592	0.9961	0.1285	0.1928
Firefox	2.56E-06	708.9164	0.0328	0.9870	0.2147	0.3221

Table 8.6: Information estimated by AML - Transition points

	TP1	MP	TP2
Windows	2004-03-29	2008-02-03	2011-12-11
OSX	2005-08-07	2007-07-27	2009-07-15
Apache	2001-10-03	2004-07-06	2007-04-10
IIS	1999-08-26	2001-04-25	2002-12-24
IE	2003-09-09	2007-03-03	2010-08-25
Firefox	2006-10-02	2008-09-26	2010-09-21

estimated a bit later than the time point supposed to be, due to the code evolutions. The reason, for OSX, is that the slope has very strong linear behavior in the actual data, so that AML model hard to pinpoint the transition points. For IIS, on February 2008, IIS 7 has been released, so that apparently previous saturation phase turned into stairway-like pattern, and the prediction has been misplaced. When we consider currently observed strong linear trends across the discovery patterns, transition points, especially MP and TP2, in Table 8.6 should not be accurate estimations because of the continuous software evolving which should trigger shifting of transition points.

8.5 Things that influence on slope level

Although upper and lower boundaries for linear rates could be estimated as shown in the previous section and there are already some VDMs available, it would

be nice to estimate the linear rate more precisely with more less complex relationship for taking advantage of the newly appeared linear pattern.

By its nature, a quantitative vulnerability discovery model requires empirical observations on the relationship between a growth trend in actual data and a set of factors believed to influence on the trend. At the beginning of the investigation, usually, the relationship is unknown or unclear, so that researchers generate some assumptions providing a starting point which are reasonable, or observed vaguely from the actual data but are not confirmed in a scientific way yet. For the starters, we have also some intuitive and vaguely observed assumptions that might influence on the slopes:

- Skills of programming team & maturity of vendor
- Number of installations with code sharing
- Source code edit frequency
- Software type

First, attitude of a vendor and its developers toward secure programming practice should effect on the degree of slope. Experts agree that developers' skills are important factors influence on quality of products although there have not been good references quantitatively conducted. Therefore, skill of programming team should be in inverse proportion to the slope value. Also vendor's maturity in its field should also be matter for products' quality. The better skills developers have in the more security related mature environment, the gentler slope should be produced. The relationship could be expressed something like Figure 8.7.

Second, it is intuitive that the more number of installations causes the more number of vulnerabilities discovered. This is because, as the AML model already has claimed, a system would be more thoroughly tested with a bigger group of users

Vendor	Developer skill: Low	Average	High	Extremely high
Very immature	Value 1	Value 5	Value 9	Value 13
Immature	Value 2	Value 6	Value 10	Value 14
Mature	Value 3	Value 7	Value 11	Value 15
Very mature	Value 4	Value 8	Value 12	Value 16

Figure 8.7: Example matrix for referring developer and vendor levels

or testers which will demand more number of vulnerabilities found. Along with this intuition, as Figure 8.2 and 8.3 shown, as long as popular enough successive versions are released regularly, the saturation phase will not be seen. Therefore, there should be positive growth correlations between the slope and the number of installations backed up by code sharing with successive versions. As a result, code sharing also effect on the slope. The more codes are shared, the steeper slope should be appeared from the originated version. Figure 8.8 shows the market share for the six systems and Figure 8.9 represents the estimated number of installations by the same method used in Chapter 5.

Third, Zimmermann et al. (2010) empirically examined the effectiveness of classical software metrics to predict vulnerabilities and assess how well the measurements perform on Windows Vista. They measure the correlations between the metrics and the number of vulnerabilities. The study shows that all the correlations are less than 0.3, which is considered as small effect size. However, among them, the correlation between the frequency of source code editing and the number of vulnerabilities claims the highest value. A similar result was found by Meneely and Williams (2009) on Red Hat Linux. Hence, the more frequently developers edit source code, the better chances that vulnerabilities are introduced.

Lastly, the software type matters. For example, as we have seen and will see, the growth rates and slopes in the linear phase for Web browsers and OSes are steeper

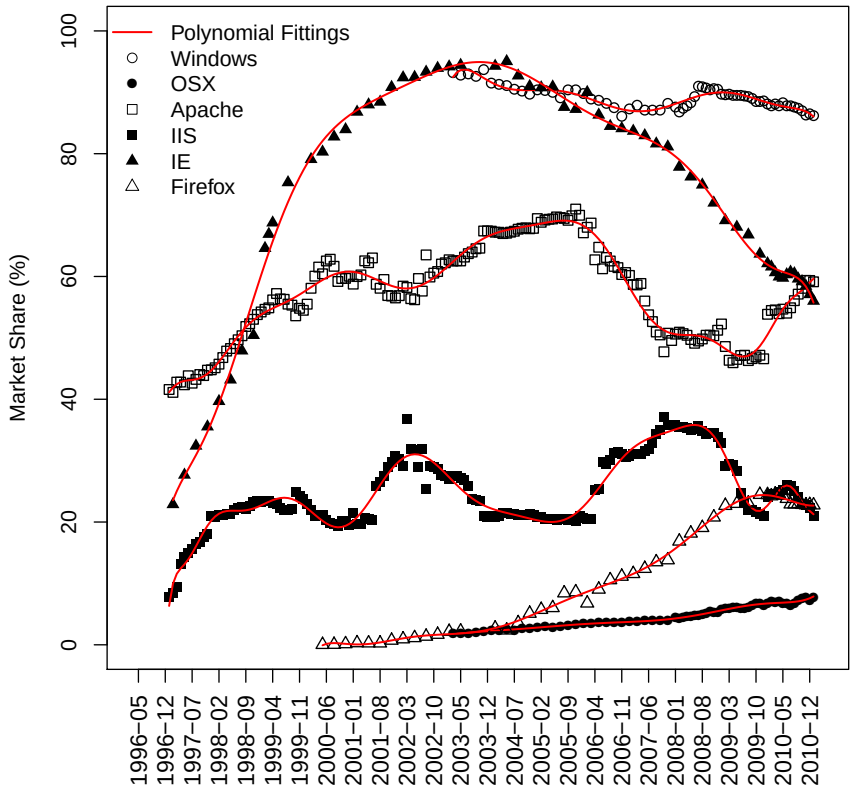


Figure 8.8: Market shares for the six systems

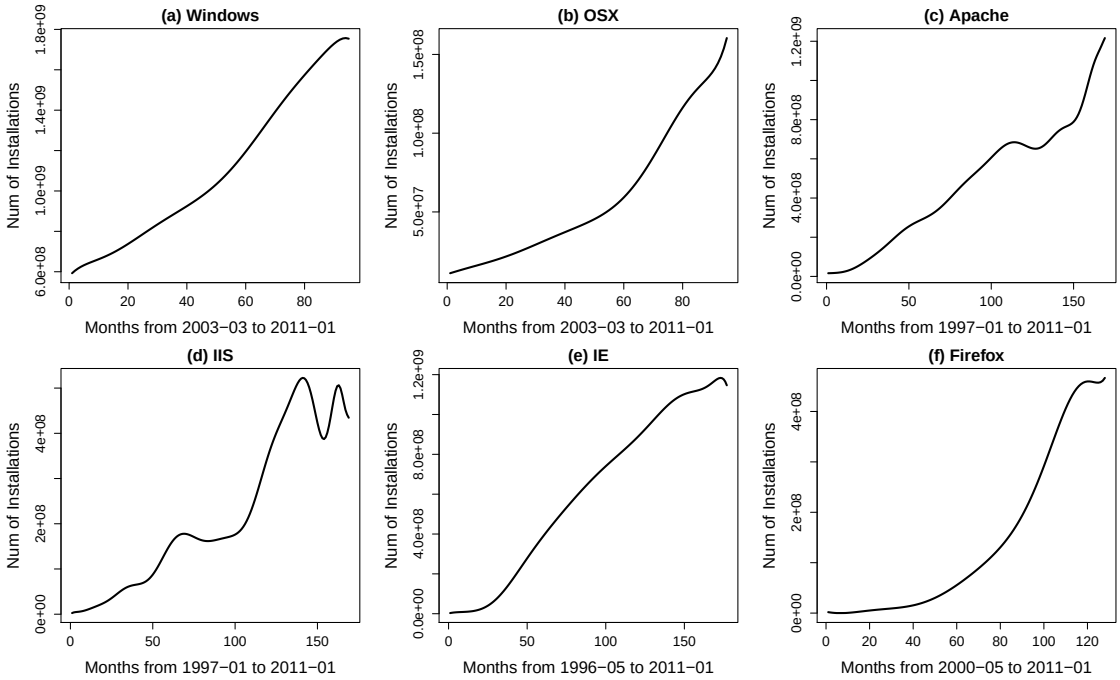


Figure 8.9: Estimated number of users

than other types of systems. Hence, there should be some empirical guidelines categorizing software systems and endowing with certain weights associated with numbers such as Figure 8.7. Software systems could be grouped into OSes, Web browsers, Web servers, Web applications, DBMSs, etc. After the classification, proper weights need to be associated.

Based on the observations above, we can derive the relationship between the linear growth rate or slope S and the elements influencing on it as following:

$$S \propto \frac{F_N, F_E, F_C}{F_S}$$

where F_N, F_E, F_C and F_S are the number of installations, editing frequency, software class, and skills of programers respectively. Now, the relationship between the number of vulnerabilities $\Omega(t)$ and the time t should be linear with rate of slope S.

$$\begin{aligned} \Omega(t) &= St + k \\ &= f\left(\frac{F_N, F_E, F_C}{F_S}\right) + k \\ &= \alpha_1 F_N(t) + \alpha_2 \frac{dF_N(t)}{dt} + k \end{aligned}$$

where f is a properly chosen function which describes the slope connected to the elements, k is a constant which assists for selecting the y-axis intercept at time is zero. In the equation, more delicately, not only the number of current installations at time t $F_N(t)$, but also the change ratio $\frac{dF_N(t)}{dt}$ should effect on the vulnerability finding rate since rapidly added fresh looks tend to find more defects in a short time. The constants α_1 and α_2 are originated from F_C, F_E and F_S with some random experimental error terms.

8.6 Summary

In this chapter, the extended linear phase, currently and noticeably observed in many multi-version software systems, has been observed, and the reason why

it happens has been inferred. Further research is needed to verify the hypothesis claimed here the relationship between the constant growth rate and the elements influencing on it.

Chapter 9

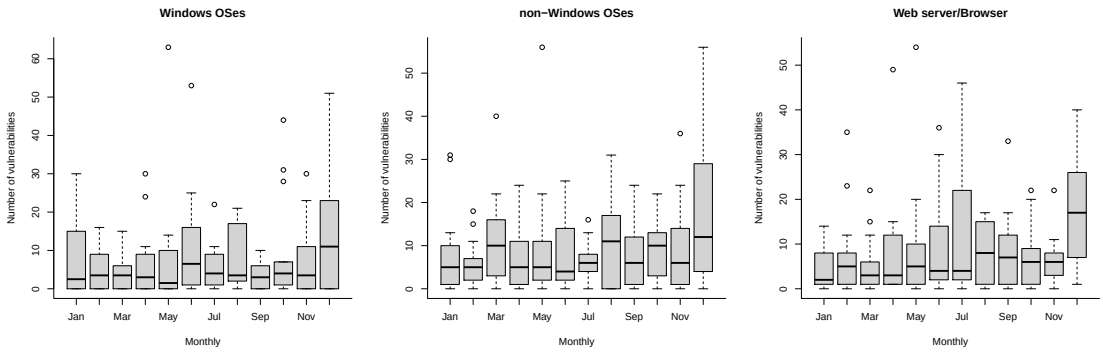
SEASONAL BEHAVIOR OF VULNERABILITY ACTIVITIES

Periodic behavior related to software vulnerability activities need to be taken into account for evaluating security risks. In this chapter, we examined datasets from mainly National Vulnerability Database (NVD) and Qualys[1] for annual variations of the vulnerability discovery processes in a multi-year life-cycle of popular software products and weekly periodicity in the Laws of Vulnerabilities reported in 2009 respectively.

For an accurate projection of vulnerability discovery process which is required to estimate the effort needed to develop patches for handling discovered vulnerabilities, a time series analysis that combines the periodic patterns may allow for us to predict the future trend more accurately. In the first part of the chapter, we carefully inspect the eighteen datasets of software systems (operating systems, Web servers and Web browsers) minded at NVD for annual seasonality in their vulnerability discovery processes. This first part analysis shows that there are indeed repetitive annual patterns.

The second part identifies the weekly periodicity and examines its statistical significance for datasets from the report called the Laws of Vulnerabilities (Qualys, 2009). The second part shows that the seven-day periodicity in presence of unpatched vulnerabilities as well as the exploitation pattern. This chapter examines

[1] `http://www.qualys.com`



(a) **(b)** **(c)**

Figure 9.1: Boxplots for each software group's cumulative number of vulnerabilities in month.

Table 9.1: Number of vulnerabilities for the eighteen software systems and observed period.

Win:	Win NT	Win XP	Win 2000	Server2003	Win 95	Win 98
Vul. #	269	294	380	207	46	88
Period	1995-2008	2000-2008	1997-2008	2002-2008	1997-2008	1999-2008
non-Win:	OSX	Solaris	HP-UX	RHL	RHEL	AIX
Vul. #	512	458	254	227	157	270
Period	1997-2008	1993-2008	1993-2008	1994-2007	1996-2008	1992-2008
Web:	Apache	IIS	IE	Firefox	Opera	Safari
Vul. #	132	138	495	369	129	110
Period	1996-2008	1996-2008	1997-2008	2003-2008	1998-2008	2003-2008

the statistical significance of the periodic behaviors using the seasonal index approach. The autocorrelation function is also used to identify the exact periodicities. The observed results should be used to optimize resource allocations and for determination of risk.

9.1 Introduction

Figure 9.1 represents the boxplots for the number of vulnerabilities from the three software groups clustered by each cumulative month, and Table 9.1 shows the corresponding entire known number of vulnerabilities for each software system with the examined periods. Here, software systems are categorized into Windows OSes (Windows NT, Windows XP, Windows 2000, Windows Server 2003, Windows 95 and Windows 98), non-Windows OSes (MAC OSX, Solaris, HP-UX, Red Hot Linux, Red

Hot Linux Enterprise, and AIX), and Web servers/browsers (Apache Web server, IIS, Internet Explorer, Firefox, Opera, and Safari). Figure 9.2 shows the number of vulnerabilities found along with the calendar time by month. All the datasets were collected from the NVD in 2010. However, for a fair comparison among the twelve months, year 2009 and 2010 are not considered since from reporting a vulnerability to becoming an official entry in NVD takes some time.

In Figure 9.1, it is visually observed that all the three groups have December peak. Moreover, a bit weaker mid-year peak is also witnessed in Windows OSes and Web server/browser systems with small fraction of vulnerabilities reported between the two peaks. We will analyze those behaviors later in the chapter using statistical methods to see whether the differences are statistically significant or not.

Somewhat consistent fluctuations in the boxplots would suggest a possible seasonal pattern that might be taken into account to make more accurate predictions about the number of vulnerabilities expected to be discovered in a future period. We try to answer the question of existence of annual and weekly periodic patterns and its significance in the activities related to software vulnerabilities based on the available datasets in quantitative manner (Joh and Malaiya, 2009; Joh et al., 2010).

9.2 Related work on seasonality

Periodic behaviors such as seasonal or weekend effect is well established research area in other disciplines such as the stock market (Heston and Sadka, 2008), high-performance computing systems (Tran and Reed, 2004), epidemiology (Rios et al., 2000), power transmission (Salehian, 2003), marine biology (Maes et al., 2004), birth defects (Carrion-Baralt et al.), etc.

Stocks tend to have relatively higher returns for some specific calendar months. The higher return during November to April is termed the Halloween Effect[2]. The

[2]http://ssrn.com/abstract=901088

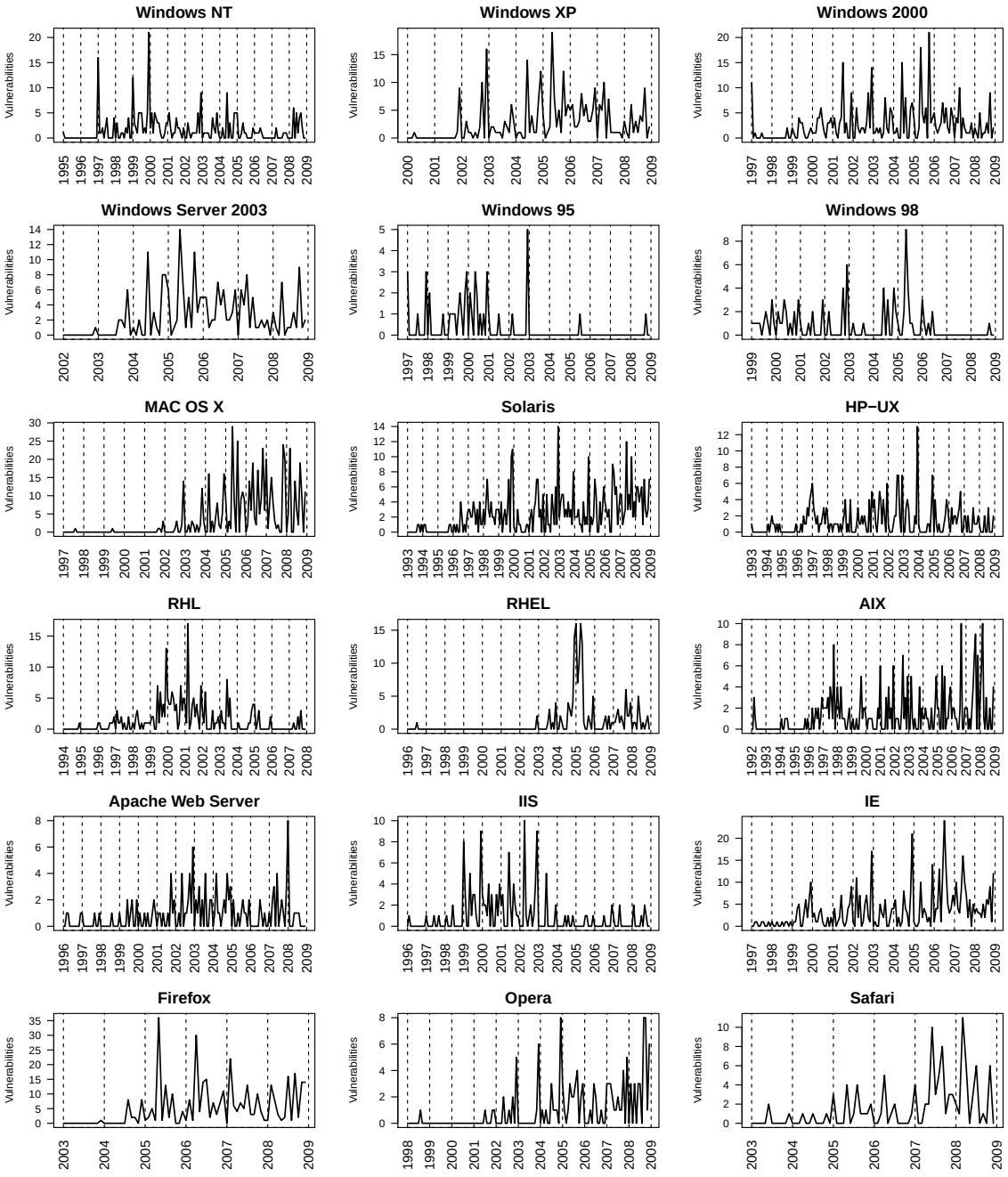


Figure 9.2: Run chart for the number of vulnerabilities in each month along with the calendar time.

repetitive and distinctive pattern with lags of 12, 24, and 36 months in the data from Jegadeesh (1990) caused by persistent seasonal effect in stock returns have identified by Heston and Sadka (2008). The study could help for seasonal stock strategies. Meanwhile, Tran and Reed (2004), to improve software application performance, utilized the fact that application's bursty I/O patterns in scientific codes due to either periodic checkpoints or nested loops, and such bursty patterns could overflow system resources. Hence, they tried to predict the I/O request patterns using time series models. These kind of access pattern forecasts could be used to make pre-fetch decisions during application execution time.

In epidemiology, Rios et al. (2000) tried to find out whether pulmonary tuberculosis has an annual seasonal pattern or not by using Autoregressive Integrated Moving Average (ARIMA) time series model, especially Autocorrelation Function (ACF) and Partial ACF. The seasonal trend for the disease could be caused by rising in indoor activities in winter which increases the risk of exposure of healthy persons to be bacilli by other infected persons. One of the other reasons could be infections of viral aetiology are more frequent and cause immunological deficiency. The model developed by the authors could express surveillance whether an incidence is greater than forecast by the model or not, so that it could be used to assess a quality of the preventive measures.

For a power transmission in utilities, Salehian (2003) attempted to model thermal rating patterns which is influenced by whether. He also used ARIMA time series model for the forecasting. Forecasting the thermal capabilities of the line would allow to meet contractual power delivery obligations. Also, in marine biology, Maes et al. (2004) tried to show what kind of elements are affecting the fish abundance which are exposed to great environmental variability such as dissolved oxygen, temperature, water quality, salinity, prey, etc. Thus, the life cycles of marine organisms have clear seasonal patterns in growth, reproduction and abundance.

Lastly, Carrion-Baralt et al. investigated whether a schizophrenic birth, which occurs more frequently in winter season, is really due to the severe winter temperatures as known previously. Unlike previous studies, they conducted their work in a tropical island having no severe winter time, and derive the conclusion that the extreme temperatures are not a sufficient explanation.

As shown above, seasonal periodicity analysis is a well-known statistical approach in the repertoire of many researchers and analysts in those disciplines.

9.3 Statistical methods for seasonality analysis

Seasonal index states how much the average for a particular period tends to be above (or below) the expected value. The monthly seasonal index values[3] are given by:

$$s_i = \frac{d_i}{d} \tag{9.1}$$

where, s_i is the seasonal index for i^{th} month, d_i is the mean value of i^{th} month, and d is a grand average. Hence, for example, a monthly seasonal index of 1.25 indicates that the expected value for that month is 25% greater than 1/12 of the overall average where the expected value is 1. To see whether the seasonal indices are statistically significant, Chi-square test has been also conducted. To evaluate the significance of non-uniformity of the distribution in calculated indices, we conducted the test for the grand total of each month against the expected value (total vulnerabilities divided by 12). For more details about the test, see Chapter 7.4.

Furthermore, to pinpoint which month's seasonal index is statistically greater or less than others, Analysis Of Variance (ANOVA) with Fisher's least significant difference (LSD) tests has been conducted on the calculated seasonal indices. For other examples of ANOVA & LSD in this dissertation, see Chapter 7.5.

[3] `http://home.ubalt.edu/ntsbarsh/Business-stat/stat-data/Forecast.htm#rseasonindex`

When there are twelve seasonal indices from each month, and μ_i is mean seasonal index value for month i. Then, LSD is testing the null hypotheses of $\mu_i = \mu_j$ where $1 \leq \{i \text{ and } j\} \leq 12$, i and j are integer, and $i \neq j$. When calculated $LSD_{i,j}$ is $|\mu_i - \mu_j| \geq LSD_{i,j}$, then the null hypothesis of $\mu_i = \mu_j$ will be rejected, and it confirms there is statistically significant difference between the two groups. $LSD_{i,j}$ can be obtained by (Ott and Longnecker, 2000):

$$LSD_{i,j} = t_{\alpha/2,d.f.} \sqrt{MS\left(\frac{1}{n_i} + \frac{1}{n_j}\right)} \tag{9.2}$$

where $d.f.$ is degrees of freedom, MS is mean square value from ANOVA test, n_i is the number of observation in group i. $t_{\alpha/2,d.f.}$ value can be obtained from the Student's t-distribution table.

The other approach to characterize periodicity is to use the autocorrelation function (ACF). ACF analysis gives us specific relationship information between related time units, such as month or day. With time series values of $z_b, z_{b+1}, ..., z_n$, the ACF at time lag k, denoted by r_k , is (Bowerman and O'connell, 1987):

$$r_k = \frac{\sum_{t=b}^{n-k}(z_t - \bar{z})(z_{t+k} - \bar{z})}{\sum_{t=b}^{n}(z_t - \bar{z})^2} \text{ , } where\ \bar{z} = \frac{\sum_{t=b}^{n} z_t}{n - b + 1} \tag{9.3}$$

ACF measures the linear relationship between time series observations separated by a lag of k time units. When an ACF value is located outside of chosen upper or lower confidence intervals, there is a statistically significant relationship associated with that time lag. An event occurring at time $t + k$ $(k > 0)$ is said to lag behind an event occurring at time t, the extent of the lag being k.

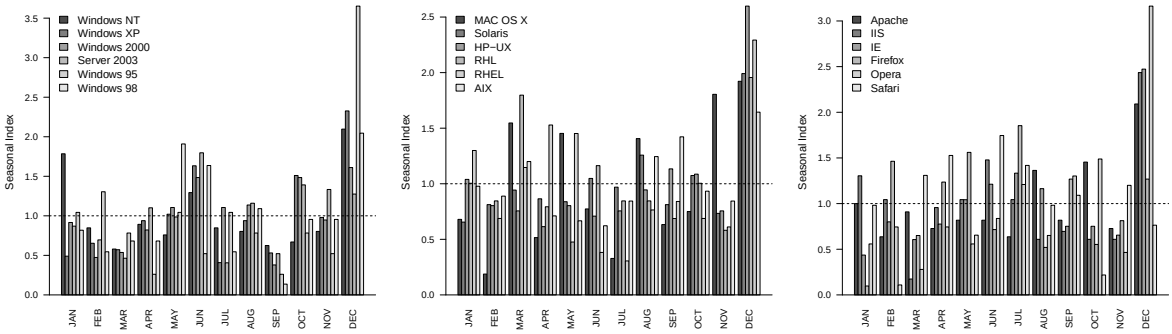
In the chapter, the seasonal index analysis backed up by the Chi-square test, ANOVA with LSD test, and the ACF analysis are applied to verify annual and weekly periodic behavior discussed below.

Table 9.2: Vulnerability discovery process seasonal indices.

	Win NT	Win XP	Win 2K	Server 2003	Win 95	Win 98
JAN	1.784387	0.489796	0.915789	0.869565	1.043478	0.818182
FEB	0.847584	0.653061	0.473684	0.695652	1.304348	0.545455
MAR	0.579926	0.571429	0.536842	0.463768	0.782609	0.681818
APR	0.892193	0.938776	0.821053	1.101449	0.26087	0.681818
MAY	0.758364	1.020408	1.105263	0.985507	1.043478	1.909091
JUN	1.29368	1.632653	1.484211	1.797101	0.521739	1.636364
JUL	0.847584	0.408163	1.105263	0.405797	1.043478	0.545455
AUG	0.802974	0.938776	1.136842	1.15942	0.782609	1.090909
SEP	0.624535	0.530612	0.378947	0.521739	0.26087	0.136364
OCT	0.669145	1.510204	1.484211	1.391304	0.782609	0.954545
NOV	0.802974	0.979592	0.947368	1.333333	0.521739	0.954545
DEC	2.096654	2.326531	1.610526	1.275362	3.652174	2.045455
χ_c^2	19.67514	19.67514	19.67514	19.67514	19.67514	19.67514
χ_s^2	56.60595	87.30612	57.05263	34.33333	33.82609	27.36364
P-value	3.93E-08	5.61E-14	3.25E-08	0.000319	0.000386	0.004048

	OSX	Solaris	HP-UX	RHL	RHEL	AIX
JAN	0.679688	0.655022	1.03937	1.004405	1.299363	0.977778
FEB	0.1875	0.812227	0.80315	0.845815	0.687898	0.888889
MAR	1.546875	0.943231	0.755906	1.797357	1.146497	1.2
APR	0.515625	0.864629	0.614173	0.792952	1.528662	0.711111
MAY	1.453125	0.838428	0.80315	0.475771	1.452229	0.666667
JUN	0.773438	1.048035	0.708661	1.162996	0.382166	0.622222
JUL	0.328125	0.969432	0.755906	0.845815	0.305732	0.844444
AUG	1.40625	1.257642	0.944882	0.845815	0.764331	1.244444
SEP	0.632813	0.812227	1.133858	0.687225	0.840764	1.422222
OCT	0.75	1.074236	1.086614	1.004405	0.687898	0.933333
NOV	1.804688	0.733624	0.755906	0.581498	0.611465	0.844444
DEC	1.921875	1.991266	2.598425	1.955947	2.292994	1.644444
χ_c^2	19.67514	19.67514	19.67514	19.67514	19.67514	19.67514
χ_s^2	164.875	52.131	65.08661	42.33921	46.5414	24.66667
P-value	1.34E-29	2.58E-07	1.04E-09	1.41E-05	2.59E-06	0.010197

	Apache	IIS	IE	Firefox	Opera	Safari
JAN	1	1.304348	0.436364	0.097561	0.55814	0.981818
FEB	0.636364	1.043478	0.8	1.463415	0.744186	0.109091
MAR	0.909091	0.173913	0.606061	0.650407	0.27907	1.309091
APR	0.727273	0.956522	0.775758	1.235772	0.744186	1.527273
MAY	0.818182	1.043478	1.042424	1.560976	0.55814	0.654545
JUN	0.818182	1.478261	1.212121	0.715447	0.837209	1.745455
JUL	0.636364	1.043478	1.333333	1.853659	1.209302	1.418182
AUG	1.363636	0.608696	1.163636	0.520325	0.651163	0.981818
SEP	0.818182	0.695652	0.751515	1.268293	1.302326	1.090909
OCT	1.454545	0.608696	0.751515	0.552846	1.488372	0.218182
NOV	0.727273	0.608696	0.654545	0.813008	0.465116	1.2
DEC	2.090909	2.434783	2.472727	1.268293	3.162791	0.763636
χ_c^2	19.67514	19.67514	19.67514	19.67514	19.67514	19.67514
χ_s^2	22.54545	41.65217	130.3333	90.41463	70.16279	25.05455
P-value	0.020472	1.86E-05	1.49E-22	1.38E-14	1.14E-10	0.008951



(a) **(b)** **(c)**

Figure 9.3: Seasonal indices of Windows OSes, non-Windows OSes and Web related software systems. Individual values are in Table 9.2

9.4 Annual variations in vulnerability discovery processes

In Figure 9.3, the mid-year (summer) and the year-end (winter) months appear to have most of the peaks, suggesting the possibility of seasonality. We examine the significance of this observed annual seasonality in this section. We will examine the null hypothesis of H_0: all the seasonal indices for the twelve months are not significantly different each other. The same methods will be used in the next section for checking the weekly periodic behavior in the exploitation pattern and preferences of unpatched vulnerabilities.

9.4.1 Seasonal index analysis

A time series data is not uniformly distributed and periodic patterns are present in a dataset when certain months have significantly more incidences of reported vulnerabilities than other months. Table 9.2 shows seasonal indices for the twelve months from the each system. A seasonal index describes how much the average for that particular period tends to be above or below the grand average. In Figure 9.3, for the Windows OSes, seasonal index values for mid-year (June) and year-end (December) have higher values, significantly above 1.0 which is the expected value.

Consequently, months between the two peaks tend to be less than the expected seasonal index of one, especially, March and September.

Unlike the Windows OSes, clear consistent mid-year seasonal patterns are not found among the indices from non-Windows OSes and Web related software systems. However, still some months tend to have more vulnerabilities than others. Also, all of them have higher incidences in December. For the Web related software systems, IIS and IE data displays a pattern similar to the Windows OSes which are the parent platforms for the two Web related systems. The mid-year and year-end periods tend to have more vulnerabilities than other months. Indices from Apache server show a pattern nearly the opposite of IIS's, in general. In December, all indices are above the expected value with an exception of Safari.

The mid-year peak may explain the higher third quarter advisories for Microsoft products (Jaquith, 2007). To evaluate the significance of non-uniformity of the distribution among the seasonal indices, we conducted Chi-square test for the grand total of each month against the expected value (total vulnerabilities divided by 12). In the paper, level of alpha chosen is 0.05. Hence, when the P-value of the Chi-square test is below than 0.05, the null hypothesis will be rejected. In Table 9.2, we can see that the systems yield extremely small p-values, thus, we have a strong evidence of non-uniform distributions of vulnerability discovery rates, where the null hypothesis is that there is no seasonality in the dataset.

To see which months have indeed statistically greater indices than other months specifically, ANOVA test with Fisher's LSD test are conducted on the mean index values from each month grouped by the software categories. As mentioned in Chapter 7.5, since ANOVA test only can tell whether the mean index values among the twelve months are the same or not, Fisher's LSD test is also conducted after confirming the inequality performance from the ANOVA test.

Table 9.3, 9.4 and 9.5 show ANOVA tables for each software group. Here, the alpha level is 0.05 for the F-test. To be statistically significant, the F value need

Table 9.3: ANOVA table for seasonal indices from Windows OSes.

Windows	SS	df	MS	F	P-value	F_{crit}
Between Groups	13.5128	11	1.228436	8.259795	1.37E-08	1.952212
Within Groups	8.923489	60	0.148725			
Total	22.43629	71				

Table 9.4: ANOVA table for seasonal indices from non-Windows OSes.

non-Windows	SS	df	MS	F	P-value	F_{crit}
Between Groups	8.977445	11	0.816131	7.873355	2.98E-08	1.952212
Within Groups	6.219443	60	0.103657			
Total	15.19689	71				

Table 9.5: ANOVA table for seasonal indices from Web server/browser

Web	SS	df	MS	F	P-value	F_{crit}
Between Groups	8.904923	11	0.809538	4.005469	0.000216	1.952212
Within Groups	12.1265	60	0.202108			
Total	21.03142	71				

to be greater than the corresponding F critical with small enough P-value (smaller than 0.05). In the tables, F values, greater than the F critical value, confirm that not all the months have equal mean values. In addition, F value from the Windows OSes is bigger than others implying that seasonal fluctuations in Windows OSes are more dynamic.

Table 9.6, 9.7 and 9.8 are showing the absolute values in differences among mean values from seasonal indices and significance of pairwise comparisons. In the tables, shaded cells represent statistically significant differences. To be shaded cell, differences between two compared mean values need to be greater than the corresponding calculated LSD value. The LSD values for each table are $LSD_{Windows}$ = 0.4453, $LSD_{non-Windows}$ = 0.3717 and LSD_{Web} = 0.5191 respectively.

Table 9.6 confirms that, in Windows OSes, i) December is greater than all the other months, ii) June is greater than July, February, March, April, November, and September, iii) September is lesser than January, May, June, August, October, and

Table 9.6: LSD test for Windows seasonal index; $\text{LSD}_{Windows} = 0.4453$

Month		JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
	Mean	0.9869	0.7533	0.6027	0.7827	1.1370	1.3943	0.7260	0.9853	0.4088	1.1320	0.9233	2.1678
JAN	0.9869	0	0.2336	0.3841	0.2042	0.1502	0.4074	0.2609	0.0016	0.5780	0.1451	0.0636	1.1809
FEB	0.7533		0	0.1506	0.0294	0.3837	0.6410	0.0273	0.2320	0.3445	0.3787	0.1700	1.4145
MAR	0.6027			0	0.1800	0.5343	0.7916	0.1232	0.3825	0.1939	0.5293	0.3205	1.5651
APR	0.7827				0	0.3543	0.6116	0.0567	0.2026	0.3738	0.3493	0.1406	1.3851
MAY	1.1370					0	0.2573	0.4111	0.1518	0.7282	0.0050	0.2138	1.0308
JUN	1.3943						0	0.6683	0.4090	0.9854	0.2623	0.4710	0.7735
JUL	0.7260							0	0.2593	0.3171	0.4060	0.1973	1.4418
AUG	0.9853								0	0.5764	0.1467	0.0620	1.1825
SEP	0.4088									0	0.7232	0.5144	1.7589
OCT	1.1320										0	0.2087	1.0358
NOV	0.9233											0	1.2445
DEC	2.1678												0

Table 9.7: LSD test for non-Windows seasonal index; $\text{LSD}_{non-Windows} = 0.3717$

Month		JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
	Mean	0.9426	0.7042	1.2316	0.8379	0.9482	0.7829	0.6749	1.0772	0.9215	0.9227	0.8886	2.0675
JAN	0.9426	0	0.2384	0.2890	0.1047	0.0056	0.1597	0.2677	0.1346	0.0211	0.0199	0.0540	1.1249
FEB	0.7042		0	0.5274	0.1336	0.2440	0.0787	0.0293	0.3730	0.2173	0.2185	0.1844	1.3632
MAR	1.2316			0	0.3938	0.2834	0.4487	0.5567	0.1544	0.3101	0.3089	0.3430	0.8358
APR	0.8379				0	0.1104	0.0549	0.1629	0.2394	0.0837	0.0849	0.0507	1.2296
MAY	0.9482					0	0.1653	0.2733	0.1290	0.0267	0.0255	0.0596	1.1193
JUN	0.7829						0	0.1080	0.2943	0.1386	0.1398	0.1057	1.2846
JUL	0.6749							0	0.4023	0.2466	0.2478	0.2137	1.3926
AUG	1.0772								0	0.1557	0.1545	0.1886	0.9903
SEP	0.9215									0	0.0012	0.0329	1.1460
OCT	0.9227										0	0.0341	1.1447
NOV	0.8886											0	1.1789
DEC	2.0675												0

Table 9.8: LSD test for Web server/browser seasonal index; $\text{LSD}_{Web} = 0.5191$

Month		JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
	Mean	0.7297	0.7994	0.6546	0.9945	0.9463	1.1344	1.2491	0.8815	0.9878	0.8457	0.7448	2.0322
JAN	0.7297	0	0.0697	0.0751	0.2648	0.2166	0.4047	0.5193	0.1518	0.2581	0.1160	0.0151	1.3025
FEB	0.7994		0	0.1448	0.1950	0.1469	0.3350	0.4496	0.0821	0.1884	0.0463	0.0546	1.2328
MAR	0.6546			0	0.3399	0.2917	0.4798	0.5944	0.2269	0.3332	0.1911	0.0902	1.3776
APR	0.9945				0	0.0482	0.1400	0.2546	0.1129	0.0067	0.1488	0.2497	1.0377
MAY	0.9463					0	0.1882	0.3028	0.0647	0.0415	0.1006	0.2015	1.0859
JUN	1.1344						0	0.1146	0.2529	0.1466	0.2888	0.3897	0.8977
JUL	1.2491							0	0.3675	0.2612	0.4034	0.5043	0.7831
AUG	0.8815								0	0.1063	0.0359	0.1368	1.1506
SEP	0.9878									0	0.1421	0.2430	1.0444
OCT	0.8457										0	0.1009	1.1865
NOV	0.7448											0	1.2874
DEC	2.0322												0

November, iv) October is greater than March and September, and v) May is greater than March. Table 9.7 confirms that, in non-Windows OSes, i) December is greater than all the other months, ii) August is greater than February and July, and iii) March is greater than February, April, June, and July. And finally, for the Web related software systems, Table 9.8 shows that December is greater than all the other month, July is greater than January and March.

9.4.2 Autocorrelation function analysis

The autocorrelation function (ACF) in time series analysis is calculated by computing the correlation between a variable value and the successive values of the same variable after some time lags. In other words, ACF measures the linear relationship between time series observations separated by a lag of k time units (Bowerman and O'connell, 1987; Rios et al., 2000). When an ACF value is located outside of defined confidence intervals at a lag k, there is a significant relationship associated with that time lag.

Table 9.9, 9.10 and 9.11 show the ACF values with 95% confidence intervals for the three software groups respectively. In the tables, **Bold** fonts indicate outside of confidence intervals and ^{superscripts} represent time lags from 0 to 23 in month. For the Windows OSes, since the mid-year and year-end periods tend to have the majority of big seasonal indices, we expect that lags corresponding to about sixth month or its multiple would have their corresponding ACF values outside the confidence interval. In Table 9.9, for Windows NT, the lags for 0, 5, 6 and 11 months are outside of confidence interval; in other words, there are strong autocorrelations with the lags that are multiple of around six confirming a seasonal pattern.

For Windows XP, lags for 0, 2, 5, 6, 12 and 18 months, for Windows 2000, 0, 5 and 18 months, for Windows Server 2003, for 0, 5 and 6 months are significantly different from zero of ACF which confirms a seasonal pattern. For Windows 95, lags

Table 9.9: Individual ACF values for Windows OSes

Windows NT; 95% confidence interval = (-.1512145 .1512145)											
1^{0}	.109^{1}	.12^{2}	.036^{3}	.054^{4}	**.208**^{5}	**.156**^{6}	.112^{7}	-.006^{8}	-.016^{9}	.027^{10}	**.218**^{11}
-.002^{12}	.098^{13}	.051^{14}	.014^{15}	-.035^{16}	-.043^{17}	.027^{18}	-.004^{19}	-.023^{20}	-.034^{21}	-.047^{22}	-.06^{23}
Windows XP; 95% confidence interval = (-.1885976, .1885976)											
1^{0}	.079^{1}	**.199**^{2}	.035^{3}	.129^{4}	**.265**^{5}	**.265**^{6}	.09^{7}	.038^{8}	.098^{9}	.154^{10}	.154^{11}
.201^{12}	.072^{13}	.119^{14}	-.078^{15}	.086^{16}	-.07^{17}	**.295**^{18}	-.025^{19}	.13^{20}	-.06^{21}	-.012^{22}	.089^{23}
Windows 2K; 95% confidence interval = (-.1633303, .1633303)											
1^{0}	-.007^{1}	.111^{2}	.033^{3}	-.021^{4}	**.28**^{5}	.098^{6}	.045^{7}	.058^{8}	.07^{9}	.129^{10}	.128^{11}
.107^{12}	.047^{13}	.074^{14}	-.006^{15}	.085^{16}	.074^{17}	**.191**^{18}	-.084^{19}	.094^{20}	-.044^{21}	-.006^{22}	.068^{23}
Windows Server 2003; 95% confidence interval = (-.2138496, .2138496)											
1^{0}	.124^{1}	.206^{2}	.036^{3}	.187^{4}	**.346**^{5}	**.33**^{6}	.196^{7}	.075^{8}	.163^{9}	.134^{10}	.143^{11}
.144^{12}	.078^{13}	.212^{14}	-.098^{15}	.126^{16}	-.098^{17}	.179^{18}	.035^{19}	.03^{20}	-.08^{21}	-.049^{22}	.007^{23}
Windows 95; 95% confidence interval = (-.1633303, .1633303)											
1^{0}	.111^{1}	.143^{2}	**.178**^{3}	.111^{4}	.143^{5}	**.23**^{6}	**.199**^{7}	.067^{8}	.142^{9}	.152^{10}	.074^{11}
.062^{12}	**.17**^{13}	.088^{14}	.065^{15}	.031^{16}	.085^{17}	-.069^{18}	.039^{19}	.027^{20}	.037^{21}	.028^{22}	.005^{23}
Windows 98; 95% confidence interval = (-.1789194, .1789194)											
1^{0}	.148^{1}	.127^{2}	.039^{3}	-.057^{4}	.133^{5}	.126^{6}	.1^{7}	.061^{8}	.105^{9}	.046^{10}	.053^{11}
.08^{12}	-.023^{13}	-.041^{14}	-.062^{15}	-.167^{16}	-.01^{17}	-.059^{18}	-.057^{19}	.045^{20}	-.124^{21}	.007^{22}	-.074^{23}

Table 9.10: Individual ACF values for non-Windows OSes

MAC OSX; 95% confidence interval = (-.1633303, .1633303)											
1^{0}	**.211**^{1}	.156^{2}	**.457**^{3}	**.273**^{4}	**.289**^{5}	**.299**^{6}	**.322**^{7}	**.232**^{8}	**.278**^{9}	**.352**^{10}	**.24**^{11}
.371^{12}	**.209**^{13}	**.252**^{14}	**.344**^{15}	**.167**^{16}	**.33**^{17}	**.248**^{18}	**.265**^{19}	**.315**^{20}	**.169**^{21}	**.178**^{22}	**.196**^{23}
Solaris; 95% confidence interval = (-.1414482, .1414482)											
1^{0}	**.226**^{1}	**.252**^{2}	**.204**^{3}	**.296**^{4}	**.202**^{5}	**.197**^{6}	.113^{7}	**.221**^{8}	**.189**^{9}	**.198**^{10}	.072^{11}
.244^{12}	.057^{13}	**.248**^{14}	.133^{15}	**.177**^{16}	**.171**^{17}	**.148**^{18}	**.191**^{19}	**.294**^{20}	**.162**^{21}	.099^{22}	.02^{23}
HP-UX; 95% confidence interval = (-.1414482, .1414482)											
1^{0}	**.212**^{1}	.125^{2}	.106^{3}	.061^{4}	-.039^{5}	.005^{6}	.056^{7}	.107^{8}	.081^{9}	-.009^{10}	.096^{11}
.301^{12}	-.079^{13}	.056^{14}	**.157**^{15}	.03^{16}	-.023^{17}	-.032^{18}	.003^{19}	-.056^{20}	-.046^{21}	.025^{22}	.058^{23}
RHL; 95% confidence interval = (-.1512145, .1512145)											
1^{0}	**.221**^{1}	**.347**^{2}	**.353**^{3}	**.318**^{4}	**.306**^{5}	**.329**^{6}	.15^{7}	**.193**^{8}	**.265**^{9}	**.249**^{10}	**.21**^{11}
.244^{12}	.114^{13}	**.186**^{14}	**.318**^{15}	.079^{16}	**.259**^{17}	.108^{18}	**.189**^{19}	.109^{20}	**.208**^{21}	.027^{22}	.073^{23}
RHEL; 95% confidence interval = (-.1569227, .1569227)											
1^{0}	**.697**^{1}	**.488**^{2}	**.499**^{3}	**.466**^{4}	**.313**^{5}	.109^{6}	.103^{7}	**.188**^{8}	.087^{9}	.001^{10}	.032^{11}
.063^{12}	.054^{13}	.01^{14}	.012^{15}	.073^{16}	.055^{17}	.004^{18}	.018^{19}	.015^{20}	.059^{21}	.004^{22}	.01^{23}
AIX; 95% confidence interval = (-.1372249, .1372249)											
1^{0}	.117^{1}	**.178**^{2}	**.149**^{3}	**.15**^{4}	.112^{5}	**.178**^{6}	**.231**^{7}	.035^{8}	.097^{9}	.104^{10}	.046^{11}
.121^{12}	.014^{13}	**.149**^{14}	-.006^{15}	.098^{16}	-.01^{17}	.066^{18}	.086^{19}	-.02^{20}	**.141**^{21}	-.048^{22}	.012^{23}

Table 9.11: Individual ACF values for Web servers/browsers

Apache Web Server; 95% confidence interval = (-.1569227, .1569227)											
1^{0}	.03^{1}	**.187**^{2}	.072^{3}	.144^{4}	.093^{5}	.046^{6}	**.278**^{7}	.083^{8}	.132^{9}	.053^{10}	.122^{11}
.106^{12}	-.012^{13}	.089^{14}	.058^{15}	.006^{16}	.046^{17}	.127^{18}	-.07^{19}	.067^{20}	-.14^{21}	**.19**^{22}	-.037^{23}
IIS; 95% confidence interval = (-.1569227, .1569227)											
1^{0}	.134^{1}	-.002^{2}	.054^{3}	**.178**^{4}	.121^{5}	**.181**^{6}	**.366**^{7}	**.172**^{8}	.009^{9}	**.164**^{10}	**.188**^{11}
.099^{12}	.038^{13}	**.245**^{14}	-.001^{15}	.079^{16}	.085^{17}	**.285**^{18}	-.038^{19}	-.036^{20}	.104^{21}	.013^{22}	.125^{23}
IE; 95% confidence interval = (-.1633303, .1633303)											
1^{0}	**.24**^{1}	.129^{2}	.122^{3}	**.208**^{4}	**.22**^{5}	.163^{6}	**.199**^{7}	.112^{8}	.054^{9}	.104^{10}	.138^{11}
.257^{12}	.064^{13}	.144^{14}	.039^{15}	.063^{16}	.011^{17}	.148^{18}	**.202**^{19}	.017^{20}	-.026^{21}	.014^{22}	.02^{23}
Firefox; 95% confidence interval = (-.2309840, .2309840)											
1^{0}	-.054^{1}	**.344**^{2}	.092^{3}	.105^{4}	.196^{5}	-.146^{6}	.189^{7}	.058^{8}	.195^{9}	.081^{10}	.227^{11}
.076^{12}	.079^{13}	.097^{14}	-.045^{15}	.01^{16}	.017^{17}	-062.^{18}	.128^{19}	-.103^{20}	119.^{21}	.036^{22}	-.103^{23}
Opera; 95% confidence interval = (-.1705930, .1705930)											
1^{0}	**.216**^{1}	**.293**^{2}	**.236**^{3}	.156^{4}	**.174**^{5}	.156^{6}	**.199**^{7}	**.174**^{8}	**.202**^{9}	.164^{10}	.125^{11}
.354^{12}	.084^{13}	.135^{14}	.019^{15}	.081^{16}	.13^{17}	**.179**^{18}	**.174**^{19}	.146^{20}	.21^{21}	.027^{22}	.017^{23}
Safari; 95% confidence interval = (-.2309840, .2309840)											
1^{0}	.208^{1}	.137^{2}	**.343**^{3}	**.286**^{4}	.066^{5}	.175^{6}	**.266**^{7}	.204^{8}	.186^{9}	.165^{10}	.067^{11}
-.018^{12}	.004^{13}	.177^{14}	.028^{15}	-.104^{16}	.128^{17}	-049.^{18}	-.07^{19}	.011^{20}	-.027^{21}	-.006^{22}	.076^{23}

of 0, 3, 6, 7 and 13, and for Windows 98, only lag of 0 is outside of the confidence interval. As we expected, roughly six month periodicity are shown. The same approach had been applied in (Rios et al., 2000; Tran and Reed, 2004) to prove seasonality in their datasets belonging to other research areas.

In Table 9.10, for Mac OSX, except the lag number of 2, all the lags are outside of the confidence interval. For the SUN Solaris, also, all the lags are outside of the confidence interval except lags number of 7, 11, 13, 15, 22, and 23. In HP-UX, only lags number 0, 1, 12 and 15 are outside of the confidence interval. For Red Hat Linux, lags number 7, 13, 16, 18, 20, 22 and 23 are inside of the confidence interval. Red Hat Enterprise Linux has lags number 0, 1, 2, 4, 5, and 8 outside of confidence intervals. For IBM AIX, lags for 0, 2, 3, 4, 6, 7, 14, and 21 months are significantly different from zero. For non-Windows OSes, around 13 month leg is shown in common.

In Table 9.11, for the Apache Web server, lags for 0, 2, 7 and 22 are outside of the confidence intervals. For the IIS, lags for 0, 4, 6, 7, 8, 10, 11, 14, and 18 are outside of the confidence intervals. For the Internet Explorer, lags of 0, 1, 4, 5, 7, 12, and 19 are located outside of the boundary. For the Firefox, lags of 0, 2 are only located outside of the confidence intervals. For the Opera Web browser, lags number of 0, 1, 2, 3, 5, 7, 8, 9, 12, 18, and 19 are outside. For the Apple Safari, lags of 0, 3, 4, and 7 are outside of the confidence interval. The two Microsoft products tend to have higher ACF values approximately every six month while others tend to have up and down pattern bimonthly (lag 2), except Safari.

9.5 Seven-day periodicity in the vulnerability scan data

In this section, another periodic behavior related to the software vulnerabilities will be presented: a much shorter weekly periodic trend. Periodic scanning is a major part of the corporate security strategy. Some security service venders such

as Qualys (2009) collect a large amount of data which is quite valuable because it comes from real systems in major industrial organizations. In this section, we have mined one such data collection to examine periodicity in the presence of unpatched vulnerabilities and the exploitations in case of a worm.

Qualys has been involved in collecting and plotting such data for several years. In a 2009 report (Qualys, 2009), they have presented the data collected during 2008 which represents 104 million global vulnerability scans including 82 million internal scans and 22 million external Internet-based scans. The data involves encountering more than 72 million critical vulnerabilities among the 680 million detections. About 3500 organizations were scanned worldwide that represented major industry sectors of Financial, Health, Manufacturing, Service, and Wholesale/Retail. There are four distinct and quantifiable attributes about software vulnerabilities introduced by Qualys (2009):

1. Half-life: is the time interval measuring a reduction of a vulnerability's occurrence by half. Over time, this metric shows how successful efforts have been to eradicate a vulnerability. A shorter half-life indicates faster remediation.

2. Prevalence: notes the turnover rate of vulnerabilities in the "Top 20" list during a year. Prevalent vulnerabilities are dangerous because they represent ongoing potent risks to computing environments. Risk rises as the prevalence rate rises because of the larger total number of top 20 risks tallied during a year.

3. Persistence: measures the total life span of vulnerabilities. The fact that vulnerabilities persist and do not conclusively die off is a red flag for security administrators. It underscores the importance of patching all systems, and ensuring that old vulnerabilities are not inadvertently installed on new systems.

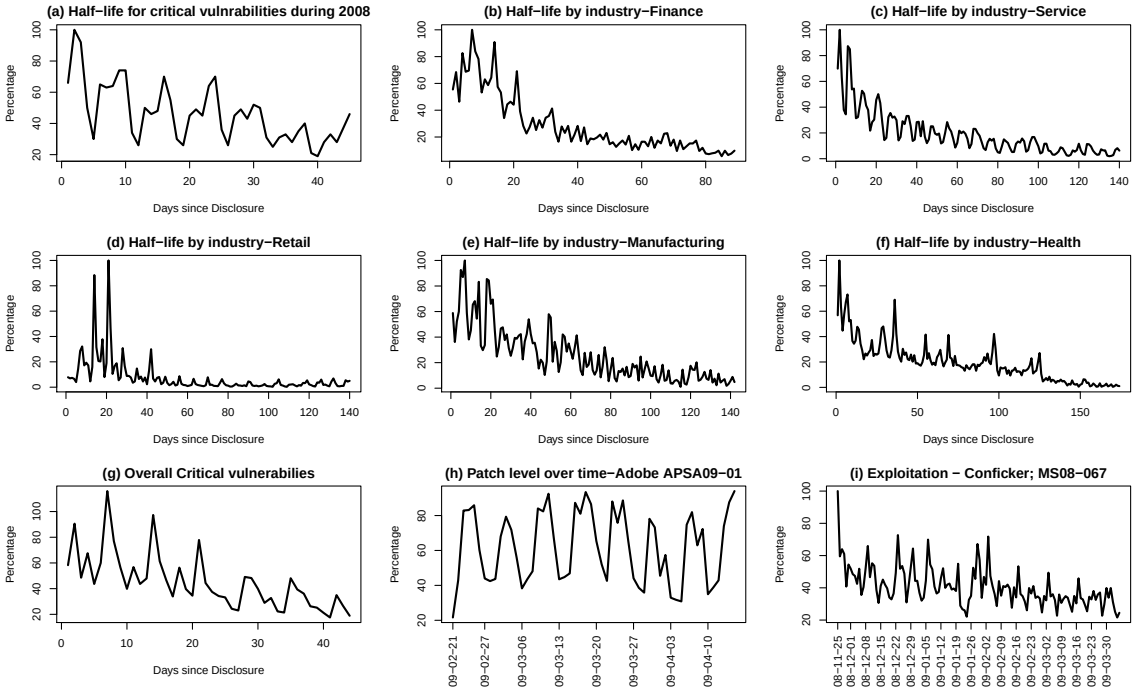


Figure 9.4: Run charts for the seven half-life plots (Critical vulnerabilities during 2008, by industries of finance, service, retail, manufacturing, and health, and overall critical vulnerabilities), patch level and Exploitation. (Qualys, 2009)

4. Exploitation: is the time interval between an exploit announcement and the first attack. This metric indicates how much reaction time you might get before someone figures out how to exploit the vulnerability. The worst scenario is a "zero day" attack because there is no reaction time.

Qualys calls the above four attributes as the Laws of Vulnerabilities. In 2009 Black Hat USA conference[4], they presented the following observations for the each law. An average duration of half-life is about 30 days, varying by industry sector. Prevalence has increased, with 60% remaining in the list in 2008 compared to 50% in 2004. Persistence remains virtually unlimited. And Exploitation is faster, often happening in less than ten days compared to 60 days in 2004.

We observed that most of the plots, in the report, visually suggest a short-term seven day periodicity shown in Figure 9.4. This section examines the statistical

[4]`http://www.blackhat.com/html/bh-usa-09/bh-us-09-main.html`

significance of the periodic pattern for the selected plots from the report by using the seasonal index and autocorrelation analyses.

Figure 9.4 shows nine run charts from the Qualys report. The plots are normalized using the maximum value set as 100%. Even visually, it is clearly observed that there are certain periodic patterns in the data. The values decline as the result of a remediation and go up due to new installations. Figure 9.5 displays corresponding ACF values from Figure 9.4. In the figure, lags of seven (or its multiples) tend to have higher values or outside of the ±95% of confidence intervals shown by the dashed lines. This demonstrates strong autocorrelations with lags that are multiples of seven days, which confirms a seven-day periodicity in the data.

Table 9.12 and 9.13 show the calculated weekly seasonal index values and Chi-square test results from Figure 9.5 respectively. Since there is no day of the week information except (h) and (i) from the figure, it is labeled as day1, day2, ..., day7 while the two cases are mentioned in specific weekdays (Sun through Sat). From (a) to (g), it is observed that values are tend to be clustered into high or low values consecutively, in general. For example, in (g), higher values appear in day7, day1, and day2 successively. For (h) and (i) weekdays (Mon ~ Thu) tend to have higher index values for the number of incidents. The observed patterns might be related to software vendors' patch release policies, organizations' patch managements, or individuals' behaviors. To be statistically significant for the calculated seasonal index values, the Chi-square statistic values need to be greater than the corresponding critical values with a small enough p-value. In the table, the small p-values confirm the non-uniform distributions.

9.6 Possible factors causing periodic behavior

Rescorla (2005) has mentioned a possible cause for the year-end seasonality. He has suggested that a large number of vulnerabilities reported during the year-end

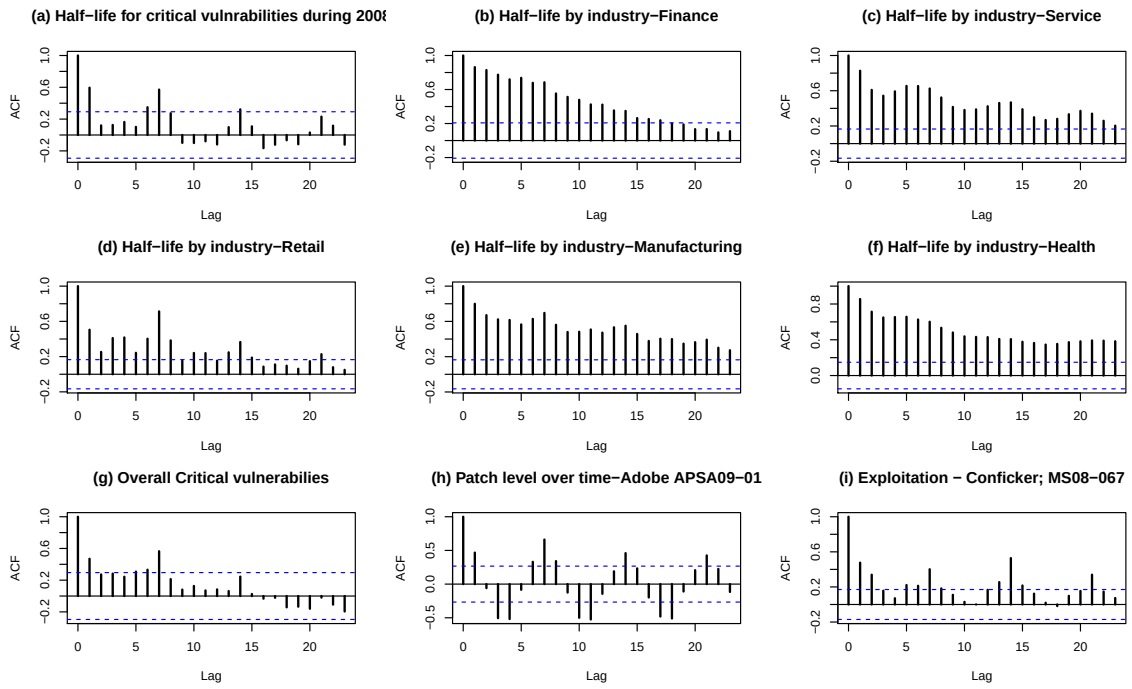


Figure 9.5: Autocorrelation functions corresponding to the plots from Figure 9.4. The dashed lines represent 95% of confidence intervals. Legs are in day.

may be a result of the end-of-year cleanup. However, he did not discuss it in detail. It might be related to year-end report which needs to be completed before the end of the year for many organizations.

Further research is needed to answer why the vulnerability discovery in Microsoft products tends to peak in the mid-year months in addition to the year-end months. One possibility is that Defcon[5], a major computer security related conference, happens in July or August mainly. The potential conference participants might have a higher incentive (Arora and Telang, 2005) to find the vulnerabilities before the conference, to brag about, especially in popular Microsoft products.

Figure 9.6 (a) shows the number of occurrences of Defcon and Black Hat each calendar month; the two best well known conventions where the security vulnerabilities are announced. Meanwhile, the August-November period appears to be associated with release of a larger number of new Microsoft products. Figure 9.6

[5]`https://www.defcon.org/main.html`

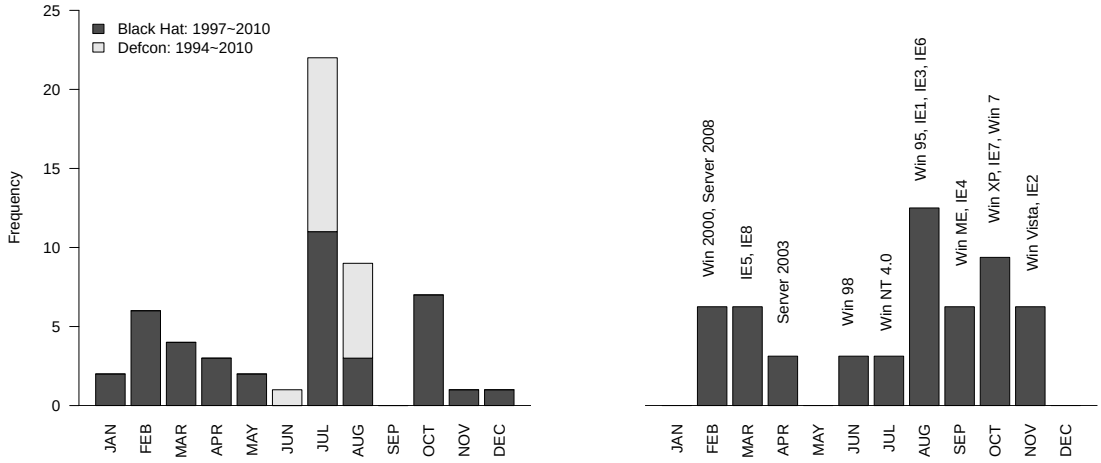
Table 9.12: Weekly seasonal index values

Label (Figure 9.5)	Day1 (Sun)	Day2 (Mon)	Day3 (Tue)	Day4 (Wed)	Day5 (Thu)	Day6 (Fri)	Day7 (Sat)
(a)	1.0494	1.4099	1.3600	0.7210	0.5425	0.9423	0.9745
(b)	0.9825	1.0672	0.7819	1.0322	0.9973	0.8794	1.2592
(c)	1.2194	1.0796	0.6290	0.6312	0.9974	1.2643	1.1788
(d)	1.1784	0.6445	0.6811	0.8007	0.3960	0.7976	2.5014
(e)	1.0571	0.7047	0.8268	1.2247	0.9567	0.9398	1.2899
(f)	1.1573	1.0117	0.8913	0.8334	0.9209	1.1374	1.0477
(g)	1.1848	1.1069	0.7631	0.9798	0.7022	0.7370	1.5258
(h)	0.6758	1.3090	1.2945	1.2569	1.0805	0.7046	0.6783
(i)	0.9559	1.0068	1.2973	1.0203	1.0353	0.9534	0.7307

Table 9.13: χ^2 test for weekly seasonal index values. $\chi^2_{critical}$ = 12.5916 for all.

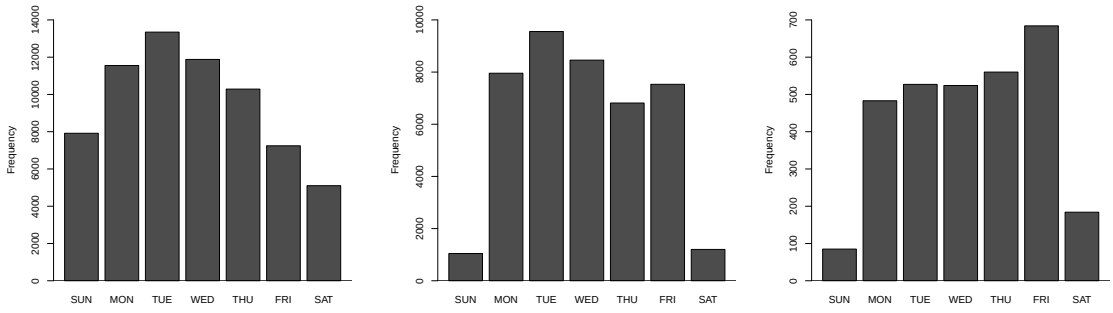
Label (Figure 9.5)	$\chi^2_{statistic}$	P-value
(a)	165.6114	3.83E-33
(b)	49.029	7.36E-09
(c)	165.0925	4.94E-33
(d)	435.1142	7.84E-91
(e)	135.1223	1.07E-26
(f)	44.6814	5.41E-08
(g)	148.7978	1.39E-29
(h)	236.8411	2.65E-48
(i)	119.9789	1.65E-23

(b) shows the products' release months for major versions of Windows OSes and Internet Explorer. The major versions of Windows and Internet Explorer tend to be released during June to November. This may be related to the starting of school semester and Christmas and New Year shopping seasons when many shoppers buy new computers with new operating systems known as IT seasonality. Condon et al. (2008) also observed that occurrences of software security incidents increase during academic calendar, and the most important form of institutional type of seasonality is the school vacations in the summer (Koc and Altinay, 2007). In December, emphasis may shift to identifying and handling vulnerabilities.



(a) Black Hat and Defcon by Month **(b)** MS release by Month

Figure 9.6: Frequency of Black Hat and Defcon by month, and Major Microsoft software system release time by month.



(a) Disclosure date **(b)** Published date **(c)** Report date

Figure 9.7: Frequency of Vulnerability by day of the week in terms of:
(a) Disclosure date: osvdb.org on 2010-10-06, Num. of vulnerabilities: 67325
(b) Published date: nvd.nist.gov on 2010-10-12, Num. of vulnerabilities: 42559
(c) Report date: datalossdb.org on 2010-12-28, Num. of Reports: 3047

The reason for similar seasonality for the Windows operating systems, IIS and Internet Explorer could be due to the fact that IIS and Internet Explorer are distributed for only the Windows platform. Vulnerabilities of the Web servers and browsers may be correlated to the parent operating system platforms.

Figure 9.7 (a) and (b) show the number of vulnerabilities grouped by day of the week in terms of disclosure date from OSVDB and published date from NVD respectively. Disclosure date is the date when vulnerabilities are disclosed to the public whereas published date is the date when vulnerabilities are published on the database, according to OSVDB and NVD respectively. Technically the two dates represent the same meaning. It is observed that both the disclosure date and the published date have the peak values on Tuesday. Generally, values getting increased toward Tuesday and getting decreased after that point. Figure 9.7 (c) represents the number of data loss incidents reported by organizations in terms of day of the week from Open Security Foundation[6]. Although the data does not directly reflect the vulnerability information, it clearly epitomizes the weekday versus weekend phenomenon.

In the meantime, Anbalagan and Vouk (2009) suggest a possible weekly pattern for fixing of ordinary defects. Those reported on Tuesdays tend to be fixed faster. The graph displaying the average correction time from their paper shows the opposite pattern to the first two plots in Figure 9.7: values getting decreased toward Tuesday and getting increased after that point. This might be because developers know that they gain more works on that day in their experience, so in other not to fall behind, more efforts are put on the defects reported on Tuesdays.

The findings, in this chapter, are backed up by the survey results conducted by Tufin Technologies[7] about "Hackers Habits" from 79 hackers attending Defcon 17 conference in 2009. Analysis from the survey reveals that Christmas and New Year

[6] http://datalossdb.org

[7] http://www.tufin.com/news_events_press_releases.php?index=2009-08-25

holiday seasons are popular for hackers aiming for western countries. And hackers spend their time for hacking on the weekdays rather than weekends. Here are some numbers related to our study from the survey:

1. 89% said taking a summer vacation would have little impact on their hacking activities.

2. 81% said they are far more active during the winter holidays with 56% citing Christmas as the best time for hacking, and 25% naming New Year's Eve.

3. 52% said during weekday evenings, they hack other systems and 32% said during the work hours in weekdays whereas only 15% of hackers spends their time for hacking on weekends.

9.7 Summary

Analysis of the vulnerability data using seasonal index and autocorrelation function approaches show that there is indeed a statistically significant annual and weekly periodic pattern in software vulnerability related activities. In the first part of the paper, for the all software systems examined, December has a higher vulnerability discovery rate. A higher incidence during the mid-year periods is also observed in Microsoft products. Also seven-day periodic behavior has been observed in the Laws of Vulnerabilities. Higher activities during the weekdays than weekends have been confirmed. Furthermore, vulnerability activity values corresponding to Tuesday tend to be higher than other days of the week.

One of the main contributions in the study is that it shows various evidences that there are truly short and long term seasonal patterns which have been recognized among security researchers vaguely so far. The results found in the chapter should be used to optimize resource allocations, patch managements, and for the general determination of IT related risks. For example, system administrators should

apply patches at least before the time when seasonal indices are relatively high for both short and long term strategies.

Chapter 10

DEFINING AND ASSESSING QUANTITATIVE SECURITY RISK IN SOFTWARE

Risk and chance frequently come together. "Risk in itself is not bad; risk is essential to progress, and failure is often a key part of learning. But we must learn to balance the possible negative consequences of risk against the potential benefits of its associated opportunity" (Scoy, 1992).

Known vulnerabilities which have been discovered but not patched represents a security risk which can lead to considerable financial damage or loss of reputation. They include vulnerabilities that have either no patches available or for which patches are applied after some delay. Exploitation is even possible before public disclosure of a vulnerability.

This chapter formally defines risk measures and examines possible approaches for assessing risk using actual data with some simulations. We explore the use of CVSS vulnerability metrics which are publically available and are being used for ranking vulnerabilities. Then, a general stochastic risk evaluation approach is proposed which considers the vulnerability lifecycle starting with creation. A conditional risk measure and assessment approach is also presented when only known vulnerabilities are considered. The proposed approach bridges formal risk theory with industrial approaches currently being used, allowing IT risk assessment in an organization, and a comparison of potential alternatives for optimizing remediation.

These actual data driven methods will assist IT managers with software selection and patch application decisions in quantitative manner.

10.1 Motivation

To ensure that the overall security risk stays within acceptable boundaries, managers need to measure risks in their organization. As Lord Calvin stated "If you cannot measure it, you cannot improve it", the security risk to a system cannot be determined without knowing how vulnerable the system is to potential threats (Otwell and Aldridge, 1989). As a result, quantitative methods are needed to ensure that the decisions are not based on only subjective perceptions.

While quantitative risk evaluation is common in some fields such as finance (Alexander, 2008), attempts to quantitatively assess security are relatively new. There has been criticism of the quantitative attempts of risk evaluation (Verendel, 2009) due to the lack of data for validating the methods. Related data has now begun to become just enough to be analyzed. Security vulnerabilities that have been discovered but unpatched for a while represent considerable threat for an organization.

While sometimes risk is informally stated as the possibility of a harm to occur (Pfleeger and Pfleeger, 2003), formally, risk is defined to be a weighted measure depending on the consequence. For a potential adverse event, the risk is stated as (Stoneburner et al., 2001):

$$Risk = Likelihood\ of\ an\ adverse\ event \times Impact\ of\ the\ adverse\ event \tag{10.1}$$

This presumes a specific time period for the evaluated likelihood. For example, a year is the time period for which annual loss expectancy is evaluated. Equation 10.1 evaluates risk due to a single specific cause. When statistically independent multiple causes are considered, the individual risks need to be added to obtain the

overall risk. A risk matrix is often constructed that divides both likelihood and impact values into discrete ranges that can be used to classify applicable causes (Cox, 2008) by the degree of risk they represent.

In the equation above, the *likelihood of an adverse event* is sometimes represented as the product of two probabilities: probability that an exploitable weakness is present, and the probability that such a weakness is exploited (Cox, 2008). The first is an attribute of the target system itself whereas the second probability depends on external factors, such as the motivation of potential attackers. In some cases, the Impact of the adverse event can be split into two factors, the technical impact and the business impact[1].

Risk is often measured conditionally, by assuming that some of the factors are equal to unity and, thus, can be dropped from consideration. For example, sometimes the external factors or the business impact is not considered. If we would replace the impact factor in Equation 10.1 by unity, the conditional risk simply becomes equal to the probability of the adverse event, as considered in the traditional reliability theory. The conditional risk measures are popular because it can alleviate the formidable data collections and analysis requirements.

A stochastic model (Joh and Malaiya, 2010a, 2011a) of the vulnerability lifecycle could be used for calculating the Likelihood of an adverse event in Equation 10.1 whereas impact related metrics from the Common Vulnerability Scoring System (CVSS) (Mell et al., 2007) can be utilized for estimating Impact of the adverse event. While a preliminary examination of some of the vulnerability lifecycle transitions has recently been done by researchers (Frei, 2009; Arbaugh et al., 2000), risk evaluation based on them have been received little attention. The proposed quantitative approach for evaluating the risk associated with software systems will allow comparison of alternative software systems and optimization of risk mitigation strategies.

[1] `http://www.owasp.org/index.php/Top_10_2010-Main`

10.2 Related work on risk measurement

In general, measuring a risk is not an easy task because defining the word "risk" is very tricky[2]. Risk could be a very subjective case by case. Even the Society for Risk Analysis (Kaplan, 1997) took four years, in its early days, to define the word and then gave up saying in the final report that maybe it's better to let each author define it in his own way, only please each should explain clearly what way that is. Consequently, there are many approaches available for measuring the risk.

Singhal and Ou (2009) utilize the attack graph concept for their risk model. Attack graph is a model for how an attacker can combine vulnerabilities to stage an attack. Their attack graph metric quantifies the risk caused by slowing patch release or delaying the patch application. The methodology measures the likelihood that such residual paths may eventually be realized by attackers. Bhatt et al. (2011) proposed a new way to define the CVSS environmental metric. Their goal is to utilize the CVSS base scores to define and compute environmental metrics for components within an enterprise network. Their metrics account for the topological interconnections. Also, the possibility of multistage attacks has been considered. They came up with the equation to measure the impact of an adversary on a network.

In (Ponemon, 2010), the authors quantify the economic impact of a cyber attack by showing the statistic facts related to the cyber crimes with various charts. Manadhata and Wing (2011) proposed to use a software system's attack surface measurement as an indicator of systems' security. A system's attack surface is the subset of its resources that an attacker can use to attack the system. They quantify a system's attack surface measurement to mitigate risk associated with the exploitation.

Vose (2008) defines a risk as a random event that may possibly occur and, if it did occur, would have a negative impact on the goals of an organization. Thus,

[2] http://www.executivebrief.com/blogs/risk-definition-debate-iso31000/

a risk is composed of three elements: the scenario, its probability of occurrence, and the size of its impact if it did occur. As the opposite concept from the risk, the author refers opportunity which is also a random event that may possibly occur but, if it did occur, would have a positive impact. Thus, an opportunity is composed of the same three elements as a risk.

The authors in (OWASP, 2010) itemized the factors in risk into the three standard elements: threat, vulnerability, and impact. The threat is consisted of skill level, motive, opportunity and size for a group of threat agents. The vulnerability is made up of ease of discovery, ease of exploit, awareness, and intrusion detection for the target system. The impact is measured in terms of loss of confidentiality, integrity, availability, and accountability. Leung (2010) puts the time consideration explicitly for the risk assessment, and also propose other variants for risk assessment such as expired-risk, resolution, active problem, etc. Moreover, an opportunity has been utilized as an opposed meaning of the risk, similar to (Vose, 2008).

Dwaikat and Parisi-Presicce (2005) define risk as the probability of violation of a basic security property enforced by the system, where the basic properties include confidentiality, integrity, authenticity and non-repudiation. An interesting thing is that distrust greatly reduces risk in ad-hoc approach while trust reduces risk in systematic risk evaluation. According to Hogganvik and Stolen (2005), risk is consisted of an unwanted incident, a frequency, and a consequence. Unwanted incident may harm more than one asset, but a risk only associated with one asset. This enables them to describe an unwanted incident that has different consequence and/or frequency for a set of assets. Risk is an abstract concept while the unwanted incident is the "real" event.

Aubert et al. (2005) gives definitions of risk based on the specific context: i) risk as an undesirable event ii) risk as a probability function iii) risk as variance iv) risk as expected loss v) endogenous and exogenous risk, and vi) IT outsourcing

risk exposure. Lastly, Weyuker (2001) measured the software risk based on the consequence of failure for a given run test set with a relatively small number of inputs not executed.

10.3 Risk matrix: scales and discretization

In general, a system has multiple weaknesses. The risk of exploitation in each weakness i is given by Equation 10.1. Assuming that the potential exploitation of a weakness is statistically independent of others, the system risk is given by the summation of individual risk values:

$$System\ risk = \sum_i likelihood_i \times impact_i \tag{10.2}$$

A risk matrix provides a visual distribution of potential risks (Engert and Lansdown, 1999; Brashear and Jones, 2008). In many risk evaluation situations, a risk matrix is used, where both impact and likelihood are divided into a set of discrete intervals, and each risk is assigned to likelihood level and an impact level. Impact can be used for the x-axis and likelihood can be represented using then y-axis, allowing a visual representation of the risk distribution. For example, the ENISA Cloud Computing report (ENISA, 2009) defines five impact levels from Very Low to Very High, and five likelihood levels from Very Unlikely to Frequent. Each level is associated with a rating. A risk matrix can bridge quantitative and qualitative analyses. Tables have been compiled that allow on to assign a likelihood and an impact level to a risk, often using qualitative judgment or a rough quantitative estimation.

The scales used for likelihood and impact can be linear, or more often non-linear. In the Homeland Security's RAMCAP (Risk Analysis and Management for Critical Asset Protection) (Brashear and Jones, 2008) approach, a logarithmic scale is used for both. Thus, 0~25 fatalities is assigned a rating "0", while 25~50 is assigned a rating of "1", etc. For the likelihood scale, probabilities between 0.5~1.0 is assigned the highest rating of "5", between 0.25~0.5 is assigned rating "4", etc.

Using a logarithmic scale for both has a distinct advantage. Sometimes the overall rating for a specific risk is found by simply adding its likelihood and impact ratings. Thus, it would be easily explainable if the rating is proportional to the logarithm of the absolute value. Consequently, Equation 10.1 can be rewritten as:

$$\begin{aligned} log(risk_i) &= log(liklihood_i) + log(impact_i) \\ rating\_Risk_i &= rating\_liklihood_i + rating\_impact_i \end{aligned} \tag{10.3}$$

When a normalized value of the likelihood, impact or the risk is used, it will result in a positive or negative constant added to the right hand side. In some cases, higher resolution is desired in the very high as well as very low regions. In such cases a suitable non-linear scale such as using the logit or log-odds function[3] can be used.

The main use of risk matrices is to rank the risks so that higher risks can be identified and mitigated. For determining ranking, the rating can be used instead of the raw value. Cox (2008) has pointed out that the discretization in a risk matrix can potentially result in incorrect ranking, but risk matrices are often used for convenient visualization. It should be noted that the risk ratings are not additive.

We will next examine the CVSS metrics, in the context of risk, that has emerged recently for software security vulnerabilities, and inspect the relationship between {likelihood, impact} in risk and {exploitability, impact} in CVSS vulnerability metric system.

10.4 Possible improvement in CVSS metrics and related work

As discussed in Chapter 2.2.3, Common Vulnerability Scoring System (CVSS) (Mell et al., 2007) has now become an industrial standard for assessing the security vulnerabilities although some alternatives are sometimes used.

[3] `http://www.aetheling.com/docs/Rarity.htm`

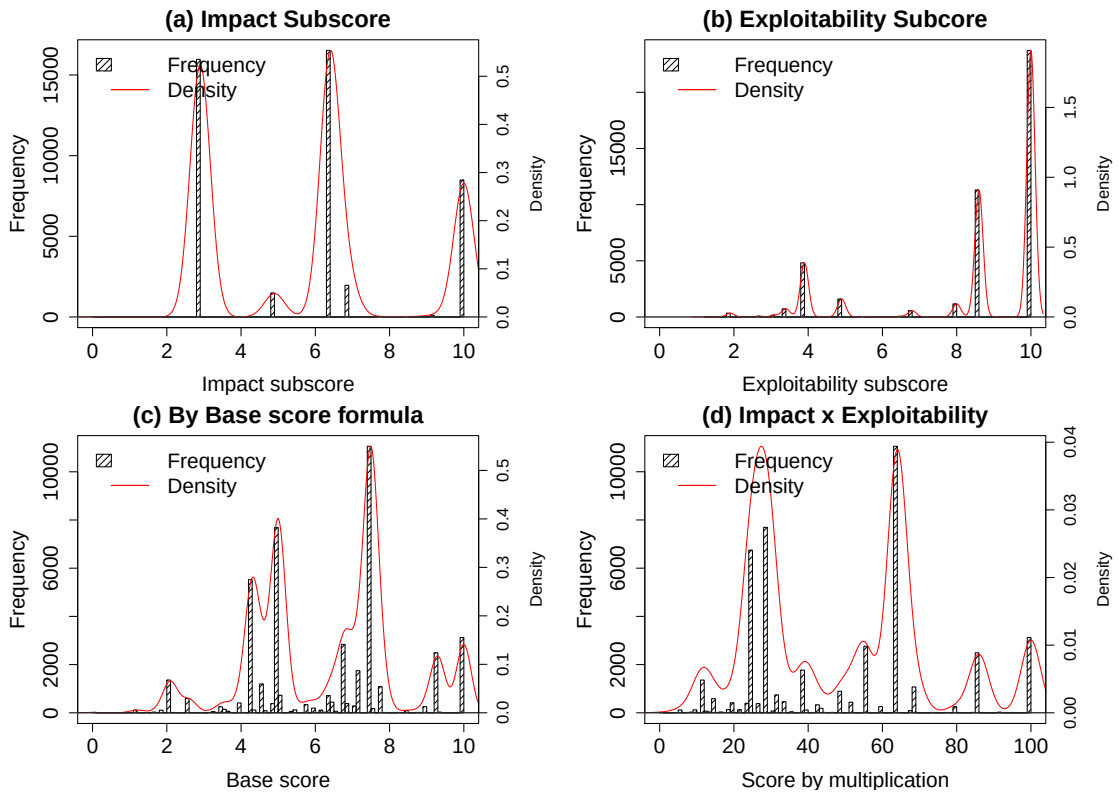


Figure 10.1: Distributions for CVSS base metric scores (100 bins); NVD on JAN 2011 (44615 vulnerabilities)

The CVSS Base metrics include two sub-scores termed exploitability and impact. The Base score formula (Mell et al., 2007), as shown in Equation 10.4, is chosen and adjusted such that a score is a decimal number in the range $[0.0, 10.0]$. The value for $f(Impact)$ is zero when Impact is zero otherwise it has the value of 1.176.

$$\begin{aligned} \textit{Base score} \;=\; & \textit{Round to 1 decimal}\{ \\ & [(0.6 \times \textit{Impact}) + (0.4 \times \textit{Exploitability}) - 1.5] \times f(\textit{Impact})\} \end{aligned} \tag{10.4}$$

The formula for Base score in Equation 10.4 has not been formally derived but has emerged as a result of discussions in a committee of experts. It is primarily intended for ranking of vulnerabilities based on the risk posed by them. It is notable that the Exploitability and Impact sub-scores are added rather than multiplied. One

possible interpretation can be that the two sub-scores effectively use a logarithmic scale, as given in Equation 10.3. Since the Impact and Exploitability sub-scores have a fairly discrete distribution as shown in Figure 10.1 (a) and (b), addition yields the distribution, Figure 10.1 (c), which would not be greatly different if we had used a multiplication. As shown in Figure 10.1 (d), we have indeed verified that using $Impact \times Exploitability$ yields a distribution extremely similar to that in Figure 10.1 (c). We have also found that multiplication generates about twice as many combinations with wider distribution, and it is intuitive since the multiplication is based on the definition of risk given in Equation 10.1.

The Impact sub-score measures how a vulnerability will impact an IT asset in terms of the degree of losses in confidentiality, integrity, and availability which constitute three of the metrics. Below, in our proposed method, we also use these metrics. The Exploitability sub-score uses metrics that attempt to measure how easy it is to exploit the vulnerability. The Temporal metrics measure impact of developments such as release of patches or code for exploitation. The Environmental metrics allow assessment of impact by taking into account the potential loss based on the expectations for the target system. Temporal and Environmental metrics can add additional information to the two sub-scores used for the Base metric for estimating the overall software risk.

A few researchers have started to use the CVSS scores in their proposed methods. Stango et al. (2009) propose a general method for threat analysis in order to prioritize threats and vulnerabilities. They pointed out that it is hard to prioritize threats and vulnerabilities due to the lack of effective metrics, and the complex and sensitive nature of security. The authors solve this issue by combining Bruce Schneier's attack trees (Schneier, 1999) and the CVSS scoring system. In the paper, CVSS is assigned on the attack tree nodes to establish a common and stable view of evaluating security so that quantifying the threat is very clear by walking through the nodes having the highest CVSS scores in each step.

Mkpong-Ruffin et al. (2007) use CVSS scores to calculate the loss expectancy. The average CVSS scores are calculated with the average growth rate for each month for the selected functional groups of vulnerabilities. Then, using the growth rate with the average CVSS score, the predicted impact value is calculated for each functional group.

Wang et al. (2009) proposes an ontology-based approach for analyzing and assessing the security posture in software products. The two key questions in the paper are: 1) given a software product, what is the trustworthiness of it? and 2) among the similar products, which one is the best product in terms of security? Basically, they first create OVM (ontology for vulnerability management) which has all the information about vulnerabilities based on widely accepted standards[4] such as CVSS, CVE, CWE, CPE, and CAPEC. Then, they calculate overall environmental score for given products according to the suggested algorithm. Here is the brief step of the calculation:

1. Retrieve vulnerability information for a product from OVM, and group them based on the combination value of AV, AC, Au from CVSS.
2. Calculate the average of C, I, A for each case for the product from CVSS.
3. Compute the adjusted impact and exploitability for each case for the product.
4. Calculate the environmental score for each case.
5. Calculate the overall environmental score for the product.

Furthermore, Wang et al. (2009) provide the example comparing Internet Explorer 7 and Firefox 3; the calculated overall environmental scores are 9.0286 and 8.3936 respectively, and they conclude that Firefox 3 is more secure than Internet

[4] `http://measurablesecurity.mitre.org/`

Explorer 7 since Firefox 3 has the smaller overall environmental score. However, they ignore other factors such as market share, patch rate, etc.

Houmb and Franqueira (2009) have discussed a model for the quantitative estimation of the security risk level of a system by combining the frequency and impact of a potentially unwanted event and have modeled it as a Markov process. They estimate frequency and impact of vulnerabilities using reorganized original CVSS metrics. And, finally, the two estimated measures are combined to calculate risk levels. Poolsappasit (2010), in his Ph.D. dissertation, refines the work by Houmb and Franqueira (2009). Poolsappasit's empirical estimation keep the CVSS design characteristics and extends the range of possible values in the model (Houmb and Franqueira, 2009) from [0.53,0.83] to [0.12,1.00] which reflects more diversities in score ranges.

10.5 Defining conditional risk measures

Researchers have often investigated measures of risk that seem to be defined very differently. Here, we show that they are conditional measures of risk and can be potentially combined into a single measure of total risk. The likelihood of the exploitation of a vulnerability depends not only on the nature of the vulnerability but also how easy it is to access the vulnerability, the motivation and the capabilities of a potential intruder.

The likelihood L_i, in Equation 10.1, can be expressed in more detail by considering factors such as probability of presence of a vulnerability v_i and how much

exploitation is expected as shown below equations for:

$$\begin{aligned} L_i &= Pr\{v_i\} \times Pr\{Exploitation \mid v_i\} \\ &= Pr\{v_i\} \times Pr\{v_i \text{ is exploitable} \mid v_i\} \times \\ &\quad Pr\{v_i \text{ is accessible} \mid v_i \text{ exploitable}\} \times \\ &\quad Pr\{v_i \text{ externally exploited} \mid v_i \text{ accessible \& exploitable}\} \\ &= L_{Ai} \times L_{Bi} \times L_{Ci} \times L_{Di} \end{aligned}$$

where L_{Bi} represents the inherent exploitability of the vulnerability, L_{Ci} is the probability of accessing the vulnerability, and, L_{Di} represents the external factors. The impact factor, I_i, from Equation 10.1 can be given as:

$$\begin{aligned} I_i &= \sum_j Pr\{\text{Security attribute } j \text{ compromised for } v_i\} \times \\ &\quad \{\text{Expected cost of } j \text{ compromised due to } v_i\} \\ &= \sum_j Pr(attribute_j, v_i) \times C_{ji} \\ &= \sum_j I_{ji} \times C_{ji} \\ &= I_{iA} \times C_{ji} \end{aligned}$$

where the security attribute $j = \{1, 2, 3\}$ represents confidentiality, integrity and availability. I_{iA} is the CVSS Base Impact sub-score whereas C_{ji} is the CVSS Environmental factors of $ConfReq$, $IntegReq$ or $AvailReq$.

The two detailed expressions for likelihood and impact above in terms of constituent factors, allow defining conditional risk measures. Often risk measures used by different authors differ because they are effectively conditional risks which consider only some of the risk components. The components ignored are then effectively equal to one.

As mentioned above, for a weakness i, risk is defined as $L_i \times I_i$. The conditional risk measures R_1, R_2, R_3, R_4 can be defined by setting some of the factors in the above equations to unity:

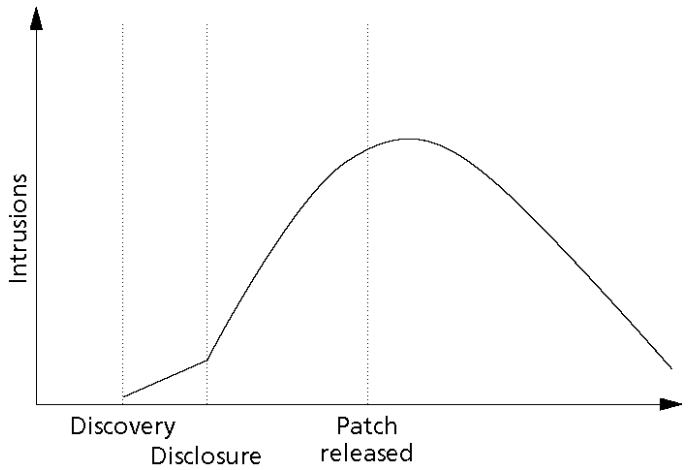


Figure 10.2: Intuitive life cycle of a system-security vulnerability (Arbaugh et al., 2000)

- R_1: by setting L_{Ci}, L_{Di}, C_{ji} as unity. The CVSS Base score is a R_1 type risk measure.
- R_2: by setting L_{Di}, C_{ji} as unity. The CVSS temporal score is a R_2 type risk measure.
- R_3: by setting L_{Di} as unity. The CVSS temporal score is a R_3 type risk measure.
- R_4: is the total risk considering all the factors.

In the next two sections, we examine a risk measure that is more general compared to other perspectives in the sense that we consider the discovery of hitherto unknown vulnerabilities. This would permit us to consider 0-day attacks within our risk framework. Then, in the following section a simplified perspective is presented which considers only the known vulnerabilities.

10.6 Software vulnerability lifecycle

A vulnerability is created as a result of a coding or specification mistake. Figure 10.2 is an intuitive lifecycle of a system-security vulnerability by Arbaugh et al. (2000). In the plot, the number of intrusions increase once a vulnerability is discovered to the right after the patch release time. Especially, the public disclosure

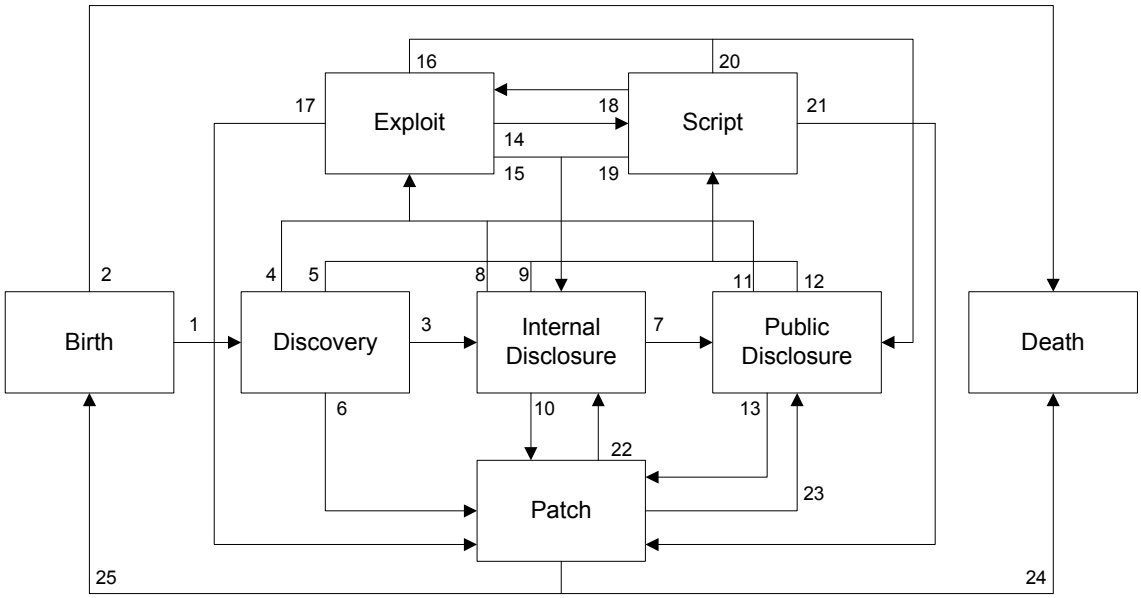


Figure 10.3: Possible vulnerability lifecycle journey

accelerate the intrusion growth rate. The rate is getting decreased after the system administrator applies a patch or workaround.

Meanwhile, Figure 10.3 shows possible vulnerability lifecycle journeys, and Table 10.1 shows the stem events and the next possible hops from Figure 10.3. After the birth, the first event is discovery. A discovery may be followed by any of these: internal disclosure, patch, exploit or script. The discovery rate can be described by vulnerability discovery models (VDM) (Alhazmi and Malaiya, 2008). It has been shown that VDMs are also applicable when the vulnerabilities are partitioned according to severity levels (Woo et al., 2011a). It is expected that some of the CVSS base and temporal metrics impact the probability of a vulnerability exploitation (Mell et al., 2007) although no empirical studies have yet been conducted.

When a white hat researcher discovers a vulnerability, the next transition is likely to be the internal disclosure leading to patch development. After being notified of a discovery by a white hat researcher, software vendors are given a few days, typically 30 or 45 days, for developing patches (Arora et al., 2009). On the other hand, if the disclosure event occurred within a black hat community, the next

Table 10.1: Stem states and next possible hops (line numbers from Figure 10.3)

stem	Next hop
Birth	Discovery(1)
	Death(2)
Discovery	Internal Disclosure(3)
	Exploit(4)
	Script(5)
	Patch(6)
Internal Disclosure	Public Disclosure(7)
	Exploit(8)
	Script(9)
	Patch(10)
Public Disclosure	Exploit(11)
	Script(12)
	Patch(13)
Exploit	Script(14)
	Internal Disclosure(15)
	Pulicity(16)
	Patch(17)
Script	Exploit(18)
	Internal Disclosure(19)
	Public Disclosure(20)
	Patch(21)
Patch	Internal Disclosure(22)
	Public Disclosure(23)
	Death(24)
	Birth(25)
Death	– Absorbing State

possible transition may be an exploitation or a script to automate exploitation. Informally, the term zero day vulnerability generally refers to an unpublished vulnerability that is exploited in the wild (Levy, 2004). Studies show that the time gap between the public disclosure and the exploit is getting narrower (Ayoub, 2007).

Norwegian Honeynet Project[5] found that from the public disclosure to the exploit event takes a median of five days, the distribution is highly asymmetric though.

When a script is available, it enhances the probability of exploitations. It could be disclosed to a small group of people or to the public. Alternatively, the vulnerability could be patched. Usually, public disclosure is the next transition right after the patch availability. When the patch is flawless, applying it causes the death of the vulnerability although sometimes a patch can inject a new fault (Beattie et al., 2002).

Frei (2009) has found that 78% of the examined exploitations occur within a day, and 94% by 30 days from the public disclosure day. In addition, he has analyzed the distribution of discovery, exploit, and patch time with respect to the public disclosure date, using a very large dataset.

10.7 Evaluating lifecycle risk

We first consider evaluation of the risk due to a single vulnerability using stochastic modeling (Joh and Malaiya, 2010a). Figure 10.4 presents a simplified model of the lifecycle for a single vulnerability, described by six distinct states. λs represent transition rates between the states. Initially, the vulnerability starts in State 0 where it has not been found yet. When the discovery leading to State 1 is made by white hats, there is no immediate risk whereas if it is found by a black hat, there is a chance it could be soon exploited. State 2 represents the situation when the vulnerability is disclosed along with the patch release and the patch is applied right away. Hence, State 2 is a safe state and is an absorbing state. In State 5, the vulnerability is disclosed with a patch but the patch has not been applied whereas State 4 represents the situation when the vulnerability is disclosed without a patch. Both State 4 and State 5 expose the system to a potential exploitation which leads

[5] http://www.honeynor.no/research/time2exploit/

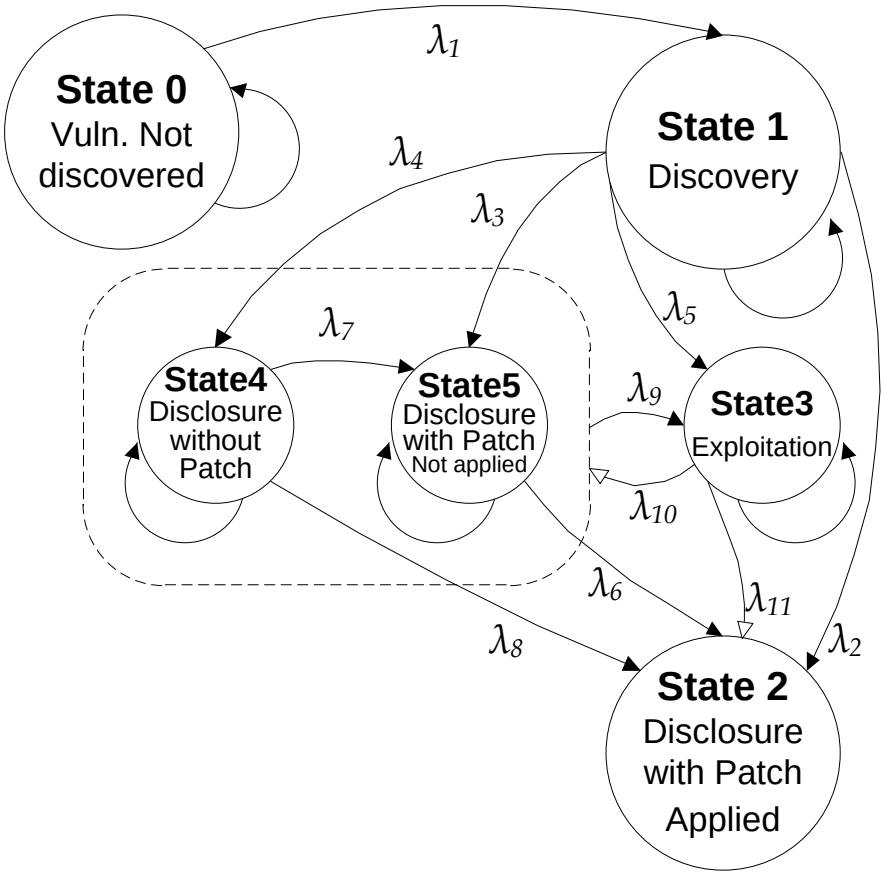


Figure 10.4: Stochastic model for a single vulnerability

to State 3. The two white head arrows (λ_{10} & λ_{11}) are backward transitions representing a recovery which might be considered when multiple exploitations within the period of interest need to be considered. In the discussion below, we assume that State 3 is an absorbing state.

In the figure, for a single vulnerability, the cumulative risk in a specific system at time t can be expressed as probability of the vulnerability being in State 3 at time t multiplied by the consequence of the vulnerability exploitation.

$$Risk_i(t) = Pr\{Vulnerability_i \text{ in State 3 at time } t\} \times exploitation\_impact_i$$

If the system behavior can be approximated using a Markov process, the probability that a system is in a specific state at t could be obtained by using Markov

modeling. Computational methods for semi-Markov (Barbu and Limnios, 2008) and non-Markov (Malaiya et al., 1992) processes exist. However, since they are complex, we illustrate the approach using the Markov assumption. Since the process starts at State 0, the vector giving the initial probabilities is $\alpha = (P_0(0)\ P_1(0)\ P_2(0)\ P_3(0)\ P_4(0)\ P_5(0)) = (1\ 0\ 0\ 0\ 0\ 0)$, where $P_i(t)$ represents the probability that a system is in State i at time t. Let $\mathbb{P}(t)$ be as the state transition matrix for a single vulnerability where t is a discrete point in time. Let the x^{th} element in a row vector of v as v_x , then the probability that a system is in State 3 at time n is $(\alpha \prod_{t=1}^{n} \mathbb{P}(t))_3$. Therefore, according to the Equation 10.1, the risk for a vulnerability i at time t is:

$$Risk_i(t) = (\alpha \prod_{t=1}^{n} \mathbb{P}(t))_3 \times impact_i \tag{10.5}$$

The impact may be estimated from the CVSS scores for Confidentiality Impact (I_C), Integrity Impact (I_I) and Availability Impact (I_A) of the specific vulnerability, along with the weighting factors specific to the system being compromised. It can be expressed as:

$$impact_i = f_c(I_C \times R_C, I_I \times R_I, I_A \times R_A)$$

where f_c is a suitably chosen function. CVSS defines environmental metrics termed Confidentiality Requirement, Integrity Requirement and Availability Requirement that can used for R_C, R_I and R_A. The function f_c may be chosen to be additive or multiplicative. CVSS also defines a somewhat complex measure termed $AdjustedImpact$, although no justification is explicitly provided. A suitable choice of the impact function needs further research.

We now generalize the above discussion to the general case when there are multiple potential vulnerabilities in a software system. If we assume statistical independence of the vulnerabilities (occurrence of an event for one vulnerability is not influenced by the state of other vulnerabilities), the total risk in a software

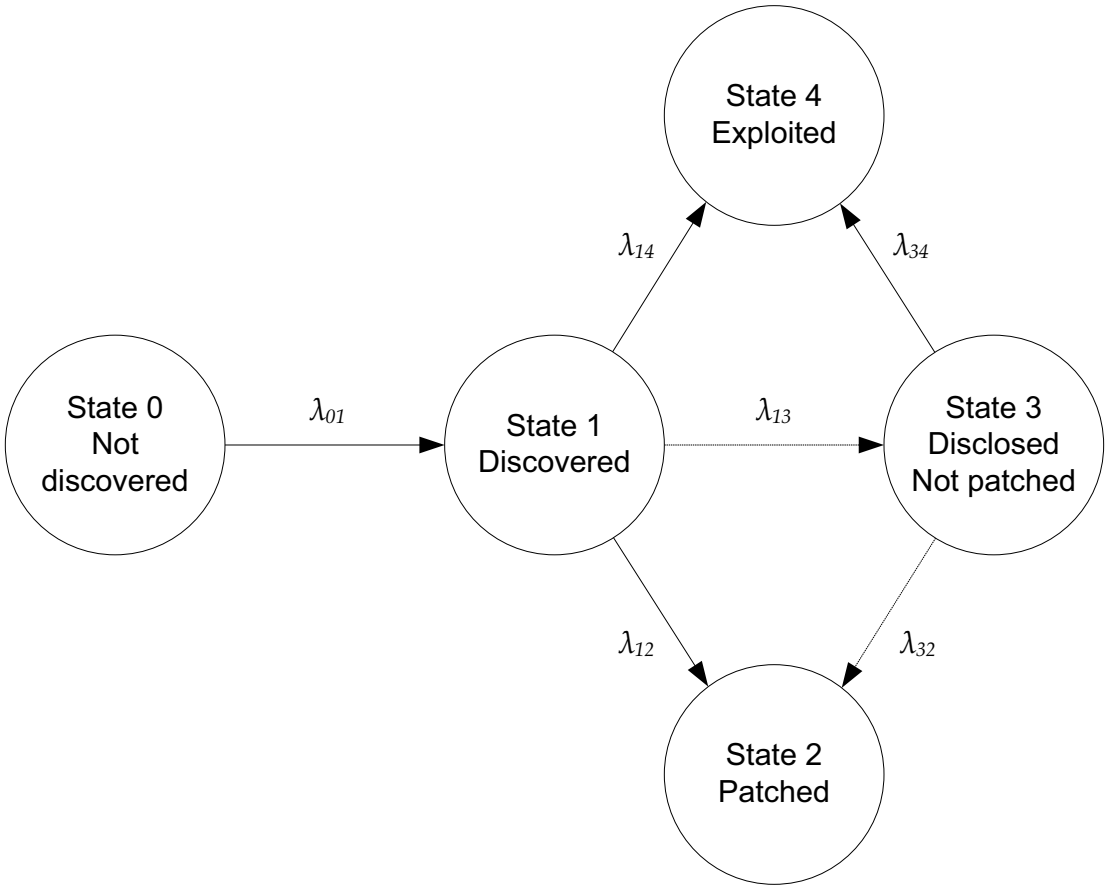


Figure 10.5: Simplified single vulnerability lifecycle - continuous time Markov chain model

system can be obtained by the risk due to each single vulnerability given by Equation 10.6. We can measure risk level as given below for a specific software system.

$$Risk(t) = \sum_{i} (\alpha \prod_{t=1}^{n} \mathbb{P}_i(t))_3 \times impact_i \tag{10.6}$$

The method proposed here could be utilized to measure risks for various units, from single software on a machine to an organization-wide risk due to a specific software. Estimating the organizational risk would involve evaluating the vulnerability risk levels for systems installed in the organizations. The projected organizational risk values can be used for optimization of remediation within the organization.

10.8 Simulate lifecycle risk model

Figure 10.5 shows the simplified single vulnerability lifecycle for a software risk analysis using continuous time Markov process. There are five states, and the λs represent the state transition rates. In this section, the simplified diagram will be analyzed to see the feasibility of the risk analysis using the continuous time Markov process. The probabilities that system is stays in State 0, State 1, State 2, State 3, and State 4 during the lifecycle can be achieved by solving the systems of differential equations:

- $\frac{d}{dt}P_0(t) = -P_0(t)\lambda_{01}$
- $\frac{d}{dt}P_1(t) = \lambda_{01}P_0(t) - P_1(t)\{\lambda_{12} + \lambda_{13} + \lambda_{14}\}$
- $\frac{d}{dt}P_2(t) = \lambda_{12}P_1(t) + \lambda_{32}P_3(t)$
- $\frac{d}{dt}P_3(t) = \lambda_{13}P_1(t) - P_3(t)\{\lambda_{32} + \lambda_{34}\}$
- $\frac{d}{dt}P_4(t) = \lambda_{14}P_1(t) + \lambda_{34}P_3(t)$

In the differential equations, $P_i(t)$ represents the probability that system will stay in State i at time t. It is about finding a solution to nonhomogeneous linear equation. When a first order linear differential equation has the following form:

$$\frac{dy}{dx} + p(x)y = q(x)$$

Then, the general solution[6] can be obtained by:

$$y = \frac{\int u(x)q(x)dx + C}{u(x)},\ where\ u(x) = e^{\int p(x)dx}$$

Figure 10.6 shows the resultant of simulated probabilities with some reasonable transition values: the values are based on the facts of $\lambda_{32} \gg \lambda_{34}$ and $\lambda_{12} \gg \lambda_{13} \gg \lambda_{14}$ in general. The result says that the system will be safely patched with 93% (Figure 10.6 (c)) while about 8% (Figure 10.6 (e)) will be exploited.

[6] `http://www.sosmath.com/diffeq/first/lineareq/lineareq.html`

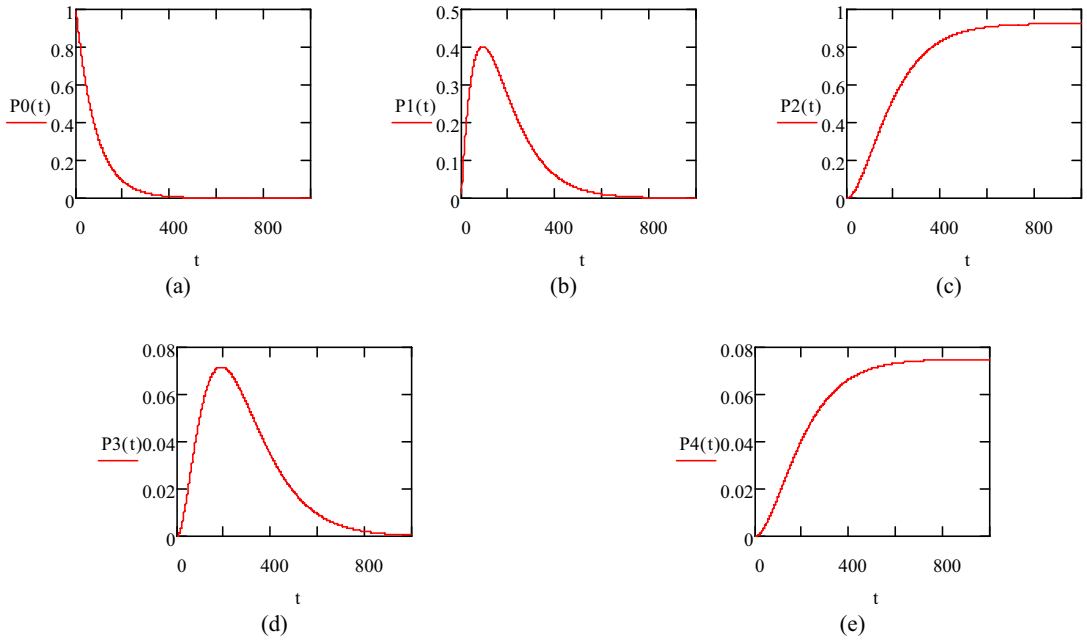


Figure 10.6: Simulated probabilities ($P0(t), P1(t), P2(t), P3(t)$, and $P4(t)$) that system is stays in each state from Figure 10.5; here, transition rates are assumed with some reasonable values: $\lambda_{01} = 1.2 \times 10^{-2}$, $\lambda_{12} = 0.7 \times 10^{-2}$, $\lambda_{13} = 0.25 \times 10^{-2}$, $\lambda_{14} = 0.05 \times 10^{-2}$, $\lambda_{32} = 0.9 \times 10^{-2}$, and $\lambda_{34} = 0.1 \times 10^{-2}$. The values are based on the facts of $\lambda_{32} \gg \lambda_{34}$ and $\lambda_{12} \gg \lambda_{13} \gg \lambda_{14}$ in general

Meanwhile, Equation 10.7 shows how to calculate risk level from Figure 10.5. The equation assumes $\lambda_{14} \approx 0$, and $P_3(t) \approx P_3(t + \Delta)$ where t and Δ represents a risk analysis starting time point and certain period of time respectively. At the final step, in the Equation 10.7, $Pr\{State\ 3\}$, $\lambda_{34} \times \Delta$, and C_i are representing system vulnerability, threat to the system and the value of loss due to the vulnerability i respectively.

$$\begin{aligned} Risk_i(t, t+\Delta) =& Pr\{exploitation\ during\ (t, t+\Delta)\} \times C_i \\ =& \int_{\tau=0}^{\Delta} Pr\{State\ 3\ at\ t\} \times Pr\{S_3 \longrightarrow S_4 | S_3\}\, d\tau \times C_i \\ =& Pr\{State\ 3\} \times \lambda_{34} \times \Delta \times C_i \end{aligned} \tag{10.7}$$

10.9 Measuring risk due to known unpatched vulnerabilities

It might take considerable effort to estimate the transition rates among the states as described in the previous section. A conditional risk measure for a soft-

ware system could be defined in terms of the intervals between the disclosure and patch availability dates that represent the gaps during which the vulnerabilities are exposed.

We can use CVSS metrics to assess the threat posed by a vulnerability. Let us make a preliminary assumption that the relationships between the Likelihood (L) and the Exploitability sub-score (ES), as well as the Impact (I) and the Impact sub-score (IS) for a vulnerability i are linear:

$$\begin{aligned} ES_i &= a_0 + a_1 \times L_i \\ IS_i &= b_0 + b_1 \times I_i \end{aligned}$$

Because the minimum values of ES and IS are zero, a_0 and b_0 are zero. That permits us to define normalized risk values, as can be seen below.

Now, a conditional risk, $Risk\_c_i$, for a vulnerability i can be stated as:

$$Risk\_c_i = L_i \times I_i = \frac{ES_i IS_i}{a_1 b_1}$$

For the aggregated conditional risk is:

$$Risk\_c = \frac{1}{a_1 b_1} \sum_i ES_i IS_i$$

A normalized risk measure $Risk'\_c(t)$ can be defined by dividing by the constant $1/a_1 b_1$, expressed as:

$$Risk'\_c(t) = \sum_i ES_i(t) IS_i(t) \tag{10.8}$$

This serves as an aggregated risk measure for known and exposed vulnerabilities. Its estimation is illustrated below using numerical data.

Figure 10.7 is a conceptual diagram to illustrate the risk gap between vulnerability discoveries and patch releases on top of the simplified three phase vulnerability lifecycle in AML model (Alhazmi and Malaiya, 2008). In the initial learning phase,

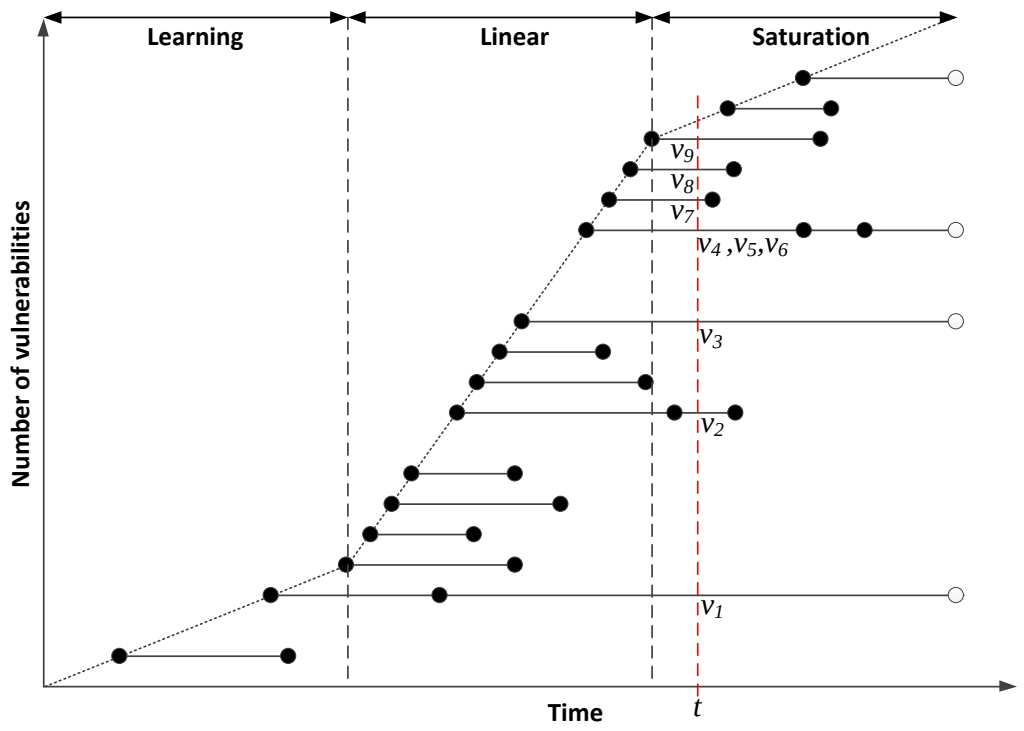


Figure 10.7: Example of the vulnerability discovery and patch in a system with simplified three phase vulnerability lifecycle

the software is gaining market share gradually. In the linear phase, the discovery rate reaches the maximum due to the peak popularity of the software, and finally, in the saturation phase, vulnerability discovery rate slows down.

In the figure, each horizontal line represents the duration for an individual vulnerability from discovery date to patch availability date. When there are multiple dots at the right side, the horizontal line represents multiple vulnerabilities discovered at the same time, but with different patch dates. A white dot is used when a patch is not hitherto available. For example, in Figure 10.7, at time t marked with the vertical red dashed line, there are nine known vulnerabilities with no patches. To calculate the conditional risk level at that time point, each single vulnerability risk level needs to be calculated first and then added as shown in Equation 10.8.

We illustrate the approach using simulated data that has been synthesized using real data. Actual vulnerability disclosure dates from NVD are used but the patch

Table 10.2: Average patch time

Vendor	0-day	30-day	90-day	180-day
Microsoft	61%	75%	88%	94%
Apple	32%	49%	71%	88%

Table 10.3: Simulated patch dates

Patch within		OS 1	OS 2	Browser 1	Browser 2
Simulated # of vuln.	0 day	289	33	54	14
	1-30	66	18	12	7
	31-90	61	23	11	9
	91-180	28	18	5	7
	No patch	30	14	7	7
Total		474	106	89	44

dates are simulated. XP[7] is currently (January 2011) the most popular OS with 55.26% share. Also, Snow Leopard is the most popular among non-Windows OSes. IE 8 and Safari 5 are the most adopted Web browsers for the two OSes.

Considerable effort and time would be needed for gathering the actual patch release dates (Arora et al., 2009), thus, simulated patch dates are used here for the four systems. The patch dates are simulated using the aggregate data (Frei, 2009) representing the fraction of vulnerabilities patched, on average, within 0, 30, 90 and 180 days as shown in Table 10.2. Note that 6% and 12% of the vulnerabilities for Microsoft and Apple respectively are not patched by 180 days. Many of them are patched later. However, because of lack of data, the simulated data treats them as unpatched vulnerabilities which would cause the data to differ from real data.

The simulated datasets are listed in Table 10.3. Note that while OS 1, OS 2, Browser 1 and Browser 2 are based on XP, Snow Leopard, IE 8 and Safari 5 respectively. They are used here only to illustrate the procedure and not for evaluation the risk levels of the actual software.

[7] `http://marketshare.hitslink.com/`

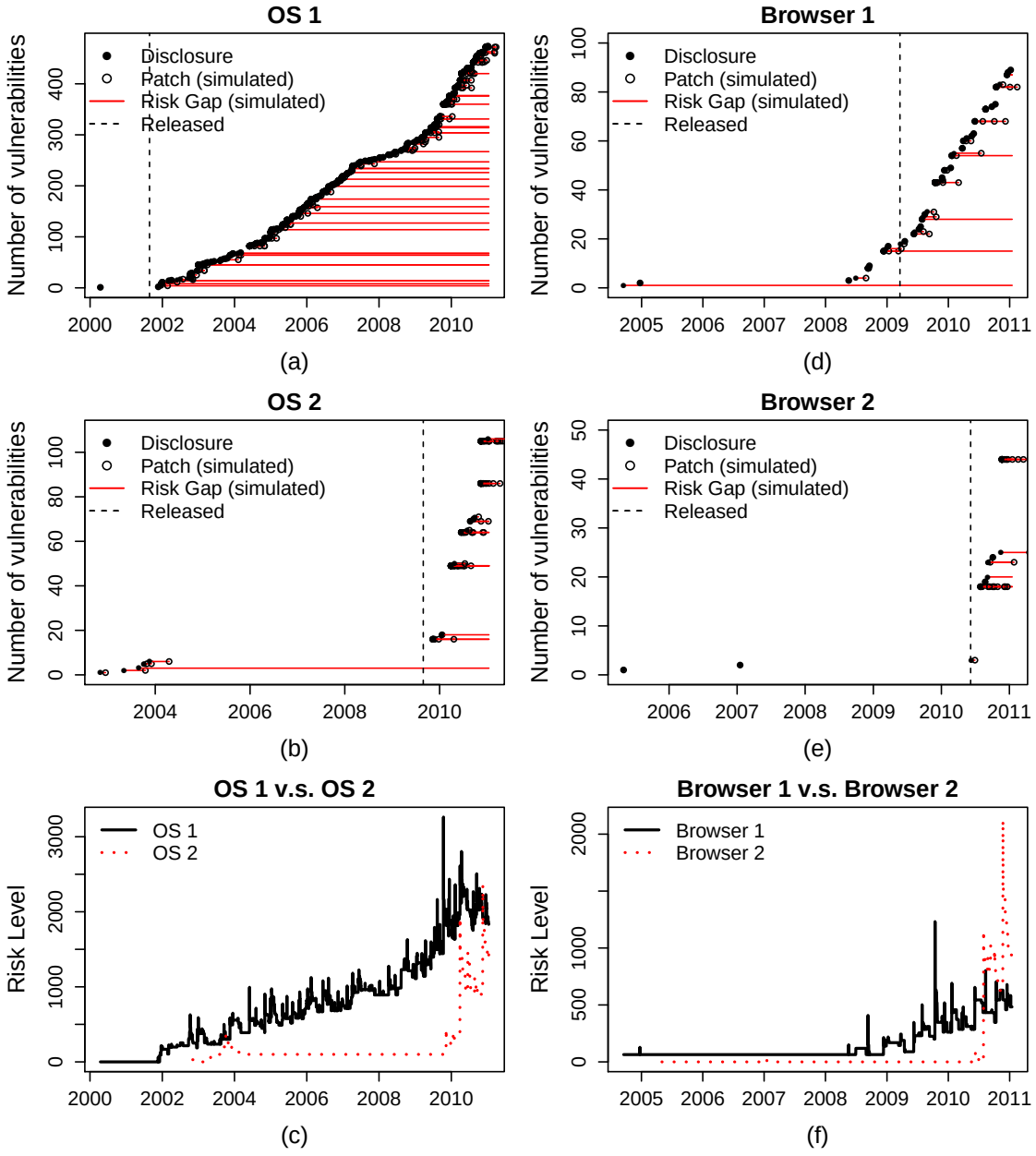


Figure 10.8: Evaluated risk gaps (a, b, c, d) and normalized risk levels (c, f)

Figure 10.8 (a, b, d, e) give the risk gaps for the four datasets. The linear trend arises as the special cases of the logistic process as we discussed in Chapter 8. Figure 10.8 (c, f) give the normalized risk levels calculated daily. As shown in the plots, OS 1 risk level has started to decline while OS 2 risk level is rapidly rising after the release date. For the browsers, Browser 2 risk level raises sharply right after the release due to the two sets of vulnerability clusters with no available immediate patches. The long term rising trend observed might be caused by vulnerabilities we have presumed to be unpatched after 180 days. Since the datasets are simulated, the results only serve as an illustration of the approach and do not represent any actual products.

10.10 Summary

This chapter presents formal measures of security risk that are amenable to evaluation using actual vulnerability data. It also explores the relationship of CVSS metrics and scores with formal expressions of risk.

While a preliminary examination of some of the software lifecycle transitions has recently been done by some researchers (Frei, 2009; Arbaugh et al., 2000), risk evaluation considering the vulnerability lifecycle has so far received very little attention. Here, a formal quantitative approach for software risk evaluation is presented which uses a stochastic model for the vulnerability lifecycle and the CVSS metrics. The model incorporates vulnerability discovery and potential 0-day attacks. The risk values for individual vulnerabilities can be combined to evaluate risk for an entire software system, which can in turn be used for evaluating the risk for an entire organization. A simplified approach for risks due to known but unpatched vulnerabilities is also given.

The proposed approach provides a systematic approach for software risk evaluation. It can be used for comparing the risk levels for alternative systems. The

approach can be incorporated into a methodology for allocating resources optimally by both software developers and end users.

Chapter 11

CONCLUSIONS

This dissertation has focussed on the quantitative analysis of software vulnerabilities. Quantitative analysis has been conducted in many research areas for several reasons such as performance assessment, metric measurement, functional evaluation, or statistical modeling. However, it has only begun recently for security vulnerabilities. We have examined the three topics of i) modeling the vulnerability discovery processes, ii) periodic behavior in vulnerability activities, and iii) risk assessment caused by software vulnerabilities.

As datasets related to software vulnerabilities have become large enough to be analyzed in meaningful manners, researchers have started to examine major attributes of vulnerabilities quantitatively. In this work, quantitative software vulnerability analyses have been presented for Web server and browser vulnerability discovery patterns, assessment of secureness for major web browsers with respect to the Common Vulnerability Scoring System, impact of data skewness on the of S-shaped vulnerability discovery models, extended linear phase vulnerability discovery behavior, periodic behavior of vulnerability activities, and finally, software risk evaluations caused by vulnerabilities in the systems.

11.1 Modeling vulnerability discovery process

We have examined vulnerability discovery process modeling in Chapters 4, 5, 7, and 8.

First, in Chapter 4 and 5, we have examined the applicability for the vulnerability discovery models, both time and effort based models, for the popular Web servers and browsers respectively. All the plots related to model fittings and prediction capabilities reveal that majority of the datasets are well represented by the vulnerability discovery models considered. The models show significant fit results even for the partitioned vulnerability datasets for specific severity levels and defect types.

The vulnerability discovery models, which were originally proposed for operating systems, are found to be applicable to Web servers and browsers also. The VDMs might be utilized for assessing vulnerability discovery rates for the near future. For the software systems undergoing a significant evolution such as a Web browser might not a good candidate for the long term prediction capability since new source codes are continually injected which introduces new vulnerabilities into the systems.

Second, in Chapter 7, the impact of skewed data is examined. New S-shaped vulnerability discovery models, based on Normal, Weibull, Beta, and Gamma distributions, have been introduced and we have compared their performance in model fitting and prediction capabilities along with the existing S-shaped AML model. It has observed that the Gamma distribution based vulnerability discovery model which assumes a right skewed distribution always provides better performance with the positively skewed datasets than other S-shaped models whereas AML and Normal VDM, which are symmetrical distributions based models, performs better than others with negative skewness datasets. This chapter also shows that an excellent goodness of fit does not necessarily mean a superior prediction capability and, thus, a model may not be the best choice based on a historical fit alone.

The results suggest that Gamma vulnerability discovery model should be used to examine the vulnerability discovery process with right skewed datasets. For

other datasets, AML is generally a better choice. However, since the analysis is only conducted for operating systems, Web servers, and Web browsers, it is needed to examine other datasets to confirm the observations. Conducting the comparisons of S-shaped models with the datasets which are categorized by different severity might give further insights. Also, it could be interesting to examine whether there are any meaningful interpretations of each model's parameters, which can lead to estimation of parameters for certain types of datasets. In spite of the results showing the superiority of in Gamma and AML models for the aggregated vulnerability datasets in prediction capabilities, analysis with categorized defect datasets does not show a clear superiority.

Finally, in Chapter 8, we have observed the linear vulnerability discovery trends which is sometimes observed among the popular software systems. The analysis shows some evidences that the linear vulnerability discovery trends are caused mainly because of code sharing between the consecutive versions.

11.2 Periodic behavior in vulnerability activities

A time series analysis that combines the periodic patterns may allow for us to predict the future trend more accurately. In Chapter 9, periodic vulnerability behavior in discovery process as a long term activity has been examined. The presence of unpatched vulnerabilities as well as the exploitation pattern as short term vulnerability activities have been analyzed to evaluate their periodic trends. We have utilized the seasonal index and autocorrelation analysis approaches to establish periodicity and to determine the period.

For the vulnerability discovery process, we carefully inspect the eighteen datasets of software systems (operating systems, Web servers and Web browsers) minded at NVD for annual seasonality in their vulnerability discovery processes. The results show that there are indeed repetitive annual patterns.

For the analysis about the short term activities, it identifies the weekly periodicity. The analysis shows that the seven-day periodicity in presence of unpatched vulnerabilities as well as the exploitation pattern. It is also observed that higher activities during the weekdays than weekends. The observed results should be used to optimize resource allocations and for determination of risk.

11.3 Software risk analysis

In Chapter 10, we propose a framework for evaluating software security risks by using both stochastical and empirical methods. In the stochastic method, Markov process with Common Vulnerability Scoring System (CVSS) has been used, while for the empirical method, we modify the CVSS Base equation to conform with the formal definition of Equation 10.1 and calculate the risk gaps for the all known vulnerabilities in a system.

The challenge for Markov modeling is ensuring that the Markov assumptions are a reasonable approximation and getting the transition rates among the defined states. For empirical risk assessment, the difficulty is in getting the accurate patch dates for the entire set of vulnerabilities in a system. The two methods try to express the commonly accepted risk equation (Equation 10.1).

In Chapter 6, the base scores from Common Vulnerability Scoring System are analyzed for the four popular Web browsers quantitatively, namely, Internet Explorer, Firefox, Opera, and Safari. The base score in the scoring system measures how a vulnerability will directly affect an IT asset as the degree of losses in terms of confidentiality, integrity, and availability. Furthermore, the score captures how the vulnerability is accessed and whether or not extra conditions are required to exploit it.

The observation for the four Web browsers shows that, almost all the time, vulnerabilities are compromised from the remote networks and when no authentication

is required. This suggests that for organizations to enhance their network security, they need to strengthen the firewalls and add authentication processes in their Web services. The result also reveals that the consequences of an exploitation are getting worse.

11.4 Future work

Future work should be continually observing the trends in software vulnerability related activities to sopt any irregular or distinct behaviors which can warrant further investigation.

For the modeling vulnerability discovery process, merely applying new probability distributions on the modeling the vulnerability discovery pattern may not yield interesting resutls. Rather, investigating the software evolution and the linear behavior in mathematical manner would be more worthwhile. Although the linear trend seems quite simple to express, its explanation is complex as we have seen in Chapter 8. The constant growth rate and elements influencing on it such as market share, software age and evolution need to be examined.

The methods in this study do not make use of detailed information on evolution that may be available. Further research is needed to evaluate the impact of evolution of software products that go through many versions by explicitly considering the shared code, vulnerabilities inserted and removed in the process and the impact on resource allocation for testing and patch development.

The vulnerability discovery models could be integrated with risk assessment models in the future. A model recently proposed by Sahinoglu (2006) needs such an assessment for estimating risk and cost of loss. Furthermore, these models can be integrated into the development process to create more secure software systems (Seacord, 2005).

Further research is also needed to evaluate the degree of confidence that can be attained when these methods are used to predict the type of vulnerabilities that are anticipated and their severity levels.

The software systems used in this study represent widely used programs that have continued to evolve. We need to evaluate the applicability of the results to other types of software systems, especially which may have smaller number of users or programs that represent only for a specific version.

Further work, for the periodic behavior in vulnerability activities, can involve development of methods for prediction of future vulnerability discovery trends using Box-Jenkins time series Model (ARIMA) which uses autocorrelation function, periodogram, spectral analysis and partial autocorrelation function analysis. Chen et al. (2010) have considered a periodic factor in their multi-cycle vulnerability discovery model. However, the periodic factor in the model does not directly consider the long or short term seasonality. We need to specifically apply them into conjunction with the longer and shorter terms of trend models to improve the vulnerability discovery predictions and to optimize resource allocation.

Since the software risk assessment topic has just been started, there are many unsolved problems. First, the physical significance of a calculated unit risk level using the proposed methods needs to be established. Also, time dependent transition rates among the defined vulnerability lifecycle states in the stochastic model need to be obtained. When the distribution of the state's sojourn time is not governed by exponential distribution, which is highly likely, semi-Markov process need to be explored. The transition rates could be estimated either using the available literatures or from the examination of real datasets directly. The transition rates need to be inspected according to the types of software system.

While some data has started to become available, further research is needed to develop methods for estimating the applicable transition rates (Frei, 2009; Houmb

and Franqueira, 2009; Penta et al., 2009). In general, the computational approaches need to consider the governing probability distributions for the state sojourn times. Since the impact related scores may reflect a specific non-linear scale, formulation of the impact function also needs further research.

The risk assessment models are preliminary at the moment. Ultimately, we need to make the model detailed enough, so that it is closer to the real world scenario. We might consider the game theory which can include the behaviors of the black hat, and white hat, finders as well as black vulnerability market, white vulnerability market. Finally risk levels of software vulnerabilities and software systems need to be evaluated using the developed method and validated using actual data.

Bibliography

Acer, M. and Jackson, C. (2010). Critical vulnerability in browser security metrics. In *Web 2.0 Security & Privacy*.

Akiyama, F. (1971). An example of software system debugging. In *World Computer Congress*, pages 353–359.

Albaiceta, G., Garcia, E., and Taboada, F. (2007). Comparative study of four sigmoid models of pressure-volume curve in acute lung injury. *BioMedical Engineering OnLine*, 6(1):7.

Alexander, C. (2008). *Market Risk Analysis, Volume I, Quantitative Methods in Finance*. Wiley.

Alhazmi, O., Malaiya, Y., and Ray, I. (2005). Security vulnerabilities in software systems: A quantitative perspective. In Jajodia, S. and Wijesekera, D., editors, *Data and Applications Security XIX*, volume 3654 of *Lecture Notes in Computer Science*, pages 281–294. Springer Berlin / Heidelberg.

Alhazmi, O., Malaiya, Y., and Ray, I. (2007). Measuring, analyzing and predicting security vulnerabilities in software systems. *Computers & Security*, 26(3):219–228.

Alhazmi, O. H. and Malaiya, Y. K. (2005). Quantitative vulnerability assessment of systems software. In *RAMS '05: Proceedings of the RAMS '05. Annual Reliability and Maintainability Symposium, 2005.*, pages 615–620. IEEE Computer Society.

Alhazmi, O. H. and Malaiya, Y. K. (2006a). Measuring and enhancing prediction capabilities of vulnerability discovery models for apache and iis http servers. In *ISSRE '06: Proceedings of the 17th International Symposium on Software Reliability Engineering*, pages 343–352, Washington, DC, USA. IEEE Computer Society.

Alhazmi, O. H. and Malaiya, Y. K. (2006b). Prediction capabilities of vulnerability discovery models. In *RAMS '06: Proceedings of the RAMS '06. Annual Reliability and Maintainability Symposium, 2006.*, pages 86–91, Washington, DC, USA. IEEE Computer Society.

Alhazmi, O. H. and Malaiya, Y. K. (2008). Application of vulnerability discovery models to major operating systems. *Reliability, IEEE Transactions on*, 57(1):14–22.

Alhazmi, O. H., Woo, S.-W., and Malaiya, Y. K. (2006). Security vulnerability categories in major software systems. In *IASTED International conference on communication, network and information security*, pages 138–143.

Álvarez, G. and Petrovic, S. (2003). A new taxonomy of web attacks suitable for efficient encoding. *Computers & Security*, 22(5):435 – 449.

Anbalagan, P. and Vouk, M. (2008). On reliability analysis of open source software - fedora. In *ISSRE '08: Proceedings of the 2008 19th International Symposium on Software Reliability Engineering*, pages 325–326, Washington, DC, USA. IEEE Computer Society.

Anbalagan, P. and Vouk, M. (2009). "days of the week" effect in predicting the time taken to fix defects. In *DEFECTS '09: Proceedings of the 2nd International Workshop on Defects in Large Software Systems*, pages 29–30, New York, NY, USA. ACM.

Anderson, R. (2002). Security in open versus closed systems – the dance of boltzmann, coase and moore. In *Conf. on Open Source Software: Economics, Law and Policy*, pages 1–15.

Arbaugh, W. A., Fithen, W. L., and McHugh, J. (2000). Windows of vulnerability: A case study analysis. *Computer*, 33(12):52–59.

Arora, A., Krishnan, R., Telang, R., and Yang, Y. (2009). An empirical analysis of software vendors' patch release behavior: Impact of vulnerability disclosure. *INFORMATION SYSTEMS RESEARCH*, page isre.1080.0226.

Arora, A. and Telang, R. (2005). Economics of software vulnerability disclosure. *IEEE Security and Privacy*, 3(1):20–25.

Arora, A., Telang, R., and Xu, H. (2008). Optimal policy for software vulnerability disclosure. *Manage. Sci.*, 54(4):642–656.

Aslam, T., Krsul, I., and Spafford, E. (1996). Use of a taxonomy of security faults. Technical report.

Aubert, B. A., Patry, M., and Rivard, S. (2005). A framework for information technology outsourcing risk management. *SIGMIS Database*, 36:9–28.

Aura, T., Bishop, M., and Sniegowski, D. (2000). Analyzing single-server network inhibition. In *Proceedings of the 13th IEEE workshop on Computer Security Foundations*, pages 108–, Washington, DC, USA. IEEE Computer Society.

Ayoub, R. (2007). An analysis of vulnerability discovery and disclosure: Keeping one step ahead of the enemy. *A Frost & Sullivan*, 5.

Barbu, V. S. and Limnios, N. (2008). *Semi-Markov Chains and Hidden Semi-Markov Models Toward Applications: Their Use in Reliability and DNS Analysis*. Springer, New York, NY.

Barua, S. K. and Srinivasan, G. (1987). Investigation of decision criteria for investment in risky assets. *Omega*, 15(3):247–253.

Beattie, S., Arnold, S., Cowan, C., Wagle, P., and Wright, C. (2002). Timing the application of security patches for optimal uptime. In *LISA '02: Proceedings of the 16th USENIX conference on System administration*, pages 233–242, Berkeley, CA, USA. USENIX Association.

Bhatt, S., Horne, W., and Rao, P. (2011). On computing enterprise it risk metrics. Technical report, Hewlett Packard Development Company, L.P.

Bishop, M. (1999). Vulnerability analysis: An extended abstract. In *Proceedings of the Symposium on Recent Advances in Intrusion Detection*, pages 125–136.

Bowerman, B. L. and O'connell, R. T. (1987). *Time Series Forecsting: Unified concepts and computer implementation*. Boston: Duxbury Press, 2nd edition.

Brashear, J. P. and Jones, J. W. (2008). *Wiley Handbook of Science and Technology for Homeland Security*, chapter Risk Analysis and Management for Critical Asset Protection (RAMCAP Plus), pages 1–15. Wiley. DOI: 10.1002/9780470087923.hhs003.

Brocklehurst, S., Chan, P. Y., Littlewood, B., and Snell, J. (1990). Recalibrating software reliability models. *IEEE Trans. Softw. Eng.*, 16(4):458–470.

Browne, H., Arbaugh, W., McHugh, J., and Fithen, W. (2001). A trend analysis of exploitations. In *Security and Privacy, 2001. S P 2001. Proceedings. 2001 IEEE Symposium on*.

Bulmer, M. G. (1965). *Principles of statistics*. MIT Press.

Carrion-Baralt, J. R., Smith, C. J., Rossy-Fullana, E., Lewis-Femandez, R., Davis, K. L., and Silverman, J. M.

Chen, K., Feng, D.-G., Su, P.-R., Nie, C.-J., and Zhang, X.-F. (2010). Multi-cycle vulnerability discovery model for prediction. *Journal of Software*, 21(9):2367 –2375.

Cheswick, W. R. and Bellovin, S. M. (1994). *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley, Reading, MA, first edition.

Compton, B. T. and Withrow, C. (1990). Prediction and control of ada software defects. *Journal of Systems and Software*, 12(3):199 – 207.

Condon, E., He, A., and Cukier, M. (2008). Analysis of computer security incident data using time series models. In *ISSRE '08: Proceedings of the 2008 19th International Symposium on Software Reliability Engineering*, pages 77–86, Washington, DC, USA. IEEE Computer Society.

Cox, L. A. T. (2008). Some limitations of "risk = threat × vulnerability × consequence" for risk analysis of terrorist attacks. *Risk Analysis*, 28(6):1749–1761.

Cukier, M. and Panjwani, S. (2009). Prioritizing vulnerability remediation by determining attacker-targeted vulnerabilities. *IEEE Security and Privacy*, 7:42–48.

Dormann, W. and Rafail, J. (2006). Securing your web browse. Technical report, CERT Coordination Center.

Duebendorfer, T. and Frei, S. (2010). Web browser security update effectiveness. In *Proceedings of the 4th international conference on Critical information infrastructures security*, CRITIS'09, pages 124–137, Berlin, Heidelberg. Springer-Verlag.

Dwaikat, Z. and Parisi-Presicce, F. (2005). Risky trust: risk-based analysis of software systems. In *Proceedings of the 2005 workshop on Software engineering for secure systems-building trustworthy applications*, SESS '05, pages 1–7, New York, NY, USA. ACM.

E. E. Schultz, J., Brown, D. S., and Longstaff, T. A. (1990). Responding to computer security incidents. Technical report, Lawrence Livermore National Laboratory.

Eick, S. G., Graves, T. L., Karr, A. F., Marron, J. S., and Mockus, A. (2001). Does code decay? assessing the evidence from change management data. *IEEE Trans. Softw. Eng.*, 27(1):1–12.

Engert, P. A. and Lansdown, Z. F. (1999). Risk matrix 2.20 user's guide. Technical report, Bedford, Massachusett.

ENISA (2009). Cloud computing - benefits, risks and recommendations for information security. Technical report, European Network and Information Security Agency (ENISA).

Farrell, S. (2010). Why didn't we spot that? [practical security]. *Internet Computing, IEEE*, 14(1):84 –87.

Ford, F., Thompson, H., and Casteran, F. (2005). Role comparison report — web server role. Technical report, Security Innovation.

Frei, S. (2009). *Security Econometrics - The Dynamics of (In)Security*. Eth zurich, dissertation 18197, ETH Zurich. ISBN 1-4392-5409-5, ISBN-13: 9781439254097.

Frei, S., Tellenbach, B., and Plattner, B. (2008). 0-day patch - exposing vendors (in)security performance. In *BlackHat 2008 Europe*.

Goonatilake, R., Herath, A., Herath, S., Herath, S., and Herath, J. (2007). Intrusion detection using the chi-square goodness-of-fit test for information assurance, network, forensics and software security. *J. Comput. Small Coll.*, 23:255–263.

Gopalakrishna, R. and Spafford, E. H. (2005). A trend analysis of vulnerabilities. Technical report, CERIAS, Purdue University.

Gousios, G., Karakoidas, V., Stroggylos, K., Louridas, P., Vlachos, V., and Spinellis, D. (2007). Software quality assessment of open source software. In Papatheodorou, T. S., Christodoulakis, D. N., and Karanikolas, N. N., editors, *Current Trends in Informatics: 11th Panhellenic Conference on Informatics, PCI 2007*, volume A, pages 303–315, Athens. New Technologies Publications.

Grazioli, S. and Jarvenpaa, S. L. (2000). Perils of internet fraud: an empirical investigation of deception and trust with experienced internet consumers. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 30(4):395–410.

Greco, G., Greco, S., and Zumpano, E. (2004). Collaborative filtering supporting web site navigation. *AI Commun.*, 17:155–166.

Grosskurth, A. and Godfrey, M. W. (2007). A case study in architectural analysis: The evolution of the modern web browser. In *EMSE*.

Hallaraker, O. and Vigna, G. (2005). Detecting malicious javascript code in mozilla. In *Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems*, pages 85–94, Washington, DC, USA. IEEE Computer Society.

Hatton, L. (1997). Reexamining the fault density-component size connection. *IEEE Softw.*, 14:89–97.

Heston, S. L. and Sadka, R. (2008). Seasonality in the cross-section of stock returns. *Journal of Financial Economics*, 87(2):418–445.

Hogg, R. V. and Craig, A. T. (1978). *Introduction to Mathematical Statistics*. New York:Macmillan, 4th edition.

Hogganvik, I. and Stolen, K. (2005). Risk analysis terminology for it-systems: does it match intuition? *Empirical Software Engineering, International Symposium on*, 0:13 – 22.

Houmb, S. and Franqueira, V. (2009). Estimating toe risk level using cvss. pages 718–725.

Houmb, S. H., Franqueira, V. N., and Engum, E. A. (2010). Quantifying security risk level from cvss estimates of frequency and impact. *Journal of Systems and Software*, 83(9):1622 – 1634. Software Dependability.

Howard, M. (2009). Improving software security by eliminating the cwe top 25 vulnerabilities. *Security Privacy, IEEE*, 7(3):68 –71.

Huang, S., Tang, H., Zhang, M., and Tian, J. (2010). Text clustering on national vulnerability database. In *Computer Engineering and Applications (ICCEA), 2010 Second International Conference on*, volume 2, pages 295 –299.

IEEE (1990). Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, page 1.

Jackson, C., Barth, A., Bortz, A., Shao, W., and Boneh, D. (2009). Protecting browsers from dns rebinding attacks. *ACM Trans. Web*, 3:2:1–2:26.

Jaquith, A. (2007). *Security Metrics: Replacing Fear, Uncertainty, and Doubt*. Addison-Wesley Professional.

Jegadeesh, N. (1990). Evidence of predictable behavior of security returns. *Journal of Finance*, 45(3):881–98.

Jin, S., Wang, Y., Cui, X., and Yun, X. (2009). A review of classification methods for network vulnerability. In *SMC'09: Proceedings of the 2009 IEEE international conference on Systems, Man and Cybernetics*, pages 1171–1175, Piscataway, NJ, USA. IEEE Press.

Joh, H., Chaichana, S., and Malaiya, Y. K. (2010). Short-term periodicity in security vulnerability activity. *Software Reliability Engineering, International Symposium on*, 0:408–409.

Joh, H. and Malaiya, Y. K. (2008a). Seasonality in vulnerability discovery in major software systems. *Software Reliability Engineering, International Symposium on*, 0:297–298.

Joh, H. and Malaiya, Y. K. (2008b). Vulnerability discovery modeling using weibull distribution. *Software Reliability Engineering, International Symposium on*, 0:299–300.

Joh, H. and Malaiya, Y. K. (2009). Seasonal variation in the vulnerability discovery process. In *ICST '09: International Conference on Software Testing, Verification, and Validation*, pages 191–200, Los Alamitos, CA, USA. IEEE Computer Society.

Joh, H. and Malaiya, Y. K. (2010a). A framework for software security risk evaluation using the vulnerability lifecycle and cvss metrics. *Proc. International Workshop on Risk and Trust in Extended Enterprises (RTEE)*, 0:430–434.

Joh, H. and Malaiya, Y. K. (2010b). Modeling skewness in data with s-shaped vulnerability discovery models. *Software Reliability Engineering, International Symposium on*, 0:406–407.

Joh, H. and Malaiya, Y. K. (2011b). Modeling skewness in vulnerability discovery. *Technical report*.

Joh, H. and Malaiya, Y. K. (2011c). Periodic behavior in software vulnerability activities. *Technical report*.

Joh, H. and Malaiya, Y. K. (July 18~21, 2011a). Defining and assessing quantitative security risk measures using vulnerability lifecycle and cvss metrics. In *The 2011 International Conference on Security and Management (sam)*, pages 10–16.

Jones, J. A. (2007). An introduction to factor analysis of information risk (fair), a framework for understanding, analyzing, and measuring risk. Technical report, Risk Management Insight.

Kaplan, S. (1997). The words of risk analysis. *Risk Analysis*, 17(4):407–417.

Kargl, F., Maier, J., and Weber, M. (2001). Protecting web servers from distributed denial of service attacks. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 514–524, New York, NY, USA. ACM.

Kenney, J. F. and Keeping, E. S. (1962). *Mathematics of Statistics, Pt. 1*. 3rd edition.

Kim, J. (2007). Vulnerability discovery in multiple version software systems: A open source and commercial software system. Master's thesis, Computer Science Department, Colorado State University, Fort Collins, CO.

Kim, J., Malaiya, Y. K., and Ray, I. (2007). Vulnerability discovery in multi-version software systems. In *HASE '07: Proceedings of the 10th IEEE High Assurance Systems Engineering Symposium*, pages 141–148, Washington, DC, USA. IEEE Computer Society.

Koc, E. and Altinay, G. (2007). An analysis of seasonality in monthly per person tourist spending in turkish inbound tourism from a market segmentation perspective. *Tourism Management*, 28(1):227 – 237.

Krsul, I. V. (1998). *Software vulnerability analysis*. PhD thesis, West Lafayette, IN, USA. Major Professor-Spafford, Eugene H.

Kumar, A. (2005). Phishing-a new age weapon. Technical report, Open Web Application Security Project.

Landwehr, C. E., Bull, A. R., McDermott, J. P., and Choi, W. S. (1994). A taxonomy of computer program security flaws. *ACM Comput. Surv.*, 26:211–254.

Leung, A., Yan, Z., and Fong, S. (2004). On designing a flexible e-payment system with fraud detection capability. In *Proceedings of the IEEE International Conference on E-Commerce Technology*, pages 236–243, Washington, DC, USA. IEEE Computer Society.

Leung, H. K. N. (2010). Variants of risk and opportunity. In *In Proc. of the 2010 Asia Pacific Software Engineering Conference*, APSEC '10, pages 395–403.

Levy, E. (2004). Approaching zero. *IEEE Security and Privacy*, 2:65–66.

Lewis, N. (1988). Using binary schemas to model risk analysis. In *Proceedings 1988 Computer Security Risk Management Model Builders Workshop*, pages 35–48.

Li, P. L., Shaw, M., Herbsleb, J., Ray, B., and Santhanam, P. (2004). Empirical evaluation of defect projection models for widely-deployed production software systems. In *SIGSOFT '04/FSE-12: Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering*, pages 263–272, New York, NY, USA. ACM.

Liu, Q. and Zhang, Y. (2010). Vrss: A new system for rating and scoring vulnerabilities. *Computer Communications*, In Press, Corrected Proof:–.

Lyu, M. R. (1995). *Handbook of Software Reliability*. McGraw-Hill.

Machie, A., Roculan, J., Russell, R., and Velzen, M. V. (2001). Nimda worm analysis - executive summary. Technical report, SecurityFocus.

Madan, B. B., Goševa-Popstojanova, K., Vaidyanathan, K., and Trivedi, K. S. (2004). A method for modeling and quantifying the security attributes of intrusion tolerant systems. *Perform. Eval.*, 56:167–186.

Maes, J., Van Damme, S., Meire, P., and Ollevier, F. (2004). Statistical modeling of seasonal and environmental influences on the population dynamics of an estuarine fish community. *Marine Biology*, 145:1033–1042.

Malaiya, Y. K., Karunanithi, N., and Verma, P. (1992). Predictability of software reliability models. *Reliability, IEEE Transactions on*, 41(4):539–546.

Manadhata, P. K. and Wing, J. M. (2011). An attack surface metric. *IEEE Trans. Softw. Eng.*, 37:371–386.

Massacci, F. and Nguyen, V. H. (2010). Which is the right source for vulnerability studies? an empirical analysis on mozilla firefox. Technical report, University of Trento, Italy.

Matwyshyn, A. M., Cui, A., Keromytis, A. D., and Stolfo, S. J. (2010). Ethics in security vulnerability research. *IEEE Security and Privacy*, 8:67–72.

Mayerfeld, H. (1988). Definition and identification of assets as the basis for risk management. In *Proceedings 1988 Computer Security Risk Management Model Builders Workshop*, pages 21–34.

McGraw, G. (2003). From the ground up: the dimacs software security workshop. *Security Privacy, IEEE*, 1(2):59 – 66.

McQueen, M., McQueen, T., Boyer, W., and Chaffin, M. (2009). Empirical estimates and observations of 0day vulnerabilities. In *HICSS '09: Proceedings of the 42nd Hawaii International Conference on System Sciences*, pages 1–12, Washington, DC, USA. IEEE Computer Society.

Mell, P., Scarfone, K., and Romanosky, S. (2007). Cvss: A complete guide to the common vulnerability scoring system version 2.0. Technical report, Forum of Incident Response and Security Teams (FIRST).

Meneely, A. and Williams, L. (2009). Secure open source collaboration: an empirical study of linus' law. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 453–462, New York, NY, USA. ACM.

Mkpong-Ruffin, I., Umphress, D., Hamilton, J., and Gilbert, J. (2007). Quantitative software security risk assessment model. In *QoP '07: Proceedings of the 2007 ACM workshop on Quality of protection*, pages 31–33, New York, NY, USA. ACM.

Moitra, S. D. (1990). Skewness and the beta distribution. *The Journal of the Operational Research Society*, 41(10):953–961.

Moore, D., Shannon, C., and claffy, k. (2002). Code-red: a case study on the spread and victims of an internet worm. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, IMW '02, pages 273–284, New York, NY, USA. ACM.

Moshchuk, A., Bragin, T., Gribble, S. D., and Levy, H. M.

Musa, J. (1999). *Software Reliability Engineering*. McGraw-Hill.

Musa, J. D. and Okumoto, K. (1984). A logarithmic poisson execution time model for software reliability measurement. In *ICSE '84: Proceedings of the 7th international conference on Software engineering*, pages 230–238, Piscataway, NJ, USA. IEEE Press.

Neuhaus, S. and Zimmermann, T. (2009). The beauty and the beast: Vulnerabilities in red hat's packages. In *Proceedings of the 2009 USENIX Annual Technical Conference*.

Neuhaus, S. and Zimmermann, T. (2010). Security trend analysis with cve topic models. Technical report, University of Trento, Italy.

NIST (2006). Glossary of key information security terms. *NIST IR 7298*.

Okamura, H., Tokuzane, M., and Dohi, T. (2009). Optimal security patch release timing under non-homogeneous vulnerability-discovery processes. In *Proceedings of the 20th IEEE international conference on software reliability engineering*, ISSRE'09, pages 120–128, Piscataway, NJ, USA. IEEE Press.

Ott, R. L. and Longnecker, M. T. (2000). *An Introduction to Statistical Methods and Data Analysis*. Duxbury press, 5th edition.

Otwell, K. and Aldridge, B. (1989). The role of vulnerability in risk management. In *Computer Security Applications Conference, 1989., Fifth Annual*, pages 32–38.

OWASP (2010). The ten most critical web application security risks. Technical report, The Open Web Application Security Project.

Ozment, A. (2007a). Improving vulnerability discovery models. In *QoP '07: Proceedings of the 2007 ACM workshop on Quality of protection*, pages 6–11, New York, NY, USA. ACM.

Ozment, A. (2007b). *Vulnerability Discovery and Software Security*. PhD thesis, University of Cambridge Computer Laboratory, Cambridge, UK.

Ozment, A. and Schechter, S. E. (2006). Milk or wine: does software security improve with age? In *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium*, Berkeley, CA, USA. USENIX Association.

Penta, M. D., Cerulo, L., and Aversano, L. (2009). The life and death of statically detected vulnerabilities: An empirical study. *Information and Software Technology*, 51(10):1469–1484. Source Code Analysis and Manipulation, SCAM 2008.

Pfleeger, C. P. and Pfleeger, S. L. (2003). *Security in Computing*. Prentice Hall PTR, 3rd edition.

Ponemon, I. (2010). First annual cost of cyber crime study with ponemon institute. Technical report, sponsored by ArcSight.

Poolsappasit, N. (2010). *Towards an efficient vulnerability analysis methodology for better security risk management*. PhD thesis, Colorado State University.

Qualys, I. (July 28, 2009). The laws of vulnerabilities 2.0. In *Black Hat 2009, Presented by Wolfgang Kandek (CTO)*.

Radianti, J., Rich, E., and Gonzalez, J. (2009). Vulnerability black markets: Empirical evidence and scenario simulation. In *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on*, pages 1 –10.

Rajeev Gopalakrishna, E. H. S. and Vitek, J. (2005). Vulnerability likelihood: A probabilistic approach to software assurance. Technical report, CERIAS, Purdue University.

Rescorla, E. (2003). Security holes... who cares? In *SSYM'03: Proceedings of the 12th conference on USENIX Security Symposium*, pages 75–90, Berkeley, CA, USA. USENIX Association.

Rescorla, E. (2005). Is finding security holes a good idea? *IEEE Security and Privacy*, 3:14–19.

Rios, M., Garcia, J. M., Sanchez, J. A., and Perez, D. (2000). A statistical analysis of the seasonality in pulmonary tuberculosis. *Eur J Epidemiol*, 16(5):483–8.

Rosenberg, J. (1997). Some misconceptions about lines of code. In *Proceedings of the 4th International Symposium on Software Metrics*, pages 137–, Washington, DC, USA. IEEE Computer Society.

Sahinoglu, M. (2006). Quantitative risk assessment for dependent vulnerabilities. In *RAMS '06: Proceedings of the RAMS '06. Annual Reliability and Maintainability Symposium, 2006.*, pages 82–85, Washington, DC, USA. IEEE Computer Society.

Salehian, A. (2003). Arima time series modeling for forecasting thermal rating of transmission lines. In *Transmission and Distribution Conference and Exposition, 2003 IEEE PES*, volume 3, pages 875 – 879 vol.3.

Scarfone, K. and Mell, P. (2009). An analysis of cvss version 2 vulnerability scoring. In *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on*, pages 516 –525.

Schneier, B. (1999). Attack trees. *Dr. Dobb's Journal of Software Tools*, 24(12):21–29.

Schryen, G. (2009). Security of open source and closed source software: An empirical comparison of published vulnerabilities. In *Proceedings of the 15th Americas Conference on Information Systems*.

Scoy, R. L. V. (1992). Software development risk: Opportunity, not problem (cmu/sei-92-tr-030). Technical report, Software Engineering Institute at Carnegie Mellon University, Pittsburgh, Pennsylvania.

Seacord, R. C. (2005). *Secure Coding in C and C++ (SEI Series in Software Engineering)*. Addison-Wesley Professional.

Seacord, R. C. and Householder, A. D. (2005). A structured approach to classifying security vulnerabilities. Technical report, CMU/SEI-2005-TN-00, Carnegie Mellon University.

Singhal, A. and Ou, X. (2009). Techniques for enterprise network security metrics. In *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*, CSIIRW '09, pages 25:1–25:4, New York, NY, USA. ACM.

Snow, D. (1988). A general model for the risk management of adp systems. In *Proceedings 1988 Computer Security Risk Management Model Builders Workshop*, pages 145–162.

Stango, A., Prasad, N. R., and Kyriazanos, D. M. (2009). A threat analysis methodology for security evaluation and enhancement planning. In *SECURWARE '09: Proceedings of the 2009 Third International Conference on Emerging Security Information, Systems and Technologies*, pages 262–267, Washington, DC, USA. IEEE Computer Society.

Stone, R. (1980). Sigmoids. *Bulletin in applied statistics*, 7(1):59–119.

Stoneburner, G., Goguen, A., and Feringa, A. (2001). Risk management guide for information technology systems. Technical report, National Institute of Standards and Technology (NIST). Special Publication 800-30.

Tran, N. and Reed, D. (2004). Automatic arima time series modeling for adaptive i/o prefetching. *Parallel and Distributed Systems, IEEE Transactions on*, 15(4):362 – 377.

Vache, G. (2009). Vulnerability analysis for a quantitative security evaluation. In *ESEM '09: Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 526–534, Washington, DC, USA. IEEE Computer Society.

Venter, H., Eloff, J., and Li, Y. (2008). Standardising vulnerability categories. *Computers & Security*, 27(3-4):71 – 83.

Verendel, V. (2009). Quantified security is a weak hypothesis: a critical survey of results and assumptions. In *Proceedings of the 2009 workshop on New security paradigms workshop*, NSPW '09, pages 37–50, New York, NY, USA. ACM.

Voit, E. O. and Schwacke, L. H. (2000). Random number generation from right-skewed, symmetric, and leftskewed distributions. *Risk Analysis*, 20(1):59–72.

Vose, D. (2008). *Risk Analysis: A quantitative guide*. John Wiley & Sons, Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, England, 3rd edition.

Wang, J. A. and Guo, M. (2009). Ovm: an ontology for vulnerability management. In *CSIIRW '09: Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research*, pages 1–4, New York, NY, USA. ACM.

Wang, J. A., Guo, M., Wang, H., Xia, M., and Zhou, L. (2009). Environmental metrics for software security based on a vulnerability ontology. In *Secure Software Integration and Reliability Improvement, 2009. SSIRI 2009. Third IEEE International Conference on*, pages 159 –168.

Wang, J. A., Wang, H., Guo, M., Zhou, L., and Camargo, J. (2010). Ranking attacks based on vulnerability analysis. *Hawaii International Conference on System Sciences*, 0:1–10.

Weyuker, E. J. (2001). Difficulties measuring software risk in an industrial environment. In *The International Conference on Dependable Systems and Networks*, DSN '01, pages 15–24.

White, D. S. D. (2006). Limiting vulnerability exposure through effective patch management: threat mitigation through vulnerability remediation. Master's thesis, Department of Computer Science, Rhodes University.

Woo, S.-W., Alhazmi, O. H., and Malaiya, Y. K. (2006a). An analysis of the vulnerability discovery process in web browsers. In *Proc. 10th IASTED Int. Conf. on Software Engineering and Applications*, pages 172–177.

Woo, S.-W., Alhazmi, O. H., and Malaiya, Y. K. (2006b). Assessing vulnerabilities in apache and iis http servers. *Dependable, Autonomic and Secure Computing, IEEE International Symposium on*, 0:103–110.

Woo, S.-W., Joh, H., Alhazmi, O. H., and Malaiya, Y. K. (2011a). Modeling vulnerability discovery process in apache and iis http servers. *Computers & Security*, 30(1):50 – 62.

Woo, S.-W., Joh, H., and Malaiya, Y. K. (2011b). Modeling vulnerability discovery process in web browsers using calendar time and usage. *Technical report*.

Younis, A. A., Joh, H., and Malaiya, Y. K. (July 18~21, 2011). Modeling learningless vulnerability discovery using a folded distribution. In *The 2011 International Conference on Security and Management (sam)*.

Zhou, Y. and Davis, J. (2005). Open source software reliability model: an empirical approach. *SIGSOFT Softw. Eng. Notes*, 30(4):1–6.

Zimmermann, T., Nagappan, N., and Williams, L. (2010). Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista. In *Proceedings of the 2010 Third International Conference on Software Testing, Verification and Validation*, ICST '10, pages 421–428, Washington, DC, USA. IEEE Computer Society.