DISSERTATION

QUANTIFYING THE SECURITY RISK OF DISCOVERING AND EXPLOITING SOFTWARE VULNERABILITIES

Submitted by

Awad A Younis Mussa

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2016

Doctoral Committee:

Advisor: Yashwant Malaiya

Indrajit Ray
Charles W. Anderson
Leo Vijayasarathy

ABSTRACT

QUANTIFYING THE SECURITY RISK OF DISCOVERING AND EXPLOITING SOFTWARE VULNERABILITIES

Most of the attacks on computer systems and networks are enabled by vulnerabilities in a software. Assessing the security risk associated with those vulnerabilities is important. Risk models such as the Common Vulnerability Scoring System (CVSS), Open Web Application Security Project (OWASP) and Common Weakness Scoring System (CWSS) have been used to qualitatively assess the security risk presented by a vulnerability. CVSS metrics are the de facto standard and its metrics need to be independently evaluated.

In this dissertation, we propose using a quantitative approach that uses an actual data, mathematical and statistical modeling, data analysis, and measurement. We have introduced a novel vulnerability discovery model, Folded model, that estimates the risk of vulnerability discovery based on the number of residual vulnerabilities in a given software. In addition to estimating the risk of vulnerabilities discovery of a whole system, this dissertation has furthermore introduced a novel metrics termed time to vulnerability discovery to assess the risk of an individual vulnerability discovery.

We also have proposed a novel vulnerability exploitability risk measure termed Structural Severity. It is based on software properties, namely attack entry points, vulnerability location, the presence of the dangerous system calls, and reachability analysis. In addition to measurement, this dissertation has also proposed predicting vulnerability exploitability risk using internal software metrics.

We have also proposed two approaches for evaluating CVSS Base metrics. Using the availability of exploits, we first have evaluated the performance of the CVSS Exploitability factor and have compared its performance to Microsoft (MS) rating system. The results showed that exploitability metrics of CVSS and MS have a high false positive rate. This finding has motivated us to conduct

further investigation. To that end, we have introduced vulnerability reward programs (VRPs) as a novel ground truth to evaluate the CVSS Base scores. The results show that the notable lack of exploits for high severity vulnerabilities may be the result of prioritized fixing of vulnerabilities.

ACKNOWLEDGEMENTS

First and foremost, I would like to express the sincere appreciation to my advisor, Dr. Yashwant K. Malaiya, who has constantly supported me with his insight and 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. Charles W. Anderson, Dr. Chihoon Lee, and Dr. Leo Vijayasarathy, for agreeing to be on the board, reviewing my works, and their suggestions, criticism and support.

First and foremost, I would like to thank my Mam, may God bless her soul in peace, for teaching me nothing comes without struggles and sacrifices. I would also like to thank my wife for being patient and for sacrificing everything just to see me succeeding. I would also like to thank my brother Faisel for his continuous emotional and physical support during my whole educational journey. I also would like to thank my kids Laith and Hind for being such an inspiration for me to stay focused on what I do in a daily basis so I can get some time for them. I also would like to thank my brothers Gaith, Saad, Madina, Hemmaly, Sayed, Agiella, Fathie, Yakoob, Moosbah, and Baast for being supportive and carrying about my life and education. I also would like to thank all my brothers and sisters for their encouragements and support. Last but not least I would like to thank my friends and especially Mohammed Alshaka, Ali Alhomoodie, Mahdi Omar, Gadafie Khalaifa, and Saad Ahmedy for their warm feelings and engorgement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

This chapter introduces the problem description, research motivations, and dissertation outlines.

1.1 Problem Description and Research Motivation

Most of the attacks on computer systems and networks are enabled by vulnerabilities in a software. A vulnerability is a software defect which might be exploited by malicious users causing loss or harm [2]. The number of reported discovered software vulnerabilities, as shown in Figure 1.1, shows that software is vulnerable in spite of the recent advances in practice to intensify software security such as vulnerability avoidance, vulnerability removal, and intrusion blocking.

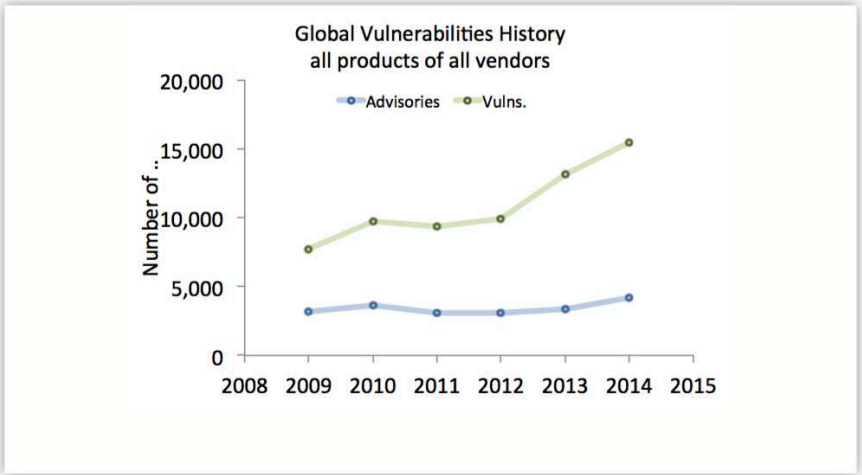


Figure 1.1: Number of reported vulnerabilities (year 2008 2015, [1])

The number of security Studies have also shown that the time gap between the vulnerability public disclosure and the release of an automated exploit is getting smaller [3] as shown in Figure 1.2. Thus, assessing the risk associated with those vulnerabilities is critical. Risk models such as the Common Vulnerability Scoring System (CVSS) [4], Open Web Application Security Project (OWASP) [5] and Common Weakness Scoring System (CWSS) [6] have been used to qualitatively

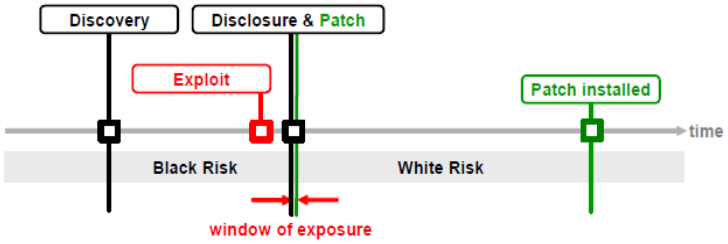


Figure 1.2: Time gap between public disclosure and the release of an exploit

assess the security risk presented by a vulnerability. CVSS metrics have become a de facto standard that is commonly used to assess the severity of a vulnerability. Thus, its metrics need to be independently evaluated.

The research on quantitative assessment of software vulnerability risk seeks to forecast how likely a vulnerability is to be discovered and then exploited based on actual data, using mathematical and statistical modeling and measurement. Quantitative assessment of software vulnerabilities, however, is in its infancy status and it is considered challenging because the full nature of what should constitute an appropriate measure of software security is yet to be established. The main research questions in this dissertation are:

- RQ1: What are the factors that can be used to assess:
  - Likelihood of discovery
  - Likelihood of exploitation
  - Impact of exploitation
- RQ2: How can we objectively derive them?
- RQ3: What type of assessment method should be used:
  - Analysis
  - Measurement
  - Modeling

- RQ4: How can we evaluate CVSS Base metrics?

Besides, quantitative security is challenged by the availability of data. Recently data and standards have been introduced by security community. Figure 1.3 shows the standards and data sources used in this dissertation. More details about them can be found in chapter 2. It should be noted that the Bugzilla has been used to map vulnerabilities to their location in the source code.

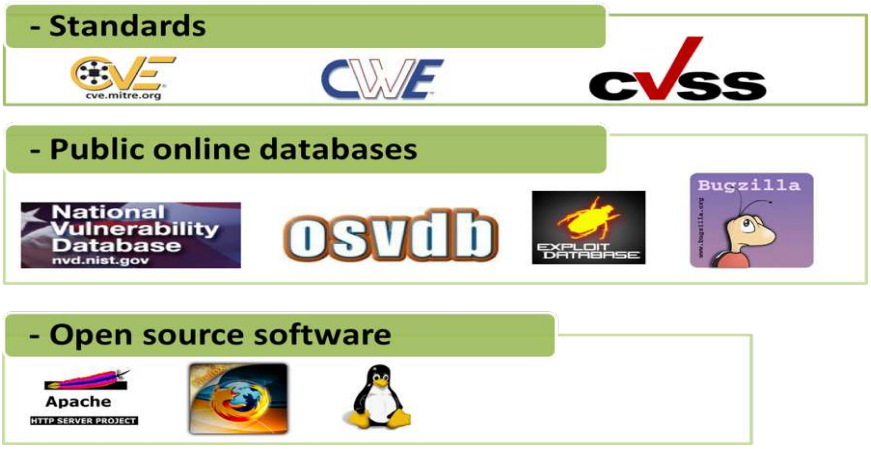


Figure 1.3: Software Vulnerabilities Data

1.2 Research Objectives and Contribution

The main objective of this research is to propose a framework that can address the challenging questions discussed in the problem description. Here, we mainly focus on reducing subjectivity and minimize human involvement. To meet these objectives, Figure 1.4 shows the main three layers of the proposed framework. It should be noted that in addition to building a model, measuring and analyzing vulnerability discovery and exploitation risk have been conducted as well but not explicitly mentioned in the Figure 1.4.

To address the risk of vulnerabilities discovery risk, this dissertation proposes a novel vulnerability discovery model, Folded model. It is a time based model that uses known reported vulnerabilities data to estimate the total number of the remaining vulnerabilities in a software. This number is then used to quantify the residual vulnerabilities by subtracting the number of known

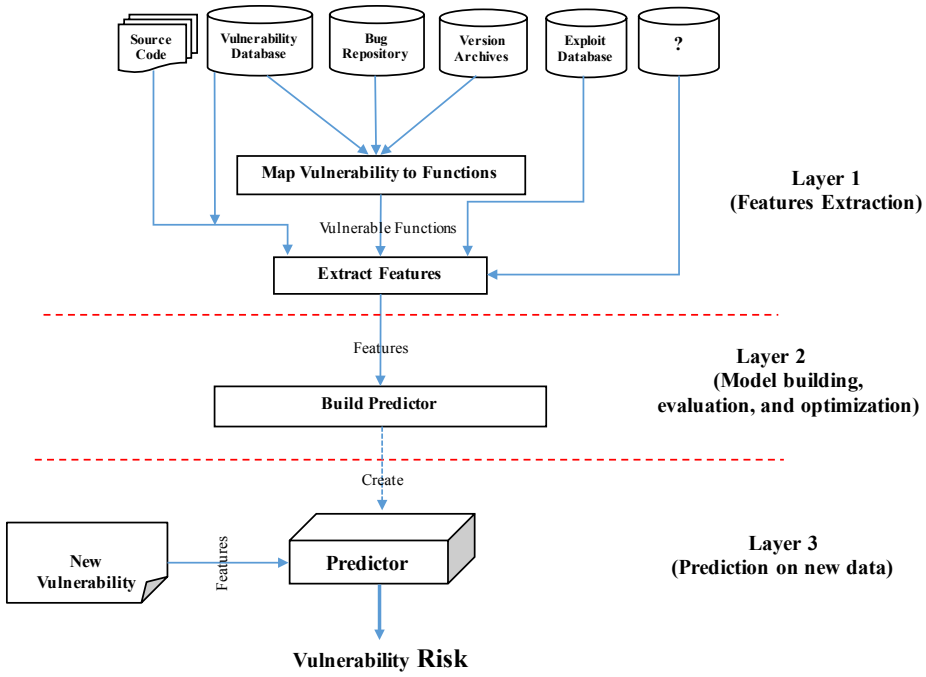


Figure 1.4: Proposed Framework

reported vulnerabilities from a predicted total number of vulnerabilities. In addition to estimating the risk of vulnerabilities discovery of the whole system, this dissertation propose a metric to assess an individual software vulnerability discovery termed Time To Vulnerability Disclosure (TTVD). TTVD is the time taken from when the version containing the vulnerability was first released until the time a vulnerability is discovered and hence disclosed to the public.

For addressing vulnerabilities exploitability risk, this dissertation proposes a novel vulnerability exploitability risk measure, Structural Severity. It is based on software properties, namely attack entry points, vulnerability location, the presence of the dangerous system calls, and reachability analysis. These properties represent metrics that can be objectively derived from attack surface analysis, vulnerability analysis, and exploitation analysis. In this dissertation, internal software attributes (metrics) that can be used to predict vulnerability exploitability risk have been examined. We characterize the vulnerable functions that have no exploit and the ones that have an exploit using eight metrics: Source Line of Code, Cyclomatic complexity, CountPath, Nesting Degree, Information Flow, Calling functions, Called by functions, and Number of Invocations. We first test the discriminative power of the individual selected metrics using the Welch t-test. Then we select

a combination of the metrics using three feature selection methods and evaluate their predictive power using four classifiers.

For validating CVSS Base metrics, this dissertation used two approaches. First, using the availability of exploits we evaluated the performance of the CVSS Exploitability factor and compared its performance to Microsoft (MS) rating system measures. The results showed that exploitability metrics in CVSS and MS do not correlate strongly with the existence of exploits (ground truth), and have a high false positive rate. The high false positive rate result makes me think about exploring different ground truth to explain why too many vulnerabilities have no exploit for them. To address this challenge, we introduced the vulnerability reward programs (VRPs) as a novel ground truth to evaluate the CVSS Base scores. Having more eyes on the code means that VRPs uncovered many more vulnerabilities and that makes finding and exploiting vulnerabilities more difficult for malicious actors. The fact that there are more number of vulnerabilities with a high CVSS scores and have no exploits or attacks is may be because vulnerabilities that are discovered by VRPs result in prioritized fixing.

1.3 Dissertation Outline

The rest of this dissertation is organized as follows. Chapter 2 introduces software vulnerabilities and their related topics such as vulnerabilities standards and databases, exploit database, open source software, and bug repositories. Chapter 3 presents the assessment of vulnerabilities discovery risk for the whole system. Chapter 4 describes the risk assessment of individual vulnerabilities. Chapter 5 explores the relationship between attack surface metric and vulnerabilities density. In Chapter 6, a measure for vulnerability exploitability is introduced. Chapter 7 investigates the characteristics and the models for prediction vulnerabilities exploitability. Chapter 8 evaluates CVSS Exploitability factor using the the availability of exploits vulnerabilities and compared its performance to Microsoft (MS) rating system exploitability measure. Chapter 9 introduces the vulnerability reward programs as a novel ground truth to evaluate the CVSS Base scores. Chapter 10 presents the conclusions and Chapter 10 outlines directions for future work.

1.4 Publication History

The materials in this dissertation have previously been presented at conferences ([7], [8], [9], [10], [11], [12], [13], [14]). Besides, one of the materials in the dissertation is currently under the review([15])

Chapter 2

Background

This chapter describes the concepts and topics related to software vulnerabilities.

2.1 Software Vulnerability Definition

A software vulnerability is defined as a defect in software systems which presents a considerable security risk [2]. A subset of the security related defects, vulnerabilities, are to be discovered and become known eventually. The vulnerabilities are thus a subset of the defects that are security related. The finders of the vulnerabilities disclose them to the public using some of the common reporting mechanisms available in the field. The databases for the vulnerabilities are maintained by several organizations such as National Vulnerability Database (NVD) [16], Open Source Vulnerability Database (OSVDB) [17], BugTraq [18], as well as the vendors of the software. Vulnerabilities are assigned a unique identifier using MITRE Common Vulnerability and Exposure (CVE) service.

2.2 Software Vulnerabilities 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 (CVE)

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 [19]. It is a dictionary of publicly known information of security vulnerabilities and exposures. CVEs common identifiers enable data exchange be- tween 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. CVEs stan- dardized 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 advi- sories).

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 (CWE)

Common Weakness Enumeration (CWE) is a list of software weakness types, and is sponsored by the National Cyber Security Division in the US Department of Homeland Security [20]. 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 the information for more than 860 programming, design, and architecture error types that can lead to exploitable vulnerabilities. Since around September 2007, National Vulnerability Database have provided some selected CWE names in the vulnerability database.

2.2.3 Common Vulnerability Scoring System (CVSS)

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 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. 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) [21] .

CVSS defines a number of metrics that can be used to characterize vulnerability. The measures termed scores are computed using assessments, called metrics, of vulnerability attributes based on the opinions of experts in the field. For each metric, a few qualitative levels are defined and a numerical value is associated with each level. CVSS is composed of three major metric groups: Base, Temporal and Environmental as shown in Figure 2.1. The Base metric represents the intrinsic characteristics of vulnerability, and is the only mandatory metric. The optional Environmental and Temporal metrics are used to augment the Base metrics, and depend on the target system and changing circumstances. The Base metrics include two sub-scores termed exploitability and impact. The CVSS scores for known vulnerabilities are readily available in the majority of public

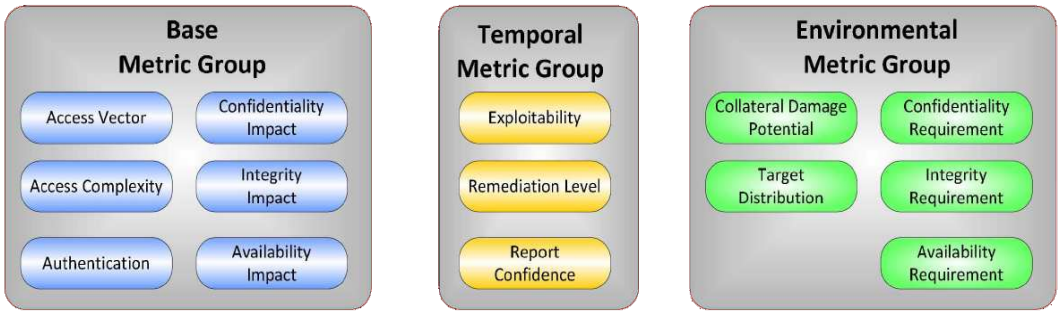


Figure 2.1: Three CVSS Metric Groups

vulnerability databases on the Web. A score is a 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.

The base metric group, range of [0.0, 10.0], represents the intrinsic and fundamental characteristic 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 of the three elements: 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).

The impact sub-score is composed of the three key aspects in information security components: confidentiality (C), integrity (I) and availability (A). 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. Since the central goal of CVSS is producing comparable vulnerability scores,

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/
DragonSoft Vulnerability Database	http://vdb.dragonsoft.com/
US-CERT Vulnerability Notes Database,(CERT)	http://www.kb.cert.org/vuls
French Security Incident Respense,Team (FrSIRT)	http://www.vupen.com/english/
Secunia	http://secunia.com/
Vulnview	http://www.vulnview.com/
CERIAS	https://coopvdb.cerias.purdue.edu/
Security Tracker	http://www.securitytracker.com/

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.

The optional environmental and temporal metrics are used to augment the Base Score metrics and depend on the target system and changing circumstances.

2.3 Public Vulnerability Databases

One of the first things to do for the vulnerability analysis is collecting the datasets to be analyzed. Fortunately, there are many publically available vulnerability databases on the Web; Table 1 shows some of them. Usually, they are overlap and complement each other, so there is no one best source. Many of them provide CVE identifiers, severity, CVSS scores, and published date. If it is available, they also provide vulnerability patch date, discovery date, vulnerability type, etc. There are many vulnerability databases and security advisories in 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.

2.4 Exploit Database

EDB records exploits and vulnerable software [22]. It is used by penetration testers, vulnerability researchers, and security professionals. It reports vulnerabilities for which there is at least

a proof-of-concept exploit. EDB is considered as a regulated market for the exploits. EDB contains around 24075 exploits as the time of writing this paper. Most of its data are derived from Metasploit Framework, a tool for creating and executing exploit code against a target machine. It provides a search utility that uses a CVE number to find vulnerabilities that have an exploit.

2.5 Open Source Software

Open source software is software whose source code is available for modification or enhancement by anyone [23]. Source code is the part of software that most computer users don't ever see. The open source codes that are used in the dissertation are Apache HTTP server, Linux Kernel, and Firefox.

2.6 Bug Repositories

A bug repository is a vital database in modern software development. Many software projects create and maintain bug repositories for storing and updating the information of problems or suggestions about projects. The widely available bug repositories have provided an important platform for investigating the quality of software. With the growth in scale, developers in large projects must handle a large number of bugs in bug repositories. For example, as for April 2015 the Linux repository has over 400, 000 commits. Bugzilla is a bug or issue tracking system [24]. Bug-tracking systems allow individual or groups of developers effectively to keep track of outstanding problems with their product [25].

Chapter 3

Assessing Software Vulnerability Discovery Risk

This chapter proposes a novel vulnerability discovery model, Folded model. This model address the risk of vulnerabilities discovery for the whole software when the pattern of the data is linear. It is a time based model that uses known reported vulnerabilities data to estimate the total number of the remaining vulnerabilities in a software. This number is then used to quantify the residual vulnerabilities by subtracting the number of known reported vulnerabilities from a predicted total number of vulnerabilities.

3.1 Vulnerability Discovery Risk Estimation at Software Level

Vulnerability discovery risk can be measured at the software level by estimating the number of the residual vulnerabilities in the system as shown in the following:

$$RVD = VD - KVD$$

RVD stands for residual vulnerabilities density, VD stands for vulnerabilities density, and KVD stands for known vulnerabilities density. Whereas know vulnerabilities can be determined from one of the vulnerabilities databases such as NVD, determining the total number of vulnerabilities of a system is hard. Recently, a few vulnerabilities discovery models (VDMs) have been proposed to estimate the number of total vulnerabilities in a given system. They include Rescorlas exponential model [26], Andersons thermodynamic model [27], and Alhazmi Malaiya Logistic (AML) model [28], each of them is based on its own assumptions and is characterized by its specific parameters.

Investigating the prediction capability and accuracy of these models has been studied by Alhazmi and Malaiya [29]. It has been found that the AML model generally fits the data for several software systems better than other models. The AML model is obtained using the assumption that

as the market share of a software increases, the rate of vulnerability discovery also increases. When the software starts losing its market share, or when there are a few vulnerabilities remaining to be found, the vulnerability discovery rate decreases [28]. Thus, the motivation of the vulnerability finders, both white hat and black hat, is driven by the market share. The AML model is logistic, and thus the increase and decrease in the discovery process is assumed to be symmetric around the peak. However, it has been noted [30, 31] that the discovery rate may not be necessarily symmetrical. This limitation of the AML model can possibly be addressed using alternative models that capture asymmetric behavior.

Kim [30], and Joh and Malaiya [32] have shown that asymmetric VDMs are feasible and have better performance than the symmetric models in some cases. In this dissertation, we examine the Folded model suggested by Kim [30], as an alternative VDM. Kim however did not examine the model using actual datasets. Here, we examined the applicability of the Folded VDM using actual vulnerability discovery data for four popular software systems. Specifically, we compare the Folded and AML models using goodness of fit and prediction capabilities for these datasets.

The chapter is organized as follows. In Section 3.1, the AML model is discussed and its potential limitations are identified. In section 3.2, the Folded VDM will be introduced. Section 3.3 presents the results of the comparison of the AML and Folded models using goodness of fit tests and prediction capabilities. Finally, the concluding comments are given along with the issues that need further research.

3.2 The Vulnerability Discovery Models

The VDMs proposed recently are somewhat analogous to software reliability growth model (SRGM), but there are significant differences. VDMs are probabilistic models for modeling the discovery rate of vulnerabilities in software systems [33]. These models use the historical data such as release date, the discovery date of vulnerabilities and possibly the system usage data. While the vulnerabilities are security related defects, they tend to be treated differently compared with ordinary software defects [34, 35]. Normal defects found after release are frequently ignored and not fixed until the next release because they do not represent a high degree of risk. On the other

hand, software developers need to patch vulnerabilities right after they are found, due to the high risks they represent. The security issues can greatly impact not only organizations such as banks, brokerage houses, on-line merchants, government offices but also individuals.

Quantitative risk analysis of systems with a continual vulnerability discovery has only recently started to be investigated. A few VDMs proposed by researchers include Anderson [27], Rescorla [26], Kim [30], Alhazmi and Malaiya Logistic model [36], Alhazmi and Malaiya Effort based model [37], Ozment and Schechter [38], and Chen et al. [39]. Figure 3.1 shows classification of vulnerability discovery models. Each model has its own mathematical representation and parameters. As a result, different VDMs can make somewhat different projections using the same data. No specific guidance is currently available about which models should be used in a given situation. Rescorla [26] has introduced quadratic and exponential VDMs. He fitted the

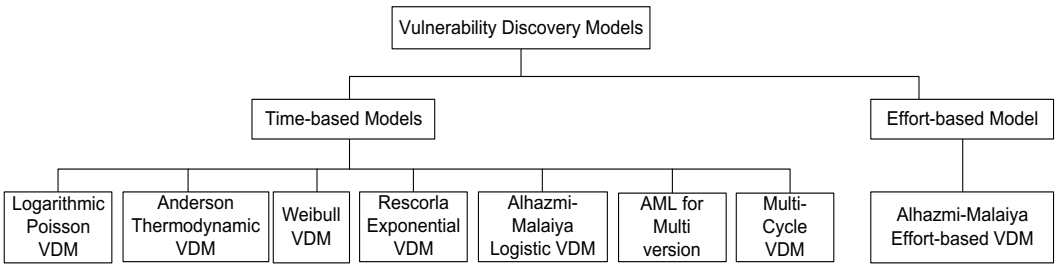


Figure 3.1: Taxonomy for Vulnerability Discovery Models

proposed models but did not evaluate their predictive accuracy. Anderson [27] proposed a thermodynamic vulnerability discovery model, but did not apply the model to any actual data. Alhazmi and Malaiya [28] proposed the logistic vulnerability discovery model, termed the AML model. The AML model presumes a symmetric software vulnerability discovery process. This model has shown a good statistically significant goodness-of-fit for the wellknown operating systems such as Windows and Red Hat Linux, and some Internet applications such as browsers and HTTP servers. Its predictive capability was tested by Alhazmi and Malaiya [29] and it has shown good results. In another study [36], they found that the AML model provides a better goodness-of-fit compared to Rescorla and Anderson models.

Alhazmi and Malaiya [36] have also proposed an effort-based model which utilizes the number of system installations as the independent factor instead of calendar time. They argued that it is

much more rewarding to discover a vulnerability in a system which is installed on a large number of computers. However, the effort-based model requires the number of users for a target product in market share which is not always easy to be obtained. Woo et al. [40] have examined the goodness-of-fit as well as the prediction capability for the effort-based model.

Joh et al. [31] have studied Weibull VDM, which was first proposed by Kim [30]. They argued that the assumption made by the AML model that the rate of discovering vulnerability is symmetric around the peak value is not always true. They used Weibull distribution to capture the asymmetric behavior as an alternative to the AML model. However, the Weibull model did not always provide a good fit.

3.3 The Symmetrical AML VDM

The AML VDM [28] is a time-based model. It assumes that at the release of the software the vulnerability discovery rate increases gradually. This is known as the learning phase in which the software gains market-share and installed bases remain small. After the learning phase, the system starts to attract more users and the number of vulnerabilities grows linearly. In this phase, which is known as the linear phase, the maximum vulnerability discovery rate is obtained by finding the slope. The learning phase is considered as the most important phase because most of the vulnerabilities will be discovered during this phase. However, when the system starts to be replaced by a newer version and users start to switch to the next version and as a result the vulnerability finders start to lose interest in finding vulnerabilities in the older version. As a result, the vulnerability discovery rate drops. Therefore, the cumulative number of vulnerabilities becomes stable. The three phases are shown in Figure 3.2(a).

The AML model assumes that the vulnerability discovery processes are controlled by the market share of the soft-ware and the number of the undiscovered vulnerabilities. The model assumes that the vulnerability discovery rate is given by the differential equation:

$$\frac{d\Omega}{dt} = A\Omega(B - \Omega) \tag{3.1}$$

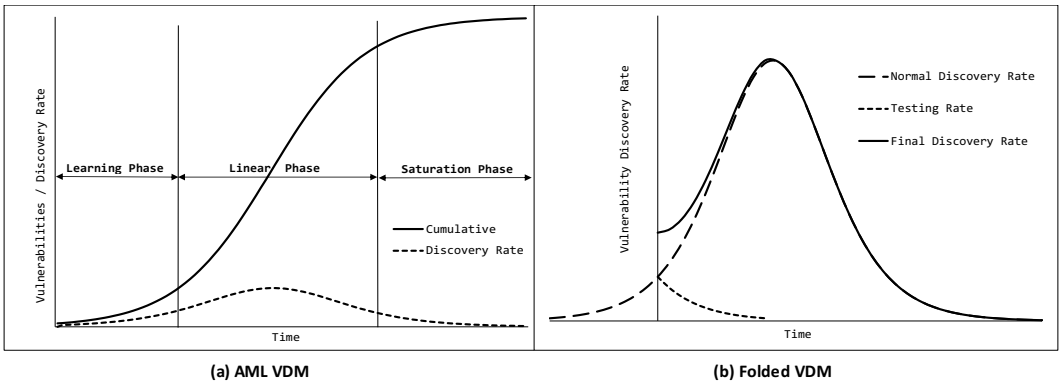


Figure 3.2: Vulnerability discovery process and rates for AML and Folded VDMs

Equation (3.1) has two factors. The first factor $A\Omega$, where A is a constant, increases as the market share increases, and $(B - \Omega)$, where B represents the total number of vulnerabilities, decreases as the remaining vulnerabilities decreases. Equation (1) can be solved to obtain the logistic expression for $\Omega(t)$:

$$\Omega(t) = \frac{B}{BCe^{-ABt} + 1} \tag{3.2}$$

Note that $\Omega(t)$ approaches B as the calendar time t approaches infinity. The parameters A and C determine the shape of the curve. C is a constant introduced while solving Equation (3.1).

AML model assumes a symmetrical vulnerability discovery rate as shown by the dotted curve in Figure 3.2 (a). Although the AML model has been found to fit real data of many software systems, there is no compelling reason why the rise and fall should be symmetric since they may be controlled by different factors. Some datasets do show a noticeable asymmetry [32]. These findings violate the symmetric assumption made by this model. Thus, looking for alternative VDMs that can deal with this trend is needed.

Actual data can show a departure from the s-shape assumed by the logistic model. In many cases, a software system gradually evolves as code is modified or patched or additional code is added. This will inject new vulnerabilities into the system which will delay the onset of saturation. In many cases, a new version is widely anticipated and is adapted by many users soon after its release. This will result in the learning period to shrink or even disappear. In the next section we consider the Folded VDM that offers the capability of modeling the behavior when the learning period is very small.

3.4 Asymmetrical Folded VDM

The normal distribution is symmetric around its mean and is defined for a random variable that takes values from $-\infty$ to $+\infty$. In some cases, a distribution is needed that has no negative values. Daniel [41] had proposed a half-normal distribution that folds the normal distribution at the mean that now corresponds to value zero. A more general version of it was proposed by Leone et al. [42] which is termed a Folded normal distribution that is defined for a random variable taking values between 0 and $+\infty$. It is obtained by folding the negative values into the positive side of the distribution. Whenever measurements of a normally distributed random variable are taken and the algebraic sign is discarded, the resulting distribution will be a Folded distribution. The folded distribution has been found usable in industrial practices such as measurement of flatness, straightens, and determination of the centrality of the sprocket holes in motion picture film [42]. The probability density function (pdf) and the cumulative distribution function (cdf) of the distribution are both derived from their counterparts in the normal distribution, pdf and cdf.

The Folded distribution, as applied to vulnerability discovery, is illustrated in Figure 3.2 (b). The vulnerability discovery starts at time t = 0 which corresponds to the release time of the software. Since the initial value is non-zero because of the contribution of folding, the learning period is minimized as shown in Figure 3.2. Hence, here, we propose the Folded VDM as an asymmetrical model as suggested by Kim [30]. The proposed vulnerability discovery rate of the Folded model is given by Equation (3.3).

$$f(t) = \frac{\gamma}{\sqrt{2\pi\sigma}} \left[e^{-\frac{(t-\tau)}{2\sigma^2}} + e^{-\frac{(t+\tau)}{2\sigma^2}} \right], t \geqslant 0 \tag{3.3}$$

Here, t represents the calendar time, τ is a location parameter, is a scale parameter, and γ represents the number of vulnerabilities that will be eventually discovered. The second term in Equation (3.3) represents the part of the distribution folded to the positive side as shown in Figure 3.2 (b) which shows the discovery process for the Folded VDM. The cumulative number of vulnerabilities described by Folded VDM is presented in Equation (3.4).

$$F(t) = \frac{\gamma}{2}\left[erf\left(\frac{t-\tau}{\sqrt{2\sigma}}\right) + erf\left(\frac{t+\tau}{\sqrt{2\sigma}}\right)\right], t \geqslant 0 \tag{3.4}$$

where $erf()$ is the error function which is used to calculate the integral from zero. Figure 3.3 shows the cumulative Folded vulnerability discovery process along with the behavior of AML. Figure 3.3 also shows the lack of the learning phase for the Folded model. Compared to AML, the

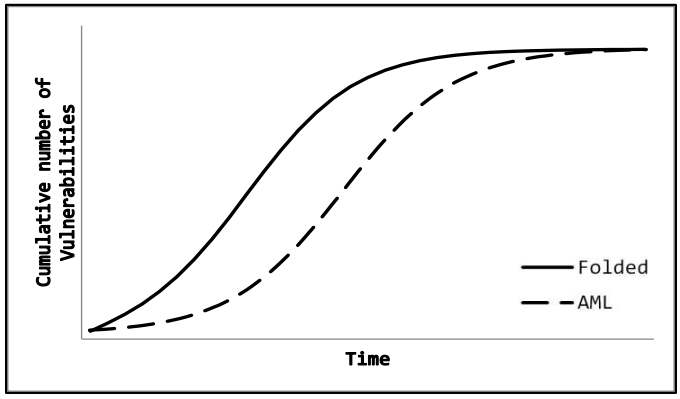


Figure 3.3: General cumulative vulnerability discovery trends

Folded VDM has shorter learning phase or missing learning phase which makes the normal distribution asymmetric. It results in a higher discovery rate at the beginning which may be especially applicable to the cases where $\Omega(t)$ plot is linear even at the beginning.

3.5 Model comparisons and observations

We have fitted the AML and Folded VDMs to the four datasets: Windows 7, OSX 5.x, Apache Web Server 2.0.x, and Internet Explorer 8. Table 3.1 shows released dates, market shares and the number of vulnerabilities in each system. These software systems have been chosen because they have relatively short learning phase, and thus they can be used to test whether the proposed Folded model is capable of capturing the learningless vulnerability discovery trend. Figure 3.3 shows model fittings for the two VDMs on the four datasets. While visually both models appear to fit well, in the next section we analyze the goodness of fit by evaluating the p-values.

Table 3.1: Datasets Used

	Released	***Vuln.**	**Share(%)**
Win 7	2009-JUL	80	**25.11
OSX5.x	2007-OCT	211	**1.30
Apache 2.0.x	2000-MAR	68	***62.71
IE 8	2009-MAR	72	**33.06
*http://nvd.nist.gov/ on JAN 2011. Only after the released date. **http://marketshare.hitslink.com/ on APR 2011. ***http://news.netcraft.com/ on May 2011. For total version.			

Table 3.2: χ^2 Goodness of fit tests

	AML				Folded			
	A	B	C	p-value	τ	σ	γ	p-value
Win 7	2.52E-03	96.75671	0.148963	0.6970	0.063742	11381.63	64427.18	0.9673
OSX5.x	7.49E-04	206.8057	0.064464	0.9845	0.063742	1969.209	15029.21	0.9428
Apache 2.0.x	9.88E-04	62.69023	0.113327	1.0000	0.065227	47.81145	66.26354	1.0000
IE 8	3.22E-03	73.39949	0.185473	0.7337	0.065227	97.30989	407.1494	0.9839

3.5.1 Goodness of Fit analysis

Table 3.2 shows the model parameters along with the p-values of χ^2 goodness of fit tests. The χ^2 statistic (χ_s^2) is calculated as:

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

where o_i and e_i are the observed and expected values at i^{th} time point respectively. The null hypothesis for the test is that the actual distribution is well described by model fittings. Hence, in Table 3.2, p-value close to 1 means good model fitting whereas less than 0.05 is considered as not being statistically significant when we select the level as 0.05.

Figure 3.4 suggests that all the datasets show linear discovery trends for the period examined and either do not have a learning phase or it is very short. The main reason for linearity in the early part can be because of quick adoption of the version considered as a result of the anticipation of the release. Both the users and the vulnerability finders are not waiting for the software to become sufficiently popular, they take it for granted that it will be. During the later part, the linear behavior could be that since the systems are continually evolving, new code is being injected time to time which introduces additional vulnerabilities. The saturation phases would not be seen in the

vulnerability discovery process for such systems until they stop evolving. In general, we observed that Folded VDM captures the starting and ending data points better than AML model for these datasets.

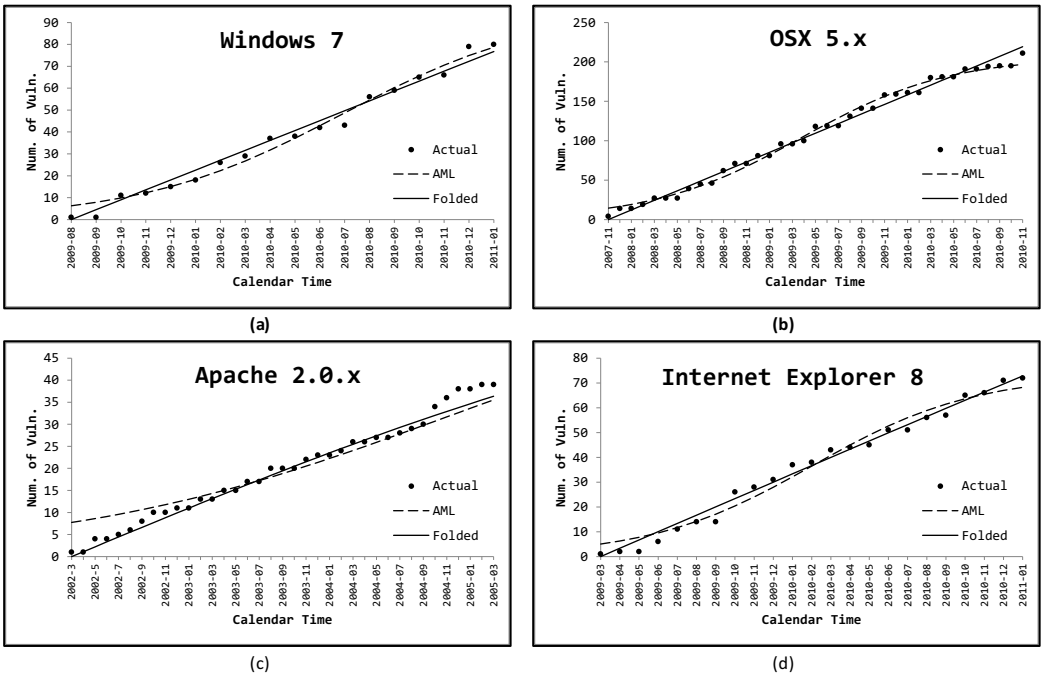


Figure 3.4: Model fitting for AML and Folded VDMs

P-values in Table 3.2 indicate that all the model fittings are statistically significant since p-value is greater than 0.05. Windows 7 and Internet Explorer 8 fit the Folded model better whereas AML fits OSX 5.x slightly better. Apache 2.0.x data fits both models very well with p-value 1. However, visual inspection tells that Folded model performs better at the beginning and the end of the time period. Folded model provides p-values which are consistently greater than 0.9 while AML has a lower value in the two cases.

3.5.2 Prediction capabilities

The main use of a model is predicting the future trends based on the available data, rather than reviewing the past behavior. In that sense, prediction capability should be considered more important than model fitting. Models having good fitting results may not necessarily possess good prediction abilities of the process behavior changes with time.

We use two normalized prediction capability measures [43], Average Error (AE) and Average Bias (AB), as given in Equation (3.5) and (3.6) respectively. AE is a measure of how well a model predicts throughout the time period, and AB indicates the general bias of the model which assesses its tendency to overestimate or underestimate.

$$AE = \frac{1}{n}\Sigma_{t=1}^{n} \mid \frac{\Omega_t - \Omega}{\Omega} \mid \tag{3.5}$$

$$AB = \frac{1}{n}\Sigma_{t=1}^{n} \frac{\Omega_t - \Omega}{\Omega} \tag{3.6}$$

In the equations, n is a total number of time points (in months in this case), and Ω is the actual number of total vulnerabilities. Ω_t is the estimated number of total vulnerabilities at time t. The normalized prediction error values for each time point are plotted in Figure 3.5. The x-axis represents the time as a percentage where 0% and 100% correspond to the release date and the final data point that the model is attempting to predict. Table 3.3 shows the values for AE and AB.

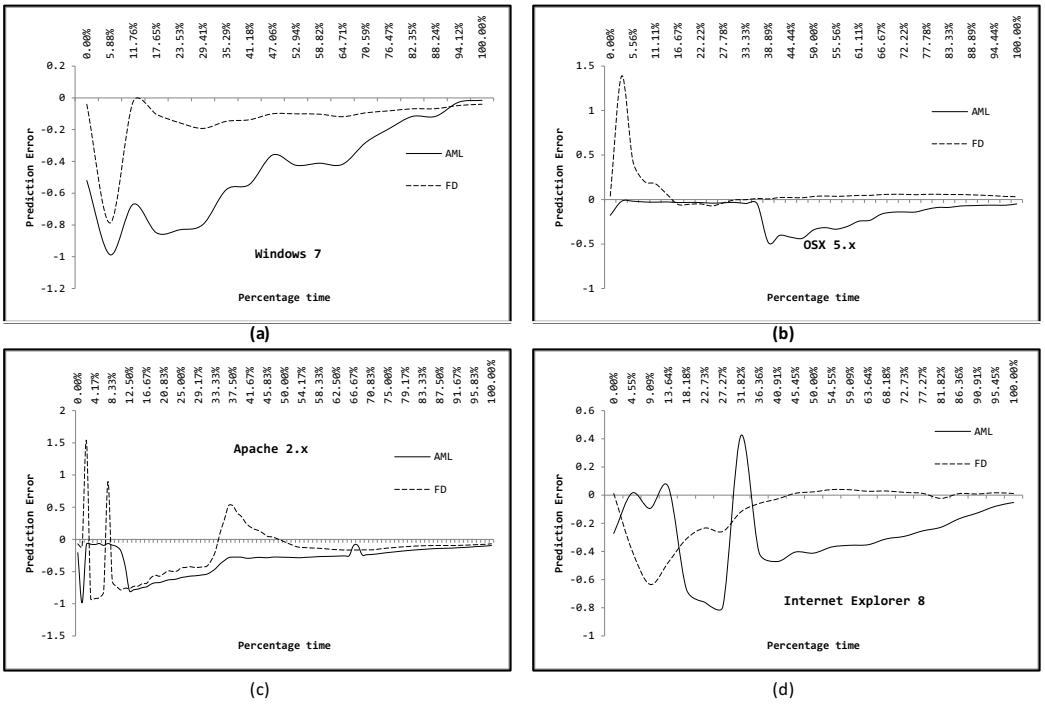


Figure 3.5: Prediction errors for AML and Folded VDMs

The error plots in Figure 3.5 show that the Folded model provides a more stable prediction with a significantly less error in most situations. In Table 3.3, the AB and AE values show that

Table 3.3: Average Bias and Average Error (% Time: $0\% \sim 100\%$)

	AB		AE	
	AML	Folded	AML	Folded
Win 7	-0.45222	-0.13405	0.452221	0.134048
OSX5.x	-0.14514	0.081817	0.145141	0.096575
Apache,2.0.x	-29.6239	-17.7062	29.62394	28.87495
IE 8	-0.27722	-0.09876	0.320391	0.121494

the Folded model almost always performs better than AML. For Windows 7, OSX 5.x and Internet Explorer 8, Folded model outperformed the AML. For Apache 2.0.x, the two models result in somewhat similar outcomes for the AE value.

3.6 Conclusion and Future Work

This research examines a new vulnerability discovery model based on the folded normal distribution and evaluates its applicability using real datasets for four major software products. It also compares the new proposed model with the symmetrical AML vulnerability discovery model.

Software developers need to estimate the resources needed for development of patches for the vulnerabilities that are likely to be found in future. A quick patch release after the discovery of a vulnerability will significantly reduce the security risk to the organizational and individual users. An organization needs to assess the resources needed to address future vulnerabilities; including the patch application effort and reserve resources needed to alleviate the impact of possible intrusions. Both of these require the use of a vulnerability discovery model that can make sufficiently accurate vulnerability discovery rate projections.

The AML model is the only model that has been formulated to specifically describe the discovery process. The fitting and prediction capability of the AML model has been found to be better than other models for most datasets. However, it has also been found that the discovery trends can be different in different circumstances. In one hand, for the software systems that has been in the market for long period of time, their behavior has been found to be better described by symmetric models such as AML logistic model which exhibits both learning and a saturation phases in addi-

tion to the linear phase. On the other hand, some systems have a vulnerability discovery rate that tends to be linear from the beginning and thus lack a learning phase.

In this research, we have formally defined and investigated the Folded vulnerability discovery model based on folded normal distribution which is asymmetric by definition and can represent a learningless discovery process. Its model fitting and prediction capabilities have been tested and com-pared with the AML model for four popular software systems. While both Folded and AML models have been found to fit the vulnerabilities datasets of Windows 7, OSX 5.x, Apache Web server 2.0.x and Internet Explorer 8 well, they differ significantly in the prediction capability. The short learning phase is apparently captured by the Folded model much better than the AML logistic model for the four datasets. The folded model consistently outperforms the AML model in terms of the prediction capabilities for the datasets with no learning phase.

The Folded model needs to be further investigated by applying it to as many software systems as possible and comparing it with other competing models. That will allow development of guidelines as to when this model would be most suitable. The significance of the parameters also needs to be examined.

Chapter 4

Assessing Individual Vulnerability Discovery Risk

This chapter shows how the risk of individual vulnerability discovery can be assessed. As different vulnerabilities have different opportunity of being discovered based on their properties such as access complexity, in this chapter we proposes a novel approach to estimate time to discovery using Time To Vulnerability Disclosure (TTVD). TTVD is the time taken from when the version containing the vulnerability was first released until the time a vulnerability is discovered and hence disclosed to the public. TTVD can be influenced by extrinsic factors such as discoverers' skills and effort and/or intrinsic attributes of a vulnerability and types. We investigate 799 reported vulnerabilities of Google Chrome. The vulnerabilities rewards program (VRP) data of Google Chrome provide us insight into the effect of effort and skills on the TTVD. We also consider 156 reported vulnerabilities of Apache HTTP server that does not use VRP. We examine the relationship of TTVD with CVSS Base metrics and vulnerabilities' types to capture the intrinsic characteristics of a vulnerability. We assess the individual and the combined relationships of those metrics with TTVD using Spearman correlation and machine learning models such as decision trees, neural networks, support vector regressions, relevance vector machines, and Gaussian processes.

4.1 Introduction

Assessing the risk presented by a software vulnerability is very important for decision makers (including development, assurance, and business) to prioritize their actions. Risk models such as Open Web Application Security Project (OWASP) [5] and Common Weakness Scoring System (CWSS) [6] assess a vulnerability risk based on the likelihood of vulnerability discovery and exploitation and the impact. However, the Common Vulnerability Scoring System (CVSS) [4] Base score does not consider the likelihood of vulnerability discovery and it assumes the vulnerability

as already being discovered and it only considers the likelihood of vulnerability exploitation and the impact. Considering the likelihood of vulnerability discovery, however, is important when the vulnerability is only known to developers at the time of discovery and the developers may choose to increase the priority of vulnerabilities that are most likely to be discovered. OWASP and CWSS subjectively estimate the likelihood of vulnerability discovery based on discoverers' needed skills and effort and the availability of source code.

In this research, we propose a quantitative metric termed Time To Vulnerability Disclosure (TTVD). TTVD is the time taken from when the software has first been released until the time a vulnerability is disclosed to the public. Arbaugh et al. [44] have proposed a vulnerability lifecycle model and have identified all possible states (from birth:after software is released, to death: after the vulnerability is patched) that a vulnerability can enter during its lifetime. Frei et al. in [45, 46] have extended the proposed model in [44] and have used it to quantify the risk of vulnerability exposure period in three levels: black, gray and white. However, neither Arbaugh nor Frei have investigated or measured the TTVD.

TTVD can be influenced by extrinsic factors such as discoverers' skills and effort and/or intrinsic attributes of a vulnerability and types. We examine 799 reported vulnerabilities of Google Chrome. The vulnerabilities rewards program (VRP) data for Google Chrome can provide us insight into the effect of effort and skills on the TTVD. Besides, we consider 156 reported vulnerabilities of Apache HTTP server that does not use VRP. We also examine the relationship of TTVD with CVSS Base metrics (Base metrics capture the intrinsic characteristics of a vulnerability [4]) and with vulnerabilities' types. We assess the individual and the combined relationships of those metrics and the types with TTVD using Spearman correlation and machine learning models such as decision trees, neural networks, support vector regressions, relevance vector machines, and Gaussian processes.

TTVD depends on the likelihood of a vulnerability being discovered. Thus, knowing TTVD can be used in quantitatively assessing the likelihood of vulnerability discovery. The actual discovery date, however, is not usually reported to the public. According to [45], few sources such as Open Source Vulnerability Database (OSVDB) and the security bulletins of commercial vul-

nerability markets provide information, such as time of vendor notification and time of purchase, that can be used to derive the disocvery date. First, OSVDB is no longer supported and second, the security bulletins of commercial vulnerability markets only contain the time a vulnerability is reported to the vendor and not the actual discovery date [47]. Besides, our source of the studied data, NVD, does not provide information about the vulnerability discovery date.This requires us alternative approaches to find the date of discovery. In most of the cases, the disclosure date will be around 60 days after the vendor notifaction date. Frei et al. in [46] have stated that the vulnerability disclosure date implies its discovery.

This chapter is organized as follows. In Section 4.1, the related work are discussed. In the next section, the TTVD and its related topics such as vulnerability lifecycle, software release date, vulnerability disclosure date, and influential factors of TTVD are introduced . In Section 4.3, the evaluation and the results of the TTVD are presented. Section 4.4 presents the discussion. Finally, concluding comments is given along with the issues that need further research.

4.2 Related Work

Schneier [48] have identified the window of exposure that a vulnerability presents when exists in a software. Arbaugh et al. [44], have suggested a vulnerability lifecycle model and using three vulnerabilities, they measured the number of intrusions during vulnerability lifecycle. Frei et al. [45, 46], extended Arbaugh et al.'s model and using more than 80000 vulnerabilities, they identified and measured three types of risk exposures in black, gray, and white. However, None of these previous works consider investigating or measuring TTVD.

McQueen et al. [47] have defined 0-Day vulnerability and analyzed its lifespans. They have defined the 0-Day lifespan as the time from the vulnerability discovery date to the public disclosure date. They were only able to identify the actual vulnerability discovery dates for 15 vulnerabilities that were provided to them by a researcher. They also compared CVSS Base score to mean lifespan. However, in this study, we consider TTVD starting from the vulnerability birth date and we do not only correlate TTVD with CVSS Base score but also with the individual CVSS Base

metrics and the vulnerability type. We also study the effect of the vulnerability reward programs on the TTVD.

Joh and Malaiya [49] formally defined a risk measure. They utilized the vulnerability lifecycle and applied Markov stochastic model to measure the likelihood of vulnerability exploitability. They also used the impact related metrics of CVSS to estimate the exploitability impact. However, Joh and Malaiya did not take into consideration of TTVD.

Sommestad et al. [50], and Holm et al. [51], studied the effort required to discover a software vulnerability. They used expert estimates on vulnerability discovery effort using Cook's classical method. Finifter et al. [52], examined the characteristics of Google Chrome and Mozilla Firefox vulnerability reward programs (VRPs). The authors found that VRPs improved the likelihood of finding latent vulnerabilities. Younis et al. [10], used VRPs as a ground truth to assess the CVSS Base metrics. In this study, however, we examine VRP data of Google Chrome to have an insight into the effect of the discoverers' effort and skills on the TTVD.

OWASP [5] and CWSS [6] consider assessing the likelihood of vulnerability discovery. Both of them assess the likelihood of vulnerability discovery subjectively based on the attacker skills, effort, and the availability of the source code. In this research, we propose a quantitative measure for the likelihood of vulnerability discovery using TTVD.

4.3 Time-To-Vulnerability-Disclosure (TTVD)

TTVD is the time taken from when the software has first been released until the time a vulnerability is disclosed to the public. Our approach of measuring TTVD is based on the vulnerability lifecycle. A vulnerability is created as a result of a coding or specification mistake. A vulnerability lifecycle is a model that describes the states that a vulnerability can enter during its lifetime [44]. Figure. 4.1 shows a vulnerability lifecycle states: birth, discovery, and a discovery may be followed by any of following: internal disclosure, patch, exploit/script or public disclosure. More details about the vulnerability lifecycle can be found in [44] and [45]. The birth of the vulnerability is defined to start after the software is released or deployed. TTVD can be measured as shown

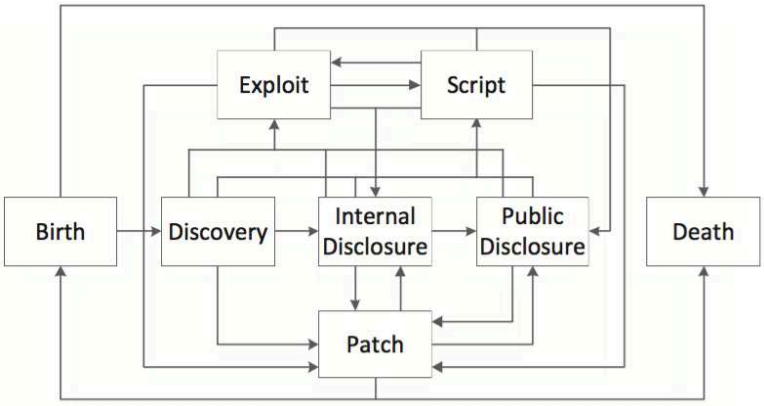


Figure 4.1: Vulnerability Lifecycle

below:

$$TTVD = VulnerabilityDisclosureDate - ReleaseDateoftheFirstAffectedVersion \quad (4.1)$$

Therefore, measuring TTVD requires determining the first affected software release date and the vulnerability discovery date. The following subsections will explain this in details.

4.3.1 Software Release Date

Frei et al. [45] indicated that the time of a vulnerability birth can be determined by looking back at the history of the vulnerability after its discovery or disclosure. However, the authors did not take that into consideration. In this work, we determine the vulnerability birth date and use it to measure TTVD. The vulnerability birth date is determined by finding the release date of the first affected software version. In order to find the release date, we first find the affected software versions. After that, we determine the release date of the first affected version. The National vulnerability DataBase (NVD) [16] records the version information of the software that is affected by a vulnerability. We have noticed, however, that some vulnerabilities do not have version information. Table 4.1 shows a list of Chrome versions affected by CVE-2014-1702 vulnerability provided by NVD. Due to the page limitation, we only show 3 affected versions out of a 107.

Table 4.1: Products Affected By CVE-2014-1702

#	Vendor	Product	Version
1	Google	Chrome	33.0.1750.0
2	Google	Chrome	33.0.1750.38
3	Google	Chrome	33.0.1750.146

After sorting this list, we have found out that the Chrome version 33.0.1750.0 is the first affected one. However, until the date of this paper, NVD does not provide information about the date of the affected software versions. To determine the date of the affected version, we used the release date information found in [53] and [54] for Chrome and in [55] and [56] for Apache HTTP server. Table 4.2 shows some samples of how the date of the affected software has been determined.

Table 4.2: First Affected Software Release Date

Vulnerability	Software Version	Release Date
CVE-2011-2170	11.0.696.65	05/06/2011
CVE-2012-2819	20.0.1132.0	6/26/2012
CVE-2013-0926	26.0.1410.0	03/26/2013
CVE-2014-1731	34.0.1847.130	04/24/2014
CVE-2015-1233	41.0.2272.102	03/24/2015

4.3.2 Vulnerability Disclosure Date

Vulnerability disclosure date (VDD) is the date on which the vulnerability is first released to the public [45]. Acording to the Orginzation for Internet Safety [57], the disclosure time can take 30 calendar days after the notification date. Besides, CERT (Computer Emergency Readiness Team) [58] has stated that vulnerabilities reported to them will be disclosed to the public 45 days after the initial report (regardless of the existence or availability of patches or workarounds from affected vendors). CERT also has stated that for difficult vulnerabilities to remediate, extension may be granted. In addition, Google [59] suggests that 60 days is a reasonable upper bound for a genuinely critical issues in widely deployed software. In most of the cases, the disclosure date will be around 60 days after the vendor notification date.

4.3.3 Influential Factors of TTVD

TTVD can be influenced by extrinsic factors such as discoverers' skills and efforts and/or intrinsic attributes of a vulnerability and types. To capture the effects of the discoverers' skills and effort, we use vulnerabilities rewards program (VRP). VRPs are programs adopted by software vendors to pay security researchers, ethical hackers and enthusiasts for exchange of discovering vulnerabilities in their software [52]. Zhao et al. [60], found a significant positive correlation between the monetary incentive and the number of vulnerabilities reported. The VRP data for Google Chrome has been used so an insight into the effects of efforts and skills on the TTVD may be provided. Besides, to examine the effects when the VRP is not used on TTVD, we consider Apache HTTP server data.

To capture the effects of intrinsic attributes of a vulnerability, we use CVSS Base score [4]. The Base score captures the intrinsic and fundamental characteristic of a vulnerability. CVSS Base score measures severity based on exploitability (the ease of exploiting a vulnerability) and impact (the effects of exploitation). Here, we use the overall Base score values and the values of the Base score individual metrics. The Base score individual metrics include the exploitability and impact metrics. The exploitability metrics are access vector, access complexity, and authentication. Whereas the impact metrics are confidentiality (CI), integrity (II), and availability (AI) metrics. The CVSS Base score assigns a score in the range between 0.0 and 10.0. CVSS score from 0.0 to 3.9 corresponds to Low severity, from 4.0 to 6.9 to Medium severity and 7.0 to 10.0 to High severity.

Hafiz and Fang [61] showed that some types of the vulnerabilities are harder to detect and require more efforts than the others. To capture the effects of vulnerability types on TTVD, we identify vulnerabilities types for every vulnerability using the Common Weakness Enumeration (CWE) [20] provided by the NVD.

4.4 Evaluation and Results

In this section, we present the results of measuring and analyzing TTVD. To measure and analyze TTVD, 799 reported vulnerabilities of Google Chrome during the period 2007 to Februrary 2016 and 156 reported vulnerabilities of Apache HTTP server during the period 1999 to Februrary 2016 were collected from NVD [16] as shown in Table 4.3. It should be noted that we were not able to find the release date for 369 vulnerabilities of Chrome. Besides, we could not find rewards information for 72 vulnerabilities of Chrome because of the unauthorized access permission, "you are not authorized to access this data." Besides, we could not find the release date and version information for 27 vulnerabilities of Apache HTTP server.

Table 4.3: Chrome and Apache HTTP Server Vulnerabilities

Software	**Total # of Vulnerabilities**	**Period**	**Examined Vulnerabilities**
Chrome	1240	2007 - Feb 2016	799
Apache	183	1999 - Feb 2016	156

Table 4.4 shows the data used to measure the TTVD. From the NVD, we obtained the vulnerability release date and the affected software version. As the NVD does not provide information about the release dates for the affected versions, we used the release date data found in [53] and [54] for Chrome and in [55] and [56] for Apache HTTP server. Using the YEARFRAC function in [62], we calculated the TTVD.

Table 4.4: Data Used to Measure TTVD

CVE	**Vulnerability Release Date**	**Softwae Version**	**Software Release Date**	**TTVD**
CVE-1999-1293	12/31/99	Apache 1.2.5	03/07/1995	4.817
CVE-2011-3348	09/20/2011	Apache 0.8.11	12/28/1995	15.728
CVE-2008-7246	09/18/2009	Chrome 0.2.149.27	09/02/2008	1.044
CVE-2010-1665	05/03/2010	Chrome 0.2.149.27	09/02/2008	1.669

Table 4.5: The Attributes of the Examined Vulnerabilities

CVE	CVSS Bases Score	Exploitability	Impact	AV	AC	AU	C	I	A	CWE	Reward
CVE-1999-1293	10	10	10	NW	L	NR	COMPLETE	COMPLETE	COMPLETE	399	N/A
CVE-2011-3348	4.3	8.6	2.9	NW	M	NR	NONE	NONE	PARTIAL	Not Given	N/A
CVE-2008-7246	5	10	2.9	NW	L	NR	NONE	NONE	PARTIAL	399	NRW
CVE-2010-1665	7.5	10	6.4	NW	L	NR	PARTIAL	PARTIAL	PARTIAL	199	500$
NW:Network, L:Low, NR:Not Required, NRW: Not Rewarded, N/A: Not Applicable.											

For every vulnerability that has its TTVD measured, we collected its CVSS Base score and individual metric values and types from the NVD for the both selected software. However, the rewards data for Chrome were collected as follows. For every existing link in NVD to Chrome bug database, we collected the assigned vulnerability's rewards data from Chrome bug repository [63]. We have found that the VRP data in the Chrome bug database have started to be recorded starting 2010. Table 4.5 shows a part of the attributes of the selected vulnerabilities (the first two vulnerabilities are for Apache and the second two are for Chrome). CVSS Base score is the severity score of a vulnerability. The exploitabiloity metrics are access vector (AV), access complexity (AC), authentication (AU). The impact metrics are confidentiality (C), integrity (I), and availability (AV). The CWE assigns a number for the vulnerabilities' type.

4.4.1 Datasets Analysis

Figure. 4.2 shows the description of the CVSS Base score data and the access complexity. We have found that the access vector and authentication values are almost constant for Chrome and Apache and thus we did not show their descriptions. For Chrome, only four vulnerabilities out of 799 require local access and the rest requires network access whereas one vulnerability requires a single authentication. For Apache, 23 vulnerabilities out of 156 require local access and the remaining requires network access whereas only three vulnerabilities require a single authentication.

As can be seen from Figure. 4.2, there are more medium severity vulnerabilities for Apache and almost the same number of medium and high severity vulnerabilities for Chrome. Besides, for both software, there are more ease to access vulnerabilities (low access complexity) and few hard to access vulnerabilities (high access complexity) for both software. We have noticed that when the access complexity is low, the low and high impact percentage for both software are almost the same

whereas the medium impact percentage for Chrome is 64.63% compared to 22.22% for Apache. In addition, when the access complexity is medium, the low impact percentage for Apache is about 68.75% whereas in Chrome the low and medium impact percentage are almost the same and the high impact percentage is about 18.14% compared to 4.6% in Apache.

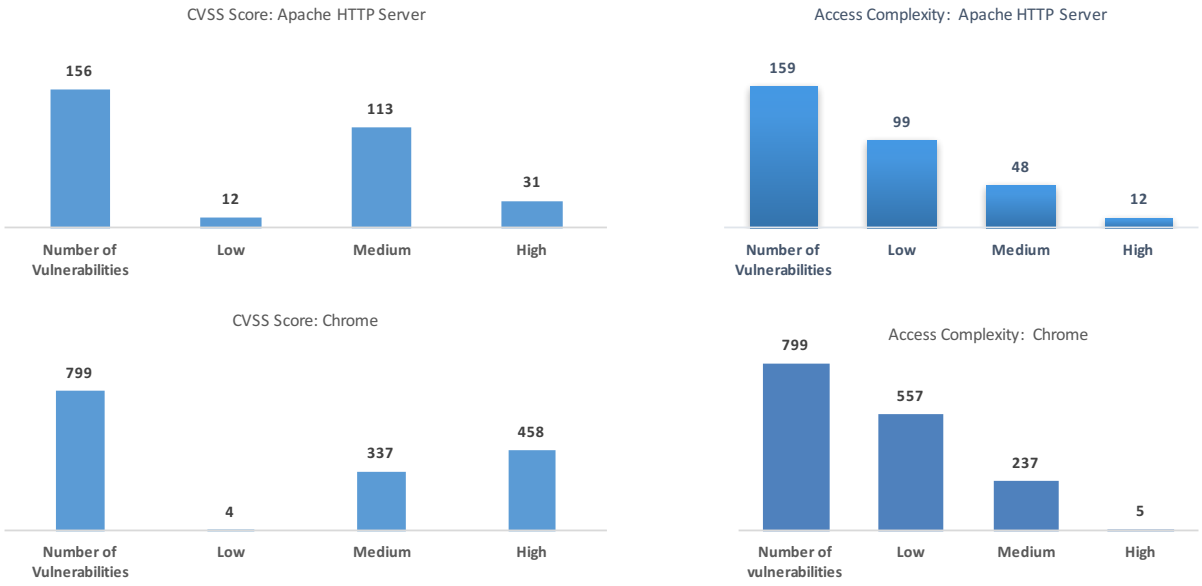
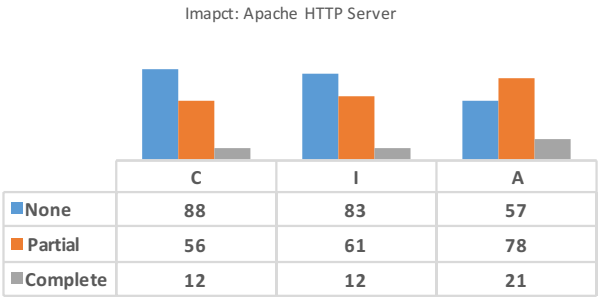
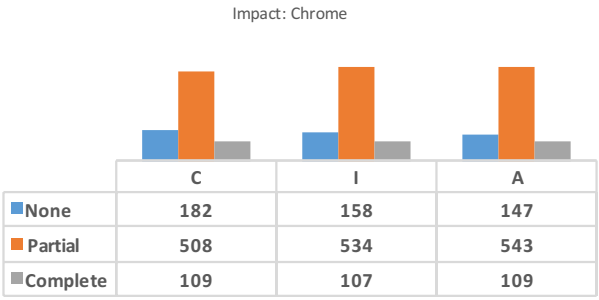


Figure 4.2: CVSS Base scores and Access Complexity Values

This observation can be also captured when considering the individual metrics values of the impact as shown in Figure. 4.3. As can be seen, there are more partial impact values for Chrome vulnerabilities than none values whereas there are almost the same number of partial and none values for Apache vulnerabilities. However, we have also noticed that the majority of the exploitability values for both datasets are high (Apache:27 are medium or low, Chrome: 10 are medium or low).



	C	I	A
None	88	83	57
Partial	56	61	78
Complete	12	12	21



	C	I	A
None	182	158	147
Partial	508	534	543
Complete	109	107	109

Figure 4.3: Impact Metrics Values for C, I, and A

Figure. 4.4 shows the vulnerability types for both datasets. Only 41.66% (65 out of 156) of the Apache vulnerabilities have been assigned CWE type by NVD, whereas only 75.46% (603 out of 799) of the Chrome vulnerabilities have CWE number assigned. We have noticed that all vulnerability types of Apache dataset are similar to the vulnerability types of Chrome. On the other hand, there are 19 different types of vulnerabilities in Chrome. It should be noted that the types in Figure. 4.4 are ordered from bottom to top. The following vulnerability types are just found in Chrome: CWE-17: Code,CWE-19: Data handling, CWE-22: Path traversal, CWE-59: Link following, CWE-254 : Security feature, CWE-287: Improper authentication, CWE-289: Improper access control. However, we have noticed that the vulnerabilities types CWE-399, CWE-20, CWE-264, CWE-119, CWE-189, and CWE-79 are the commonly found types in both datasets.

Figure. 4.5 shows the amount of the reward and their frequency. Out of the 799 vulnerabilities, 335 vulnerabilities have been rewarded. It should be noted that the not rewarded vulnerabilities are most likely have been discovered by internal discoverers (41.92%) whereas the rewarded vulnerabilities have been discovered by external discoverers (58.07%) [52]. As can be seen, around 195 vulnerabilities have been paid between 500$ to 1000$.

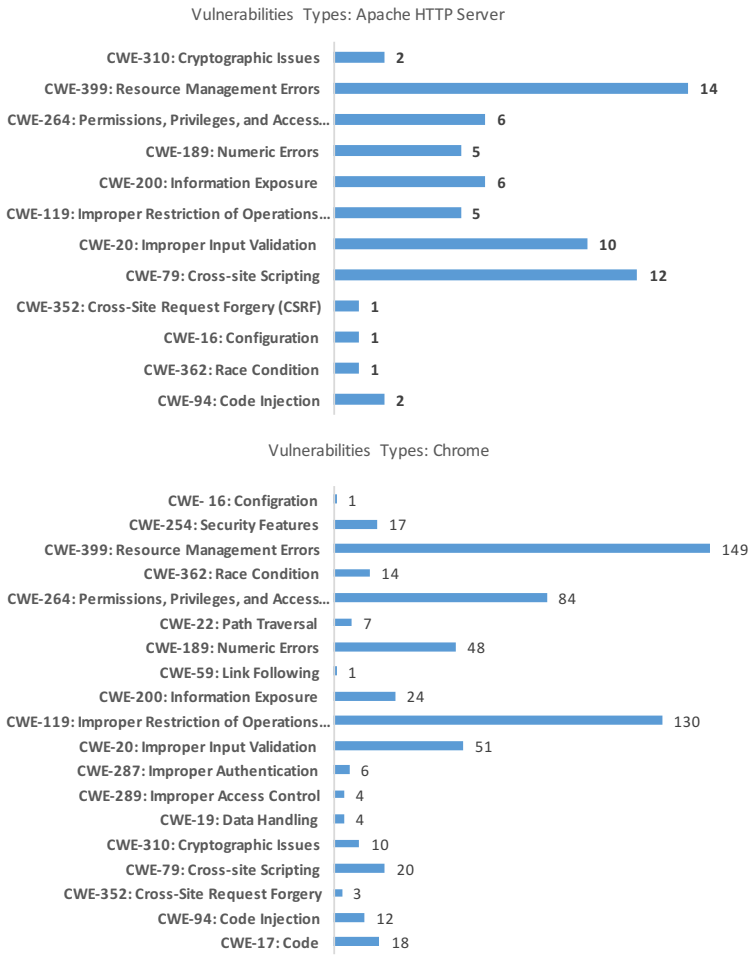


Figure 4.4: Vulnerabilities Types and Their CWE Numbers

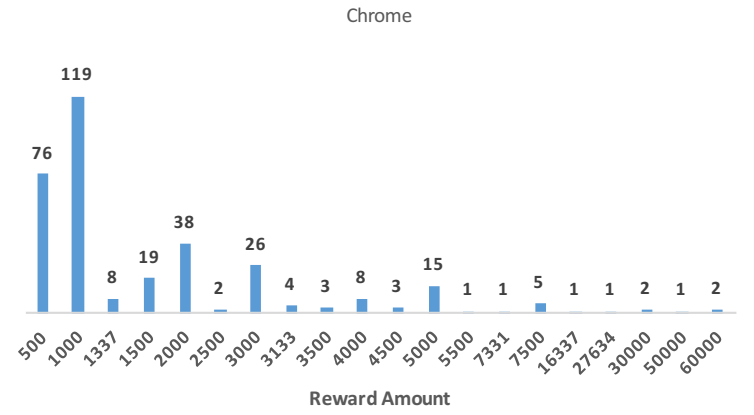


Figure 4.5: Chrome Rewarded Vulnerabilities

4.4.2 TTVD Relationships Analysis and Evaluation

In this subsection, we present the relationships analysis of the TTVD with the considered attributes. We first assess the individual relationships of CVSS Base metrics and types with TTVD using Spearman correlation. The reason we chosen to use Spearman is that one variable (TTVD) is continuous whereas the other variables are either on an ordinal or categorical scales. The result of the correlation measurement is represented by r an p values. The r value represents the strength of the correlation whereas the p value shows the significant of the correlation. The correlation is considered significant if the p value is less than 0.05. We then us machine learning models such as decision trees, neural networks, support vector regressions, relevance vector machines, and Gaussian processes to examine the selected attributes relationships when they are combined. The decision trees have been selected because they provide information about each factor's contribution on predictions and this helps to interpret a tree to make different levels of predictions. We have chosen the other regression models because they provide a relatively higher predictive accuracy with nonlinear approximation. After that, we examine the relationships between vulnerabilities types and TTVD and analyze the effects of the VRP on the TTVD.

Individual Attributes

We expect to find the vulnerabilities that have a high CVSS Base score severity (easy to exploit and has a sever exploitation impact) to have a significant short TTVD with a p value less than 0.05. The results show that the overall Base score values do not have a significant correlation with TTVD for both datasets, Apache (r= -0.085, p= 0.288) Chrome (r= 0.007, p= .834). We have also looked at the differences in the means of the Base score and the TTDV. The results show no noticeable difference, Apache (mean when the score is L= 3.275, M= 4.674, High= 4.484), Chrome (mean: L= 0.6243, M= 0.2356, High= 0.2465).

We also examined the correlation between the exploitability and the impact individual values and TTVD. We also expect to find a vulnerability with a high exploitability or impact values to have a significant short TTVD. The overall impact values have no significant correlation with TTVD, Apache (r= 0.112 ,p= 0.165), Chrome (r= 0.02557355, p= 0.4704). On the other hand, the

results show a significant negative correlation between the explotability and TTVD for both software, Apache ()r= -0.2711113, p= 0.000619), Chrome (r= -0.163, p= 0.00000369). The negativity explains that the higher the exploitability value, the easier the exploitation of the vulnerability.

After looking at the exploitability factor data, we have found that the authentication and access vector metrics are constant whereas the access complexity shows variability. Therefore, we only examine the correlation between the access complexity and the TTVD. We expect to find a vulnerability with a low access complexity to have a significant short TTVD. The results show a significant positive correlation between the access complexity and TTVD for both software, Apache (r= 0.307, p= 0.0000973) and Chrome (r= 0.1767618, p= 0.000000495).

We also examined the correlation between the impact factor metrics and TTVD. We expect to find a vulnerability with a partial or complete values of the confidentiality, integrity, or availability to have a significant short TTVD. The results show only a significant positive correlation between the confidentiality and TTVD for Chrome dataset (r= 0.0729 , p= 0.000000495) and a significant positive correlation between the integrity and TTVD for Apache dataset (r= 0.1826154 , p= 0.0.0225).

Figure. 4.6 shows the relationship between every vulnerability type and TTVD. The vulnerabilities types of Chrome are numbered from 1 to 19 based on the order in Figure. 4.4 from bottom to top. Looking at the Chrome Boxplot, we can see that vulnerabilities types (1, 3, 7, 8, 10, 11, and 13 to 19) are found in a very short time. On the other hand, the other 6 types (2, 4, 5, 6, 9, and 12) have more noticeable TTVD variability. Figure. 4.6 also shows the relationship between every vulnerability type and TTVD of Apache HTTP server dataset. The vulnerability types of Apache are also numbered based on the order in Figure. 4.4 from bottom to top. Vulnerabilities of types 1 to 4 have been found in a relatively short time compared to the others, whereas vulnerabilities of type 7 and 12 have the most TTVD variability. We have noticed that the same vulnerabilities types that takes more TTVD in Apache are taking less TTVD in Chrome.

Figure. 4.7 shows the TTVD for Apache and Chrome. It should be noted that Chrome has been using vulnerability reward program (VRP) since 2010. The effect of the VRP is noticeable on TTVD. The mean for Apache TTVD is 4.512 and the maximum TTVD is 17.570, whereas the

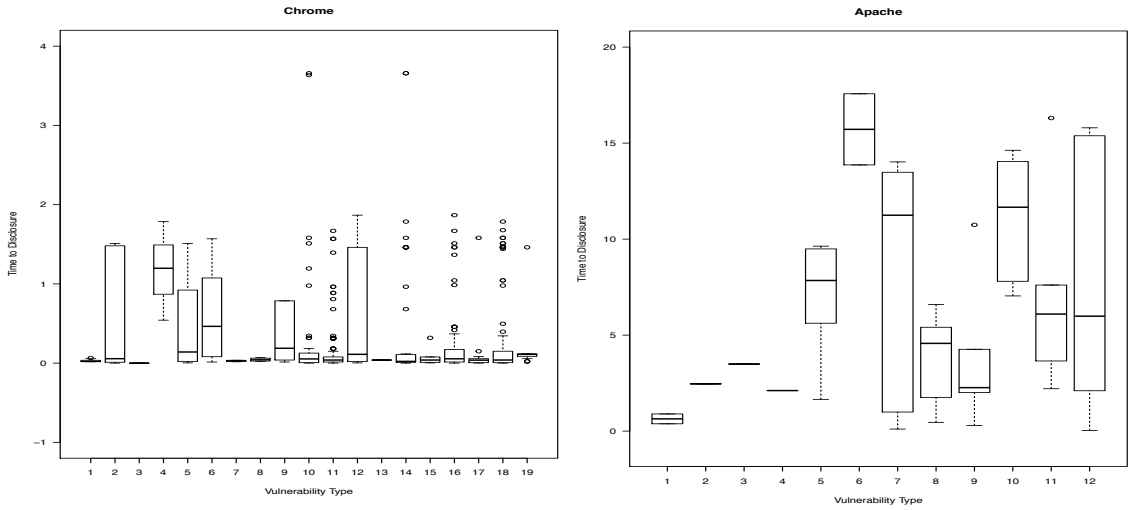


Figure 4.6: Chrome and Apache HTTP Server Vulnerabilities Types and TTVD

mean TTVD for Chrome is 0.2438 (around three months, 0.2438 * 12(months)) and the maximum TTVD is 3.6580. Due to the difference in the release dates (Apache in 1998 and Chrome in 2008) and the date the Chrome starts using of the VRP, we also compared the TTVD of the two datasets only from 2010 onward. The Boxplot looks very similar to the one in Figure. 4.7 and the descriptive statics are as follows. The mean for Apache TTVD is 7.157 and the maximum TTVD is 13.230, whereas for Chrome the mean and the the maximum TTVD are respectively 0.2173 and 3.6580.

We have also considered the difference between the TTVD for rewarded vulnerabilities and not rewarded vulnerabilities and the results are shown in Figure. 4.8. We have noticed that the rewarded vulnerabilities have smaller TTVD. The mean and the maximum TTVD for rewarded and rewarded vulnerabilities are respectively 0.1 and 1.7 and 0.2, 3.7.

Combined Attributes

To examine the correlation between the considered attributes and TTVD, we apply the mentioned above regression models. It should be noted that we are using the machine learning models to assess the correlation of the selected attributes when they are combined and for now, we are not focusing on building a model or making a prediction. The regression test uses eight features: CVSS Base score, access vector, access complexity, authentication, impact on confidentiality, integrity,

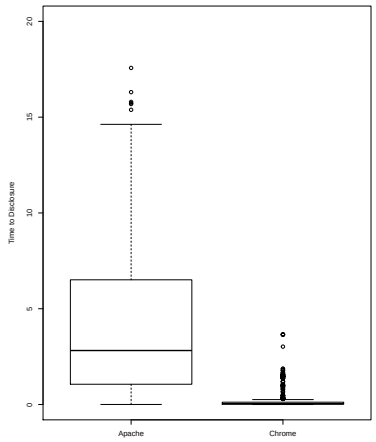


Figure 4.7: Comparing TTVD for Apache HTTP Server and Chrome

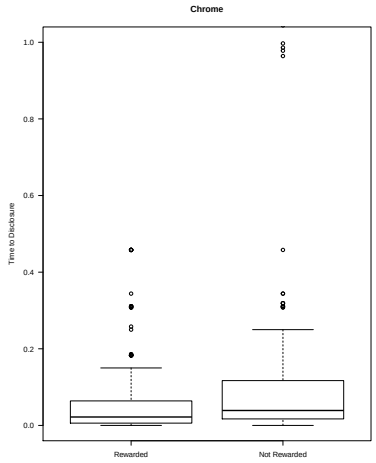


Figure 4.8: Comparing TTVD for Rewarded and Not Rewarded Vulnerabilities

and on availability, and the type of vulnerabilities. Each categorical input is transformed corresponding discrete numerical values for mathematical regression models. It should be also noted that there are overlaps on the inputs (109 unique inputs out of 799 samples in Chrome dataset and 63 unique inputs out of 156 samples in Apache dataset). In other words, there are multiple outputs (TTVD) for the same inputs (the selected features). In addition, we did not include the VRP (which repents the discoverers' individual expertise or efforts). Table 4.6 shows the 10-fold cross validation result on Chrome and Apache data sets. Decision tree and Gaussian process show the best performance for both datasets.

Table 4.6: The Best Testing RMSE with Each Algorithm

	Tree	NNet	RVM	SVR	GP
Chrome	**12.09**	12.58	12.3	12.81	12.38
Apache	33.56	37.76	35.74	37.4	**31.94**

Interestingly, simple decision tree (CART) can make an equivalent prediction to other algorithms. As can be seen from the residual plots in Figure. 4.9, the large error is produced because of the outliers and it captures majority of points with mean prediction over evenly distributed residual space.

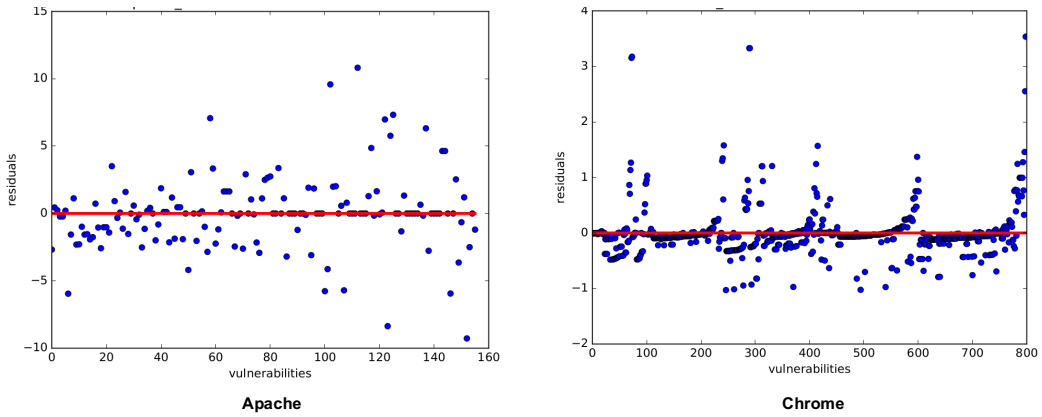


Figure 4.9: Residual Plots of the Decision Tree Fit on Chrome and Apache Datasets.

Figure. 4.10 shows the generated decision trees that give us a further reasoning analysis. First, the decision tree shows us that trees are split majorly by CVSS Base score and the type of vulnerabilities with a bit contribution from access complexity and integrity impact. Interestingly, when the CVSS Base score was considered individually, it did not show a significant correlation. Chrome TTVD estimation tree uses the Base score as a root to determine the TTVD while Apache estimation tree uses the vulnerability type as a root node. It should be noted that whenever the types lees or great then is encountered, that means less or greater than the order of the vulnerabilities types mentioned in Figure. 4.6. Based on the CEW number, the type in the decision tree can be determined.

Figure. 4.11 shows the neural network and its weights. The row index from 0 to 7 represents the features CVSS base score, access vector (AV), access complexity (AC), authentication (AU),

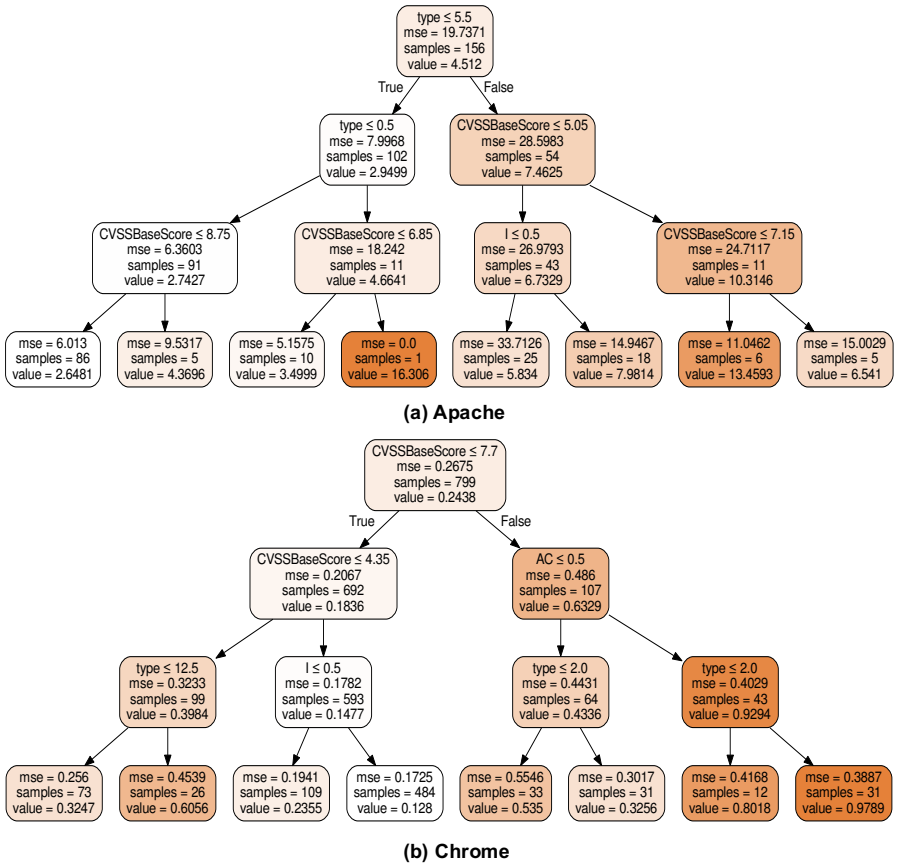


Figure 4.10: Top 3 Levels of the Decision Trees

impact on confidentiality (C), impact on integrity (I), impact on availability (A), and the type of vulnerabilities respectively. The 2 columns and 5 columns represent corresponding hidden units. We observe similar factoring from neural network weights. In Chrome data, the weights for the CVSS Base score (first row in the image) have the higher value than the other attributes and the weights for the vulnerability type have a relatively strong values. Here, we observe higher weight values in 1st and 4th hidden units, although it is not presented here, the output layer enforces more on 2nd and 3rd layer. Thus, eventually the Base score contributes more on the final prediction.

4.4.3 Threat to validity

In this paper, we have considered the datasets for only two products, the Apache HTTP server and Google Chrome. However, they are both very significant examples. The Apache HTTP server has 183 vulnerabilities belonging to different categories. Besides, the lines of the code varies be-

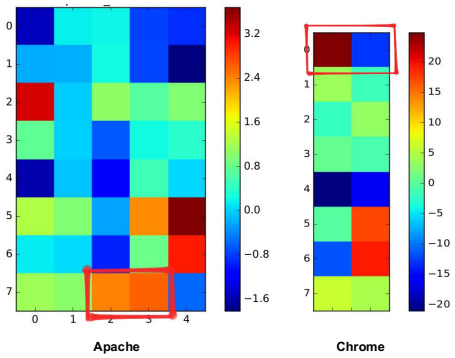


Figure 4.11: First layer neural network weights

tween 50,712 LOC to 358,633 LOC. Chrome has larger number of vulnerabilities, more than 1200. Its size in the lines of the code is 4,490,488 LOC. It also has a greater variety of vulnerabilities. We are aware that the vulnerabilities release date provided by NVD does not reflect the actual disclosure date to the public. We are not aware of any other data sources that provide reliable and complete date on the actual disclosure or the discovery date.

4.5 Discussions

We noticed that many vulnerabilities have a low access complexity values and few have a high access complexity values. The low access complexity value can be explained by two facts. First, the vulnerabilities discoverers and the techniques they use are able to find the low hanging fruit first (easy to access vulnerabilities). Second, base on our work [10], we realized that more rewards are paid when a proof of concept about exploiting a vulnerability is provided and thus the low access complexity vulnerabilities are relatively easier to be proved. For having a few high access complexity vulnerabilities, it could be explained by the fact that there are just a few of them or the current techniques' coverage levels are not yet ready to reach a deeper to reach vulnerabilities.

We have also noticed that there are more types of vulnerabilities in Chrome than in Apache HTTP server. One possibility could be because of the VRP. As more eyes are in the code, there is a higher chance that not only the number of the vulnerabilities will be discovered but also varieties of types. However, further research is required to investigate this observation. We have also observed that a specific type of vulnerabilities have a shorter TTVD. Looking at the CEW detailed

description found in [20], we have found that most of these vulnerabilities have either more than one methods of detection or have a medium or a high likelihood of exploitability (provided by CWSS measure [6]). Besides, some vulnerability types have longer TTVD. We have realized that these vulnerabilities are very popular, so it could be that the developers or testers are aware of them enough to avoid them during the development phase.

We have also observed that the rewarded vulnerabilities have smaller TTVD. This could be explained by the fact that VRPs have a larger and diverse groups of security researchers and contributors that outperformed the internal security teams or penetration testing teams [60]. Besides, we have noticed that almost half of the rewarded vulnerabilities have been paid either 500$ or 1000$. Looking at the point number 3 to 5 under the reward amounts section in [64], we can see that establishing exploitability or providing a Proof of Concept (PoC) or with a poor quality of PoC could be the reason. In other words, establishing exploitability worth more than discovering the vulnerability. We have noticed that there are more medium and high impact percentage for low and medium access complexity of Chrome dataset. This could be driven by the rewards paid by Google for more impactfull vulnerabilities.

We have also observed that the CVSS Base scores did not correlate with the TTVD when considered individually. The authors in [47] have observed the same thing when they compare 0-Day vulnerability lifespan with CVSS Base score. However, when the CVSS score was combined with other correlated attributes such as types, access complexity, and some impact metrics, CVSS Scores have been found indicative. According to Forman [65], a feature (or two features) that is completely useless by itself (themselves) can be useful when taken with others (together).

4.6 Conclusion and Future Work

In this chapter, we have introduced a new metric named Time To Vulnerability Disclosure. We have also examined its relationships with CVSS Base score and its individual metrics as well as the vulnerabilities types. We have also studied the affect of the presence and the absence of vulnerability reward program on the TTVD. The results show the importance of the metrics in explaining the effects of vulnerability rewards program and its potential to be characterized by some

intrinsic attributes. The machine learning analysis shows that relying on measuring the correlation of the individual attributes alone is a good thing to be performed first, however, combing several attributes together could provide insightful results. The results have also open up an avenue for a future work.

First, the NVD should enhance its data about version information and release dates. The recommendation provided by Glanz et al. in [66] should be taken into consideration by the NVD. Even though the vulnerabilities types have shown a great insight, finding a way to order them in a particular manner is very important for assessing the risk of vulnerability disclosure and discovery. Third, considering other type of software that use VRPs, such Firefox, and finding a way to include the VRPs as an attribute is very helpful. Finally, looking for other attributes that can be related to the TTVD is in the top of the list of our priority.

Chapter 5

Relationship Between Attack Surface And Vulnerability Density

This chapter explorers the relationship between attack surface and vulnerability density metrics. Finding a relationship could help in assessing the risk of vulnerabilities exploitability risk.

5.1 Introduction

Measurement is considered as powerful technique that can be used to evaluate security to an optimal level [67]. The Measurement to some security aspect of a system is realized by a metric. A security metric is a quantifiable measurement that indicates the level of security for a system [67]. Lately, several software security metrics have been proposed, and are still under improvement, by researchers in academia, industry, and government. Some of those are: relative vulnerability, static analysis tool effectiveness, attack surface, vulnerability density, flaw severity and severity-to-complexity, security scoring vector for web applications, and etc. [68]. Each of them is based on its specific perspective, target, and assumptions and measures different characteristics and properties of software security. They are mainly used to help decision makers in resource allocation, program planning, risk assessment, and product and service selection. They also make certain that decisions are not only based on subjective perceptions.

Essentially, quantifiable security metrics, as in the physical science, are supposed to provide precise knowledge to decision-makers. However, software is not obedient with the law of physics and has its own challenges [40]. As security is a software attribute, its measurement is not a trivial task [69]. According to [70], the lack of validation and comparisons between such metrics is considered a main challenge, which in turn makes their usability risky. Moreover, there is no single security metric that can quantify all aspects of security, and hence having a framework that combines different metrics is an important step toward measuring multiple aspects of security [70].

The main objective of this research is to understand the nature of those challenges by studying the relationship between two security metrics namely: attack surface and vulnerability density. Our choice of these two metrics is based on the following factors. Firstly, the former is already in use by Microsoft and applied by other major software companies, such as Hewlett-Packard (HP) and SAP. On the other hand, the latter is selected due to the fact that vulnerability density, as a normalized metric, allows subsequent releases of varying size to be compared [71]. Thus, this metric satisfies our objective of comparing the number of vulnerabilities of two different releases of the same system, Apache HTTP server. Apache has been chosen in particular because it is a major software component of the internet, has the highest market share among the Http servers [40], and availability of source code.

P. Manadhata et al. in [72] have investigated the validity of the attack surface metric by relating the number of vulnerability reports with the attack surface for two FTP Daemons, ProfFTPD and Wu-FTPD, along the method dimension. They have found that the attack surface of Wu-FTPD and ProfFTPD related to their reported vulnerabilities, i.e., the former attack surface is larger than the latter and thus the number of vulnerabilities of the former is larger than the latter. In their investigation, they applied the attack surface metric to compare the security of two different systems with the same functionality and then related it to the reported number of vulnerabilities. However, in our study we apply the attack surface metric to compare the security of two different versions of the same system and then relate it to their vulnerability density.

Vulnerability density metric was introduced and applied to major operating systems, namely Microsoft Windows and Read Hat Linux, in [71]. Besides, S. Woo et al in [40] have compared the vulnerability density of two HTTP web servers: Apache and IIS (Internet Information Service). In both studies the vulnerability density metric has been found significant and useful. However, vulnerability density metric has not been applied to subsequent releases of the same system. Moreover, it has not been compared with another security metric. In this study, we compare vulnerability density metric with the attack surface metric along the method dimension. Besides, we investigate the visibility of how these two metrics can complement each other.

5.2 Attack Surface Metric

A systems attack surface is the subset of the systems resources that are used by an adversary to attack the system [69]. The resources are referred to as methods (e.g., API), channels (e.g., sockets), and data items (e.g., input strings). This means that more number of available resources indicate larger attack surface and hence the system is less secure [73]. Notably, only some of these resources are considered as part of the attack surface. Moreover, their contribution to the attack surface measurement is not equal. To the relevant resources to be identified, the entry point and exit point framework is used [69]. Besides, the resource contribution is estimated using damage potential and effort ratio [69].

5.2.1 Entry Point and Exit Point Framework

The entry point and exit point framework is a formal framework that defines the set of entry points and exit points (methods), the set of channels, and the set of untrusted data items from the source code of a system [69]. Entry and Exit points are the methods that an attacker uses to either send or receive data from the system [2]. Channels are the means that used by an attacker to connect to the system [69]. Untrusted data items are the data that the attacker can either send or receive from the system [69].

A method can be either a direct or indirect entry point and/or a direct or indirect exit point [69]. In one hand, a method is a direct entry point if it receives data directly from the environment; read method defined in unistd.h in C library is an example [72]. Besides, a method is indirect entry point if it receives data from direct entry point [69]. On the other hand, a method is a direct exit point if it sends data directly to the environments system [69]; fprintf method defined in C library is an example [72]. Moreover, a method is indirect exit point if it sends data to direct exit point [69].

5.2.2 Damage Potential and Effort Ratio

Damage potential and access effort ratio is an informal means that are used to estimate damage potential and effort in terms of resources attributes [69]. The damage potential depends on the

methods privilege, the channels type, and the data items type [74], whereas, the effort depends on the rights of the resource that the attacker needs to acquire to use a resource in an attack [74]. The attackability of a method, a channel, and a data item is given as follows:

$$ac(method) = \frac{privilage}{accessright}$$
$$ac(channel) = \frac{type}{accessright}$$
$$ac(dataitem) = \frac{type}{accessright}$$

where $ac()$ is the attackability. The user of this metric is responsible for assigning a numeric values for privilege levels, types of the channel, and types of data items [74]. However, the following should be taken in considerations: the higher the privilege, channel type, or data item, the higher the damage potential, whereas the higher the access right the higher the effort [74].

5.2.3 Attack Surface Measurement Method

A system attack surface, for a given system, is measured along three dimensions: method, channel, and data items as follows:

- The entry and exit points framework is used to identify the set of methods (M), channels (C), and untrusted data items (I).
- Damage potential and effort ratio means, der(), is used to estimate the total contribution of the method: derm(m), channel: derc(c), and data items: derd(d).
- The systems attack surface is the triple:

$$< \Sigma m \in Mderm(m), \Sigma c \in Cderc(c), \Sigma d \in iderd(d) >$$

5.3 Vulnerability Density Metric

5.3.1 Vulnerability Classification

Distinction among vulnerabilities is advantageous, especially when examining the nature and extent of the problem is required. It also helps in determining the most effective kinds of protective

actions. Vulnerability classification is a growing field of research. Several methods of classifications have been proposed [40]. A vulnerability classification should be considered ideal if it poses desirable properties such as mutual exclusiveness, clear and unique definition, and coverage of all software vulnerabilities [40]. Vulnerabilities can be classified based on type, cause, severity, impact and source, etc. [40].

Nonetheless, the type and severity of vulnerabilities will be the focus of this study. In [40], it has been observed that the high severity vulnerabilities tend to be discovered and patched earlier than the medium and low vulnerabilities. On the other hand, two types of vulnerabilities namely: Authentication Issues and Permission, Privilege, and Access control; are believed to reduce the attack surface measurement if they are patched [69]. However, we believe not only the type but also the severity of vulnerabilities might have an effect on the attack surface measurement.

Vulnerability Based on Severity

Severity level of vulnerability indicates how serious the impact of exploitation can be. Three severity levels are often defined; high, medium and low. However, some other organizations use three to five levels and use their own definition for severity [40]. The NVD of the National Institute of Standards and Technology has used Common Vulnerability Scoring System (CVSS) metric for vulnerability severity with ranges from 0.0 to 10.0 [10]; CVSS uses many factors to determine the severity where the range from 0.0 to 3.9 corresponds to low severity, 4.0 to 6.9 to medium severity and 7.0 to 10.0 to high severity. The NVD (2010) describes three severity levels as follows:

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

Vulnerability Based on Type

Vulnerability type in National Vulnerability Database (NVD) is assigned using the Common Weakness Enumeration (CWE) maintained by the MITRE Corporation [19]. CWE is used as a classification method to distinguish CVEs by the type of vulnerability they represen. As in the [16], the type of vulnerabilities that are related to attack surface can be classified into:

- Authentication Issues: failure to properly authenticate users.
- Permissions, Privileges, and Access Control: failure to enforce permissions or other access restrictions for resources, or a privilege management problem.
- Cross-Site Scripting (XSS): failure of a site to validate, filter, or encode user input before returning it to another users web client.
- Format String Vulnerability: the use of attacker-controlled input as the format string parameter in certain functions.
- SQL Injection: when user input can be embedded into SQL statements without proper filtering or quoting, leading to modification of query logic or execution of SQL commands.
- OS Command Injections: allowing user-controlled input to be injected into command lines that are created to invoke other programs, using system () or similar functions.
- Information Leak / Disclosure: exposure of system information, sensitive or private information, fingerprinting, etc.

The other CWE types of vulnerabilities which might not be as relate as the above to the attack surface are: Credentials Management, Buffer Errors, Cross-Site Request Forgery (CSRF), Cryptographic Issues, Path Traversal, Code Injection, etc. [16].

5.3.2 Vulnerability Density Metric

Vulnerability density is a measure of the total number of known vulnerabilities, per thousand lines of code VKD, divided by the size of the software entity being measured [40]:

$$V_k = \frac{KnownVulnerabilities}{Size}$$

Vulnerability density is a normalized measure, given by the number of vulnerabilities per unit of code size. The size is typically measured either in Lines of Code or the installed system in bytes. The former has been chosen due to its simplicity and its correspondence to defect density metric in the software engineering domain. Vulnerability density is used to compare software systems that come under the same category. It can also be used to estimate the number of residual vulnerabilities. Assessing the risk can also be achieved by using vulnerability density. Besides, deciding when to stop testing as well as maintenance planning can be realized using vulnerability density metric [40].

5.4 Apache HTTP Server

Apache HTTP server is a Web server. It is developed and maintained by an open community of developers under the auspices of the Apache Software Foundation. Apache HTTP server is simply a piece of software that responds to requests for information sent by web browsers [75]. It provides information using static HTML pages as well as dynamic and interactive services to the clients using database queries, executable script, etc. It also supports functions such as serving streaming media, email, etc. In the presence of the cloud computing systems, it can also support virtual implementations of applications and operating systems. Apache can support variety of operating systems such as UNIX, Windows, Linux, Solaris, Novell NetWare, FreeBSD, Mac OS X, etc.

Apache HTTP server has gone through a number of improvements after its initial launch, which led to the release of several versions: 1.3.x, 2.0.x, 2.2.x, 2.3.x, and 2.4.x. This verity of releases has been one of the drivers for choosing it to be our case study. This diversity enables us to observe the improvement of the security among some of the subsequent releases. Besides, the availability

Table 5.1: Market share of the top web servers on the Internet

Product	Vendor	Web Sites,Hosted	Percent
Apache	Apache	378,267,399	64.91%
IIS	Microsoft	84,288,985	14.46%
nginx	Igor Sysoev	56,087,776	9.63%
GWS	Google	18,936,381	3.25%

of it source code and its usage share were also a choosing factors. According to [76], Apache web server is used by over 64%, Table 5.1. As Apache HTTP server releases 2.2.x takes more than 88% of the usage share among Apache releases, according to [77], and as there is a reasonable gap between the releases 2.2.x and 1.3.x, which might help in observing security improvement, both releases have been chosen for this study. The two main components of Apache web server are Apache Core and Apache Modules [75]. The former provides the main functionality of Apache HTTP server such as allocating requests and maintaining and pooling all the connections. On the other hand, the latter handle the other types of processing the server has to provide; performing user Authentication.

Apache Core consists of several small components that manage the essential implementation of Apache web server main functionality. Figure 5.1 depicts the main elements of the Apache Core and how they interact with each other to provide the required functionalities.

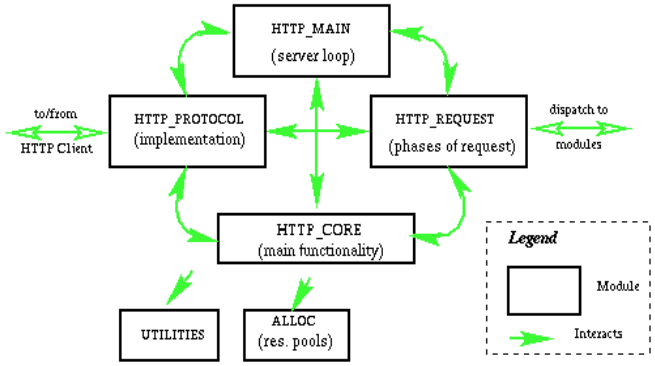


Figure 5.1: Apache HTTP Server Core Components

These components are explained as follows:

- $http\_protocol.c$: it handles the routines that communicate directly with the client by using the HTTP protocol. It handles the socket connections that the client uses to connect to the server and it manages data transfer.

- $http\_main.c$: it is in charge for the startup of the server, contains the main server loop that waits for and accepts connections, and manages timeouts.

- $http\_request.c$: it deals with the flow of request processing, passing control to the modules as needed in the right order, and error handling.

- $http\_core.c$: it implements serves documents.

- $alloc.c$: it manages allocating resource pools and keeps track of them.

- $http\_config.c$: it provides tasks for other utilities such as reading configuration files, managing the information collected from those files, and aid for virtual hosts.

However, on the other hand, Apache Modules provide additional functionality that extend and implement the functionality of the Apache HTTP server. Basically, each module is connected to the Apache core, $HTTP\_REQUEST$ component, and provides separated functionality. Thus, no module can progress without sending the information to the $HTTP\_REQUEST$ component which in turn checks and handles errors.

5.5 Measuring System Vulnerability

The vulnerability datasets of Apache HTTP server 1.3.x and 2.2.x releases are from NVD. NVD is maintained by National Institute of Standers and Technology and sponsored by the department of Home Land Security. Therefore, the unreliability of the datasets is unlikely. The collected discovered vulnerabilities were of a period from 1999 to 2012. Some of those vulnerabilities were found to be applicable to two different platforms namely MS-Windows and Linux. Only few were found to specific to Mac OS.

Table 5.2: Apache Releases Vulnerabilities

Release	Number of Versions	New Vulnerabilities	Inherited Vulnerabilities	Total
1.3.x	47	51	932	983
2.2.x	23	23	415	438

5.5.1 Number of Known Reported Vulnerabilities

The known reported vulnerabilities of Apache web server 1.3 and 2.2 versions are presented in Figure 5.2. In version 1.3, the numbers of the new vulnerabilities, which have been introduced in this version, are 18 whereas the number of the inherited vulnerabilities, which have been inherited from another releases and versions, are 11. However, the total numbers of the new and inherited vulnerabilities in version 2.2 are 11 for the two of them. As it has been observed in Figure 5.2, the numbers of the new vulnerabilities of version 1.3 are larger than its counterpart, version 2.2. However, the inherited numbers of vulnerabilities are the same.

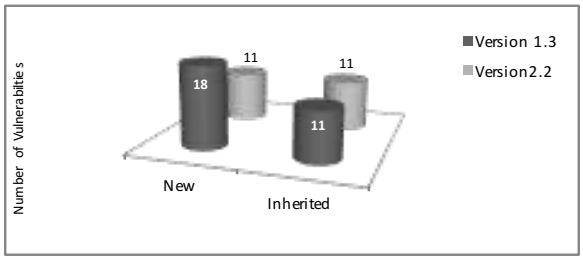


Figure 5.2: Apache HTTP Server Version 1.3 & 2.2 Vulnerabilities

Table 5.2 presents values of number of versions, new and inherited vulnerabilities, and over all vulnerabilities for two Apache web server releases 1.3.x and 2.2.x. The table shows that the numbers of the new and inherited vulnerabilities of Apache 1.3.x release are more than half of 2.2.x release. Moreover, the numbers of the inherited vulnerabilities of 1.3.x release and the 2.2.x release are way larger than the numbers of new vulnerabilities. This could be attributed to the share of code between successive versions and releases as it has been stated by Kim et al in [78].

The number of the new and inherited vulnerabilities for all versions of Apache release 1.3.x and 2.2.x are presented in Figures 5.3 and 5.4 respectively. There are forty seven versions for

Apache 1.3.x release while twenty three for Apache 2.2.x release. It has been noted that the first two versions for both releases have more new vulnerabilities than the other versions. This is due to the fact that the data available are for the beta versions. As it can be seen, the successive versions had significantly less new vulnerability. However, it has been noted that half of the versions in both releases did not have any new vulnerabilities. This could be due the truth that their market shares are not as high as the others. Woo et al in [40], have pointed out that higher market share is one of the most important factors impacting the effort spent in exploring and exploiting potential vulnerabilities. Surprisingly, it has been observed that some of the inherited vulnerabilities affect only a specific version(s) in one release but not the others.

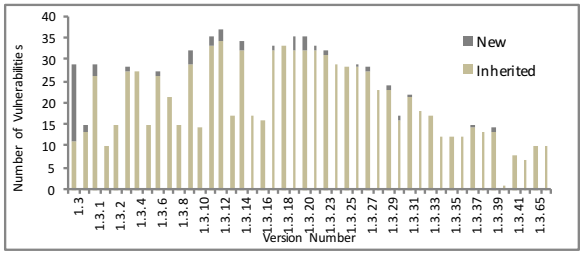


Figure 5.3: Apache HTTP Server Release 1.3.x Vulnerabilities

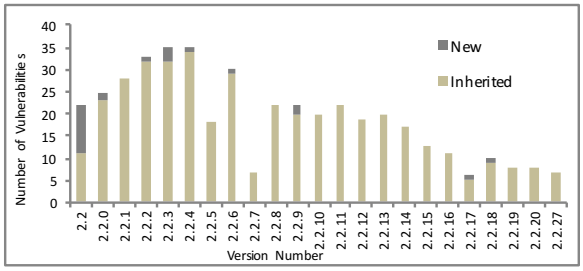


Figure 5.4: Apache HTTP Server Release 2.2.x Vulnerabilities

Table 5.3 presents known vulnerability density VKD for two versions 1.3 and 2.2 of Apache web server from two different releases, 1.3.x and 2.2.x. Since the Apache web server is an open-source project, the source code for the two chosen versions was downloaded from Apache HTTP server archive download site [79]. The code size for the two selected versions was determined using SLOCCount tool; a software metrics tool for counting physical source lines of code [80].

The major fractions of the source code of the two versions are C based and this was resolved using the SLOCCount tool. This is considered helpful since the attack surface metrics is capable

Table 5.3: New Known Vulnerabilities Density

Application	Ksloc	Known,vulnerabilities	VKD
Apache 1.3	50,712	18	0.000354946
Apache 2.2	222,029	11	4.95E-05

of measuring a C base code as well as Java base code. However, the known vulnerabilities column gives a recent count of the vulnerabilities found since the release date. Despite having smaller size than version 2.2, version 1.3 had more number of known vulnerabilities as well as known vulnerabilities density. Vulnerability Based on Severity and Type

5.5.2 Vulnerability Based on Severity and Type

Figure 5.5 shows the number of vulnerabilities based on their severity for Apache 1.3 and 2.2 versions. While the former had eight vulnerabilities of high severity, the latter did have any. Both versions had the same number of vulnerabilities of medium severity. However, vulnerabilities of low severity were not found in version 1.3 whereas only one was found in version 2.2. Overall, version 1.3 had more fractions of high and medium severity vulnerabilities. This presents a significant risk and could lead to problems such as denial of service (DoS) attack, exposure of sensitive information, etc.

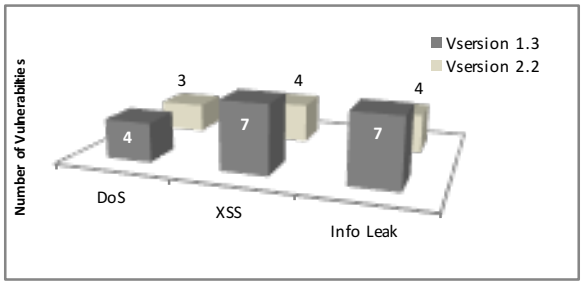


Figure 5.5: Apache HTTP Server version 1.3 & 2.2 Vulnerability Severity

Vulnerabilities based on type for Apache HTTP server version 1.3 and 2.2 are presented in Figure 5.6. Majority of the vulnerabilities for both versions are of types Cross-Site Scripting (XSS) and Information leak. The existence of more vulnerabilities of the type XSS is due to the fact that

all web servers, application servers, and web application environments are susceptible to it. Even though DoS vulnerabilities are the most type found in all Apache HTTP version and releases, version 1.3 and 2.2 had lesser numbers of them. However, among the discovered vulnerabilities four DoS vulnerabilities in version 1.3 are high severity vulnerabilities, while three DoS vulnerabilities in version 2.2 are medium severity vulnerabilities.

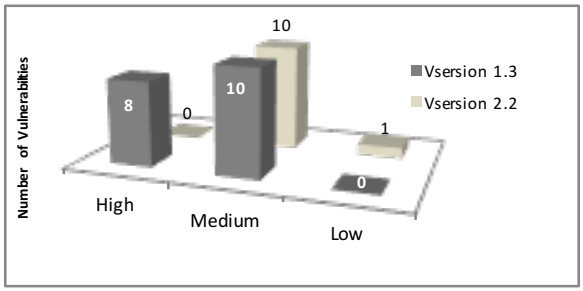


Figure 5.6: Apache HTTP Server version 2.2 Vulnerability Type

The data collected for Apache HTTP server versions, 1.3 and 2.2, and releases 1.3.x and 2.2.x suggest that the 1.3 version and its release had more vulnerabilities, vulnerability density, and severity than the version 2.2 and its entire release. Based on this fact, we expect the attack surface, which we are going to calculate in the next section, for both versions to follow the same patterns.

5.6 Measuring System Attack Surface

In this section, we will measure the attack surface along the method dimension for Apache HTP Server 1.3 and 2.2 versions. Choosing the method dimension in particular is due to the fact that attackers can only exploit vulnerability in a method if they can invoke that method either directly or indirectly. However, measuring the attack surface requires looking at the code base and finding all places which could be part of the attack surface. By finding such places what is needed next is classifying each one of them into an attack class. The code bases of the two chosen versions were obtained from [75].

The entry and exit points along the method dimension have been defined using cflow tool. It analyzes a code base written in C programming language and produces a graph charting dependencies between various functions [81]. Figure 5.7 shows some sample cflow output from Apache

HTTP Server version 1.3.0. Before counting each method, functions reachable from the main(), in the attack surface, the privilege level and access right level of the entry and exit points needed to be determined and marked.

```
1  API_EXPORT() < at http_protocol.c:1927>:
2    ap_table_get()
3    strncasecmp()
4    strcasecmp()
5    strchr()
6    parse_byterange() <*support code ... */int parse_byterange (char
*range,long clength,long *start,long *end) at http_protocol.c:85>
7    ap_pstrdup()
```

Figure 5.7: Fractional cflow output from Apache Web Server 1.3.0

Firstly, the privilege of a process runs on a UNIX system can change from one level to the other using one of uid-setting system call such as setuid. If a process drops it privilege, then every method that are invoked before the drop of the privilege will have that level of privilege whereas every method invoked after the drop of the privilege will have the new privilege. Thus, a method might be counted multiple times, once per each privilege level. The methods in the Apache HTTP Server version 1.3 and 2.2 run with a root and nobody, www-data, manager.sys, or apache privilege.

Secondly, in order to the access right of a method to be determined, we need to define the code base where the authentication is presented. Based on that, every method invoked before user authentication performed has unauthenticated access right while those methods invoked after user authentication take place has authenticated access right.

As it can be seen in Table 5.4, a value to each privilege level and access right level is assigned based on our knowledge of Apache HTTP server and Linux (Ubuntu). The number of the direct entry point and the direct exit point are shown in Table 5.5. Nonetheless, using Tables 5.4 and 5.5, the attackability along the method direction considering the effect of the resources Damage Potential-Effort Ratio was calculated as shown in Table 5.6. The measure of Apache 1.3 attack surface along the method dimension is 235.3 whereas the measure of Apache 2.3 along the same dimension is 207.3.

Table 5.4: Numeric Values of Privilege & Access Rights

Method Privilege	Value	Access Rights	Value
Root	5	root	5
apache or (www-data or nobody)	3	authenticated	3
authenticated	3	unauthenticated	1
manager.sys	3	anonymous	1

Table 5.5: Direct Entry and Exit Points

Apache 1.3			
Privilege	**Access Right**	**Direct Entry Point**	**Direct Exit Point**
root	root	4	7
root	authenticated	11	6
apache	authenticated	10	18
apache	unauthenticated	9	15
apache	anonymous	2	17
manager.sys	authenticated	1	12
Apache 2.2			
Privilege	**Access Right**	**Direct Entry Point**	**Direct Exit Point**
root	root	1	5
root	authenticated	9	14
apache	authenticated	7	16
apache	unauthenticated	5	23
apache	anonymous	3	14

Table 5.6: Attackability Measurements

Code Base	Total Contribution of the Methods	Total
Apache 1.3	11 (5/5) + 17 (5/3) + 28 (3/3) + 24 (3/1) + 19 (3/1) + 13 (3/1)=	35.3
Apache 2.2	6 (5/5) + 23 (5/3) + 23 (3/3) + 28 (3/1) + 7 (3/1) =	207.3

5.7 Observations

The total numbers of vulnerabilities as well as vulnerability density for Apache HTTP Server version 1.3 and its attack surface (Entry/Exit points along the method dimension) have been found to be more than those in Apache HTTP Server version 2.2. This result suggests that the number of vulnerability and vulnerability density seems to have correlation with that attack surface. Besides, knowing the type and severity of vulnerabilities could guide and simplify the process of the attack surface measurement. Moreover, knowing the attack points of a system could help developers to verify that those parts of the code have no exploitable vulnerabilities. This result also suggests that a framework that combines these two metrics together is visible and should be further investigated. However, further experiments with different software systems should be conducted so the result of this study can be further consolidated.

5.8 Conclusion & Future work

This chapter examines the relationship between the attack surface and vulnerability density metrics. It also investigates the visibility of how the two metrics can be used to guide each other.

There is an increase demand from developers, managers and users on controlling activities such as resource allocation, program planning, risk assessment and product and service selection. Software Security metrics are supposed to fulfill this need. However, as making decisions relies heavily on the accuracy of these metrics, validating and comparing them to each other is considered as a very significant step.

Attack surface measures system attackability, probability of a system to be attacked, of a software system while vulnerability density metric measures the density of the known vulnerability of a system per thousand line of code. Both metrics have been applied to major software systems and they have been found useful. However, both have not been compared to one another.

In this study, we have measured the attackability and the vulnerability density of two versions of the Apache HTTP server 1.3 and 2.2 using the two metrics. Apache web server has been chosen because of its highest marked share among its competitors and availability of source code. Results

have shown that attackability and vulnerability of the two selected systems imply the same idea. That is, the attackability along the method dimension and the vulnerability density measures of the Apache HTTP Server version 1.3 are greater than the version 2.2. It has been also observed that not only the number of vulnerability or vulnerability density that is related to attack surface but also the type and the severity of vulnerabilities are of degree of importance.

While only new vulnerabilities were considered in this study, considering the inherited vulnerabilities could also be helpful. Moreover, identifying the vulnerable component of a software system by applying a metric that can define the most vulnerable modules can be very useful for both vulnerability density metric as well as attack surface metric.

Chapter 6

Assessing The Risk of Vulnerabilities Exploitation

This chapter introduces a novel measure, Structural Severity, which is based on software properties, namely attack entry points, vulnerability location, the presence of the dangerous system calls, and reachability analysis. These properties represent metrics that can be objectively derived from attack surface analysis, vulnerability analysis, and exploitation analysis. To illustrate the proposed approach, 25 reported vulnerabilities of Apache HTTP server and 86 reported vulnerabilities of Linux Kernel have been examined at the source code level. The results show that the proposed approach, which uses more detailed information, can objectively measure the risk of vulnerability exploitability and results can be different from the CVSS base scores.

6.1 Introduction

Security of the computer systems and networks depend on the security of software running on them. Many of the attacks on computer systems and networks are due to the fact that attackers try to potentially compromise these systems by exploiting vulnerabilities present in the software. Recent trends have shown that newly discovered vulnerability still continues to be significant and so does the number of attacks (+37 million attacks in 2012-2013) [82]. Studies have also shown that the time gap between the vulnerability public disclosure and the release of an automated exploit is getting smaller [3]. Therefore, assessing the risk of exploitation associated with software vulnerabilities is needed to aid decision-makers to prioritize among vulnerabilities, allocate resources, and choose between alternatives.

A security metric is a quantifiable measurement that indicates the level of security for an attribute of a system [67]. Security metrics give a way to prioritize threats and vulnerabilities by considering the risks they pose to information assets based on quantitative or qualitative measures.

The metrics proposed include: vulnerability density, attack surface, flaw severity and severity-to-complexity, security scoring vector for web applications, and CVSS metrics, etc. Each of them is based on specific perspectives and assumptions and measures different attributes of software security. They are intended to objectively help decision makers in resource allocation, program planning, risk assessment, and product and service selection.

CVSS is the de facto standard that is currently used to measure the severity of individual vulnerabilities [4]. CVSS Base Score measures severity based on the exploitability (the ease of exploiting a vulnerability) and impact (the effect of exploitation). The base score is rounded to one decimal place and it is set to zero if the impact is equal to zero regardless of the formula. Exploitability is assessed based on three metrics: Access Vector (AV), Authentication (AU), and Access Complexity (AC) as shown by the following:

$$Exploitability = 20 \times AV \times AU \times AC$$

The AV metric reflects how the vulnerability is exploited in terms of local (L), adjacent network (A), or network (N). The AC metric 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 AU metric counts the number of times an attacker must authenticate to reach a target (in order to exploit a vulnerability) in terms of Multiple (M), Single (S), or None (N).

However, CVSS exploitability metrics have the following limitations. First, they assign static subjective numbers to the metrics based on expert knowledge regardless of the type of vulnerability. For instance, they assign AV 0.395 if the vulnerability requires local access, 0.646 if the vulnerability requires adjacent network access, and 1 if the vulnerability requires global network access. Second, two of its factors (AV and AU) have the same value for almost all vulnerabilities [83]. Third, there is no formal procedure for evaluating the third factor (AC) [4]. Consequently, it is unclear if CVSS considers the software structure and properties as a factor.

On the other hand, the impact sub-score measures how vulnerability will directly affect an IT asset as the degree of losses in Confidentiality (IC), Integrity (II), and Availability (IA). The impact sub-scores are all assessed in terms of None (N), Partial (P), or Complete (C) by security experts

and assigned one of the mentioned qualitative letter grades. Thus, there is a need for an approach that can take into account detailed information about the access complexity and the impact factors for less subjective exploitability measures.

6.1.1 Problem Description and Research Motivation

The risk of vulnerability exploitability is dependent upon two factors: Exploitability and Impact, as expressed in:

$$Exploitationrisk = f(Exploitability \times Impact)$$

The first factor, Exploitability, is the likelihood that a potential vulnerability can be successfully exploited. This factor concerns the question Is the vulnerability exploitable? The other factor, Impact, measures the losses that occur given a successful exploitation. This factor is related to the question Which is the most exploitable vulnerability?

There are challenges in assessing the exploitation risk that are needed to be addressed. In this work, we look into the major challenges in assessing the exploitability factor, and identify key questions that need to be addressed. Then, we discuss the challenges in assessing the impact factor and also identify a key question that is required to be tackled. To guide our analysis of the problem and to ensure that our solution is guided by recognized criteria, we used two guides from National Institute of Standards and Technology (NIST) namely: the Risk Management Guide for Information Technology Systems [84] and Directions in Security Metrics Research Report [67].

Exploitability Factor

There are three main challenges when it comes to assessing the exploitability factor. They are explained as follows.

a) Exploitability Estimators:

The main challenge is determining the estimator (attribute) to be used in assessing exploitability factor. There are a number of estimators such as: existence of exploits, existence of patches, number of vulnerabilities, number of attack entry points, exploit kits in the black market, and proof of concept. Which one of those makes a good estimator of exploitability? To answer this

Table 6.1: Limitations of exploitability estimators

Exploitability Estimator	Limitations
1. Existence of an exploit	The data required to measure this attribute is not always available.
2. Existence of a patch	The existence of a patch does not tell whether it has been applied or not because the study (Arbaugh et al. 2000) shows patches application depends on the administrators behavior.
3.Number of vulnerabilities and number of attack entry points	These estimators are not informative as they do not specify which vulnerability is exploitable and rather they estimate the exploitability of the whole system.
4. Exploit Kit in the black market	This approach requires a vulnerability intelligence provider, as the information about the attacks and tools are dynamic in nature. Moreover, if the vulnerability right now is not used by a tool or it is not a target of an attack, it does not mean that it is going to be so continually.
5. Proof of concept	It is difficult and time consuming to generate a reliable exploit.

question, lets look at every one of those estimators. As is summarized in Table 6.1, each of the estimators assesses exploitability from a specific perspective and has its own limitations. Thus, a good estimator should have the following characteristics: it should be measurable, it should not be expensive to obtain, and it should be obtained objectively.

b) Evaluation of the Measures:

The next challenge is evaluating the exploitability estimators. Obtaining the measures can be achieved by using security expert opinions, one of the estimators mentioned above, or the software structure. Relying on expert opinions leads to subjectivity and can potentially hinder the accuracy of the assessment. The challenge is: How can we reduce subjectivity and minimize human involvement in exploitation assessment? On the other hand, the data regarding reported vulnerabilities or exploits is not always available especially for newly released software. Thus, the question is: How can the risk of exploitation be assessed in the absence of historical data? The alternative could be using the software code. However, in that case the challenges would be: What type of features can be used as an indicator of vulnerability exploitability? How can these features be objectively derived? These questions are addressed in this paper.

c) Level of Assessment:

A further challenge is deciding on the level of assessment that the exportability should be assessed at. Should we assess exploitability for each individual vulnerability or the whole software? Assessing the vulnerability exploitability at the software level is not informative as it assumes that all vulnerabilities have the same risk of exploitation. This is unrealistic as different vulnerabilities have different risk of exploitation. Thus, assessing the exploitation risk at the individual vulnerability level needs to be done first. Thereafter, we can combine risk value of each individual vulnerability and get the total risk of exploitation for the whole system.

Impact Factor

Estimating the impact factor is challenging because it is context dependent. For instance, a shutdown of a mission critical server may be more severe than a print server. Manadhata and Wing [69] introduced two types of impacts: Technical impact (e.g., privilege elevation) and Business impact (e.g., monetary loss). While the latter depends on the mission and the priority of the given context, the former can be estimated at a function level. For the technical impact the key question is: What estimators should be used to determine the technical impact? The answer could be any one of these estimators: function privilege, existence of dangerous system calls, presence of authentication, or presence of exploit mitigation. These estimators are explained as follows:

- The higher the privilege of a function (e.g. root) the more damage is going to be.
- Having a dangerous system call (e.g. open) in an entry point can help the attackers escalate their privileges and hence can cause more damage to the compromised system.
- An authentication procedure along the attack path to the vulnerable location makes exploiting a vulnerability harder and hence reducing the impact.
- Exploit mitigation reduces the impact of a vulnerability exploitation.

6.1.2 Research Objectives and Contribution

The main objective of this research is to propose an approach that can address the challenging questions discussed in the problem description. We mainly focus on reducing subjectivity in assessing vulnerability exploitation risk. To that end, we base our analysis on software properties. These properties represent measures that can objectively be derived from the source code using the attack surface analysis, the vulnerability analysis, and the exploitation analysis irrespective of the level of security knowledge one has. Our proposed measure evaluates vulnerability exploitability based on the presence of a function call connecting attack surface entry points to the vulnerability location within the software under consideration. If such a call exists, we estimate how likely an entry point is going to be used in an attack based on the presence of the dangerous system calls. The dangerous system calls paradigm has been considered because they allow attackers to escalate a function privilege and hence cause more damage.

To determine the effectiveness of the proposed approach, the Apache HTTP server and Linux Kernel were selected as a case study. The two software systems have been selected because of: their rich history of publicly available vulnerabilities, availability of an integrated repository (which enables us to map vulnerabilities to their location in the source code), availability of the source code (which enables us to collect the measures of the proposed metrics), and their diversity in size and functionalities.

The chapter is organized as follows. Section 2 presents the background. In section 3, the key steps of our approach are introduced. In section 4, the evaluation and the results of the proposed approach are shown. In the following section, some observations and threats to validity are presented. Section 6 presents the related work. Finally, concluding comments are given along with the issues that need further research.

6.2 Related Concepts and Terminology

In this section, we describe the concepts and terminology related to the vulnerability exploitation risk. This section provides a brief introduction to software vulnerabilities, the attack surface

metric used in the proposed approach, the system dependence graph used for reachability analysis, and the exploit databases used to validate the analysis.

6.2.1 Software Vulnerabilities

A software vulnerability is defined as a defect in software systems that presents a considerable security risk [2]. A subset of the security related defects, vulnerabilities, are to be discovered and become known eventually [2]. The vulnerabilities are thus a subset of the defects that are security related. The finders of the vulnerabilities disclose them to the public using some of the common reporting mechanisms available in the field. The databases for the vulnerabilities are maintained by several organizations such as National Vulnerability Database (NVD) [16], Open Source Vulnerability Database (OSVDB) [17], etc., as well as the vendors of the software. Vulnerabilities are assigned a unique identifier using MITRE Common Vulnerability and Exposure (CVE) service.

6.2.2 Attack Surface Metric

A systems attack surface is the subset of the systems resources that are used by an attacker to attack the system [69]. This includes the functions that serve as the entry points and exit points, the set of channels (e.g. sockets), and the set of untrusted data items [69]. Entry and Exit points can be used by an attacker to either send or receive data from the system. Channels are the means that are used by an attacker to connect to the system. Untrusted data items are the data that an attacker can either send or receive from the system. In this study, we consider the entry points and the potential paths leading to the functions containing the vulnerabilities.

6.2.3 System Dependence Graph

A system dependence graph (SDG) introduced by [85] is an extension of the Program dependence graph (PDG) [86, 85, 87]. PDG is a directed graph representation of a program, where vertices represent program points (e.g., assignment statements, call-sites, variables, control predicates) that occur in the program and edges represent different kinds of control or data dependencies. An SDG consists of interconnected PDGs (one per procedure in the program) and extends the control and data dependencies with interprocedural dependencies. An interprocedural control-

dependence edge connects procedure call sites to the entry points of the called procedure and an interprocedural data-dependence edge represents the flow of data between actual parameters and formal parameters (and return values). A system dependence graph can be used for purposes such as code optimization, reverse engineering, program testing, program slicing, software quality assurance, and software safety analysis. This work employs SDG to determine the call sequence among functions in a source code which may lead to a potential exploitation.

6.2.4 Exploit Database (EDB)

EDB records exploits and vulnerable software [22]. It is used by penetration testers, vulnerability researchers, and security professionals. It reports vulnerabilities for which there is at least a proof-of-concept exploit. EDB is considered as a regulated market for the exploits. EDB contains around 24075 exploits as the time of writing this paper. Most of its data are derived from Metasploit Framework, a tool for creating and executing exploit code against a target machine. It provides a search utility that uses a CVE number to find vulnerabilities that have an exploit.

6.3 Approach

Figure 6.1 shows an overview of our approach for assessing vulnerability exploitability risk. It is based on combining the attack surface analysis, the vulnerability analysis, and the exploitation analysis.

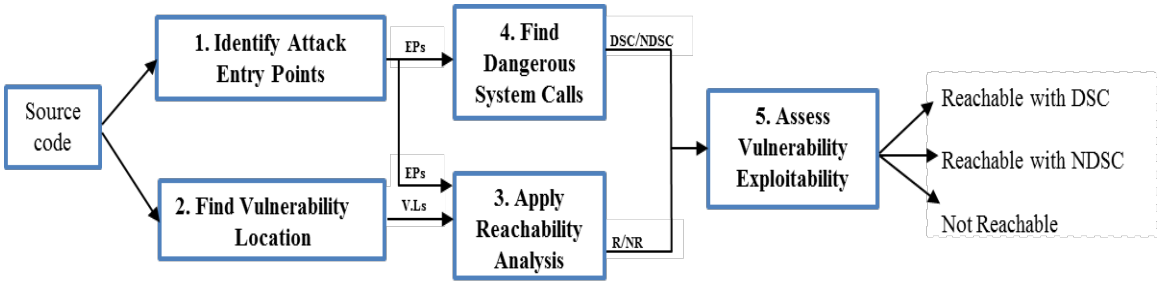


Figure 6.1: Overview of the proposed approach

We start by identifying the attack entry points and a vulnerability location from the source code. Next, we apply the reachability analysis to the identified attack entry points and the vulnerability

Table 6.2: Structural Severity Measure

Metric 1	Metric 2	Measure
Reachability	**Dangerous System Call (DSC)**	**Structural Severity**
Not Reachable	No DSC exist	Low
Not Reachable	DSC exist	Low
Reachable	No DSC exist	Medium
Reachable	DSC exist	High

location. The outcome of this step is to determine whether a vulnerability is reachable from any of the identified attack entry points (R) or it is not reachable (NR). After that, we verify whether any of the identified attack entry points has dangerous system calls. The result of this step is either the attack entry point has a dangerous system call (DSC) or it does not have a dangerous system call (NDSC). Then, we assess vulnerability exploitability based on the results of the steps three and four. The outcome of this step is a vulnerability is reachable with DSC, reachable with NDSC, or not reachable. Based on the result of this step, the risk of exploitation of a given vulnerability is assigned as shown in Table 6.2. The following subsections will explain these steps in detail.

6.3.1 Identify Attack Entry Points

We define the attack entry points using the systems attack surface entry point framework proposed by [69]. Entry points are the functions that an attacker uses to send data to the system. In this paper, we used only the entry points because they are the main target of malicious attacks. A function is a direct entry point if it receives data directly from the environment; read function defined in unistd.h in C library is an example [69]. Figure 6.2 shows how the entry points are identified. The required steps are as follows:

a) Obtain the source code

b) Identify all functions that receive data from the user environment (C/C++ Library functions)

c) Verify whether these functions are used by any of the user functions

- Identify all functions called by the main function using cflow
- Use python script to verify whether any of these functions has one of the C/C++ input functions

- If you find any, then consider that function as an EP
- Get the list of all entry points

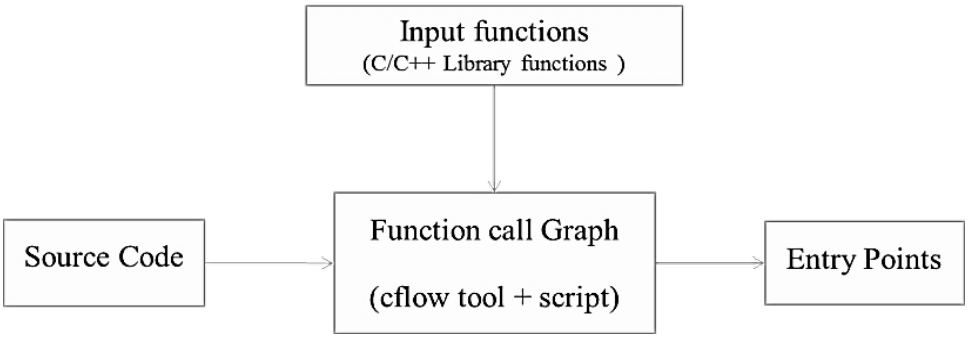


Figure 6.2: Identification of attack entry points

6.3.2 Find Vulnerability Location

The vulnerability location can be found by looking at the report in the vulnerability database or by using a static code analyzer such as Splint [88]. Figure 6.3 shows the way vulnerabilities are mapped to their locations.

a) Mapping vulnerabilities using databases:

- From the vulnerability database, identify the Vulnerability.
- From the Bug Repository and Version Archive:
  - Identify the vulnerable Version (e.g. Apache 1.3.0)
  - Identify files by mapping Bug ID to CVE number
  - Identify the vulnerable function
- For the selected vulnerable version:
  - Search inside all folders in the main folder and find all .c files and store them in a list
  - From the list, select the .c files that contain the vulnerabilities
  - For every selected file find the vulnerable function(s)

b) Mapping vulnerabilities using a static code analyzer Finding a vulnerability location can also be determined by using static code analyzers when the report does not finalize such information. The static code analyzers are tools that are used to find common bugs or vulnerabilities in the code base without the need to execute the code. Splint (Secure Programming Lint) is an example. It is a tool that uses static analysis to detect vulnerabilities in programs [88]. However, in this paper the vulnerability report was used to find the location of the vulnerability. The use of static tools were left as a future work.

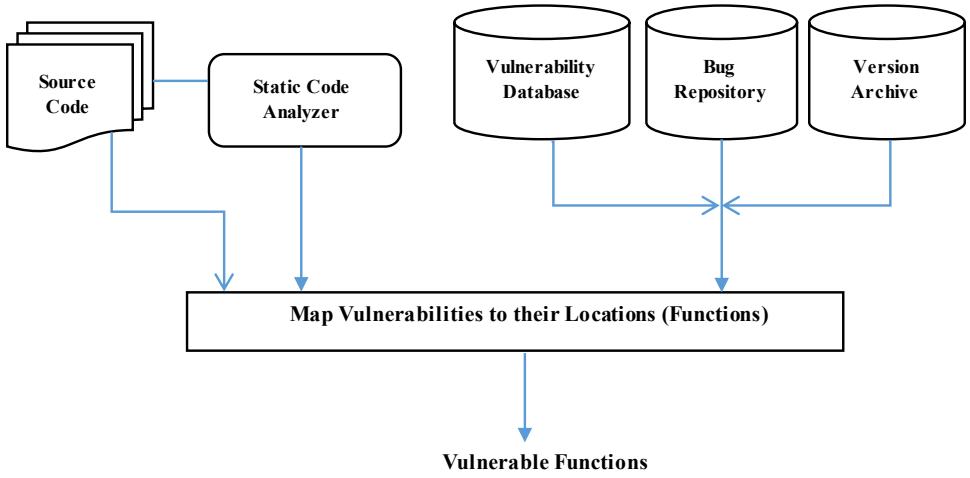


Figure 6.3: Vulnerability location identification

6.3.3 Apply Reachability Analysis

Reachability means the analysis of the call relationships between the entry points and the vulnerability locations (vulnerable functions). We employed a system dependence graph (SDG) proposed by Horwitz et al. [85] to determine the calls from an entry point function to a vulnerability location (vulnerable function). There are many tools available for automatically generating SDG from the source code. We have selected Understand. Understand is a static analysis tool for maintaining, measuring, and analyzing critical or large source code. We have used the Understand tool [89], to generate this graph from the source code. This tool has been chosen because it is user-friendly and it has a good set of APIs that allows interaction with programming languages such as Python, Perl, and C++. Figure 6.4 shows how reachability analysis is used.

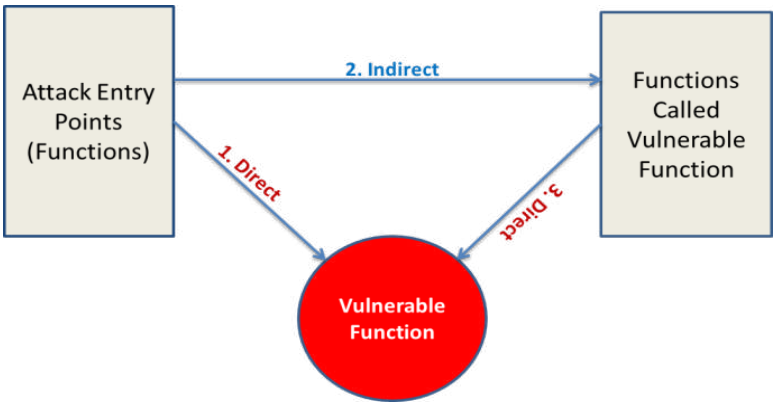


Figure 6.4: Reachability analysis

After identifying the entry points and the vulnerable functions, the reachability analysis was conducted as follows:

- Generate a (called by) graph that captures all functions that call the vulnerable function directly or indirectly
- Verify whether any of these functions is
  - directly calling the vulnerable function and an entry point
  - indirectly calling the vulnerable function and an entry point
- If so, the vulnerable function is recognized as reachable
- If not, the vulnerable function is recognized as not reachable.

6.3.4 Find Dangerous System Calls

System calls are the entry points to privileged kernel operations [90]. Calling a system call from a user function in a program can violate the least privilege principle. Massimo et al. [91] define DSCs as specific system calls that can be used to take complete control of the system, cause a denial of service, or other malicious acts. These system calls have been identified and classified into four levels of threats. Level one allows full control of the system while level two is used for denial of service attack. On the other hand, level three is used for disrupting the invoking process

Table 6.3: Dangerous system calls

Threat Level	Dangerous System Calls
1. Full control of the system	chmod, fchmod, chown, fchown, lchown, execve, mount, rename, open, link,
2. Denial of service	umount, mkdir, rmdir, umount2, ioctl, nfsservctl, truncate, ftruncate, quotactl, dup, dup2, flock, fork, kill, iopl, reboot, ioperm, clone
3. Used for subverting the invoking process	read, write, close, chdir, lseek, dup, fcntl, umask, chroot, select, fsync, fchdir, llseek, newselect, readv, writev, poll, pread, pwrite, sendfile, putpmsg, utime
4. Harmless	getpid, getppid, getuid, getgid, geteuid, getegid, acct, getpgrp, sgetmask, getrlimit, getrusage, getgroups, getpriority, sched getscheduler, sched getparam, sched get

and level four is considered harmless. Table 6.3 shows some of the DSCs as classified by Massimo et al. [91]. There are 22 system calls of the threat level one and 32 of the threat level two.

We used the existence of DSCs as an estimate of the impact of exploitation. This is because having DSCs in a user function helps the attackers escalate their privileges and hence causing more damage to the compromised system. As the attack entry points are the main entries for the attacker to the system, we verify whether these points have DSCs. We do that by looking at the list of the identified attack entry points and verify whether each entry point contains DSCs by using a python script. If any DSCs are found, we annotate that function as an entry point with DSCs.

6.3.5 Assess Vulnerability Exploitability

An individual vulnerability exploitability risk is assessed using the measures obtained from step 1 to 4. Thus, a vulnerability can be classified as one of the following:

- Reachable with Dangerous System Calls.
- Reachable with No Dangerous System Calls.

Table 6.4: Apache HTTP server and Linux Kernel vulnerabilities dataset

Software	Exploit Exist	No Exploit Exist	Total Each
Apache HTTP Server	11	14	25
Linux Kernel	64	22	86
Total All	75	36	111

- Not reachable.

6.4 Evaluation and Results

This section presents the results of the evaluation of the proposed measure and assesses its performance. To implement our approach, 111 vulnerabilities of Apache HTTP server and Linux Kernel have been chosen as shown in Table 6.4. These vulnerabilities have been collected from the National Vulnerability Database [16]. They have been selected based on the availability of information about their locations and their exploits. It should be noted that we have considered all Apache HTTP server vulnerabilities that have an exploit. Out of the 73 Linux Kernel vulnerabilities that have an exploit, we only examined 64 vulnerabilities. This is because seven out of the nine vulnerabilities depends on a configuration property and not a user input. In addition, we could not find information about the location of the other two vulnerabilities. Further, as the main focus of this research is to evaluate the capability of the proposed measure, we have only considered a few of the vulnerabilities that have no exploit. We also tried to select the vulnerabilities that are at least three or four years old, so that their lack of exploit is not due to their recent discovery.

Our approach in assessing software vulnerability exploitability is based on the sequential steps that have been discussed in section 3. As applying the steps of our approach are the same for Apache HTTP server and Linux Kernel, we are only including the detailed explanation for Apache HTTP server here. However, the final results of the investigation are going to be presented in two separate tables.

6.4.1 Define Attack Entry Points

Identifying the attack surface entry points requires looking at the code base and finding all entry points which could be a part of the attack surface. After finding such points, we then classify

Table 6.5: Entry points and Dangerous System Calls

File Name	Entry Points		Dangerous System Calls	
	C/C++ Input Functions	Entry Point Function	Denial of Service	Full Control
protocol.c	read	ap_get_mime_headers	-	-
	read	ap_read_request	-	-
	get	ap_set_sub_req_protocol	-	-
http_filter.c	scanf	ap_http_filter	-	-
http_core.c	scanf	register_hooks	-	-
mod_usertrack.c	get	spot_cookie	-	-
mod_proxy.c	scanf	register_hooks	-	link
mod_rewrite.c	scanf	register_hooks,	-	-
		hookuri2file		
mod_proxy_ajp.c	read	ap_proxy_ajp_request	-	-
mod_deflate.c	read packets	deflate_out_filter	dup	-
mod_proxy_http.c	read, get, gethostbyname	Stream_reqbody_cl,	-	-
		Ap_proxy_http_reques,	dup	-
		proxy_http_handler	dup	-
mod_proxy_ftp.c	read packets	Proxy_send_dir_filter	dup	-
mod_proxy_balancer.c	get	balance_handler	dup	-
mod_status.c	get	Ststus_handler	-	-
core.c	get	include_config	-	-
main.c	get	main,	exit	open
mod_autoindex	read	index_directory	dup	open
proxy_http.c	read socket	ap_proxy_http_handler	dup	-
proxy_util.c	read buffer	ap_proxy_send_fb	-	-
mod_include.c	getc	get_tag	-	-
mod_imap.c	getline	imap_handler	-	open
mod_negotiation.c	get	do_negotiation	dup	open

each one of them into an attack class. The code bases of the chosen version were obtained from [79].

The entry points along the method dimension were defined using the cflow tool and a python script. The cflow tool analyzes a code base written in C programming language and produces a graph charting dependencies among various functions [81]. Using the python script, we identified the entry points for the whole system and selected the ones that are related to the chosen vulnerabilities location. These entry points are shown in Table 6.5.

CVE-2012-0053
protocol.c in,the Apache HTTP Server 2.2.x through 2.2.21 does not properly,restrict header information during construction of Bad Request (aka 400),error documents, which allows remote attackers to obtain the values of HTTP Only,cookies via vectors involving a (1) long or (2) malformed header in,conjunction with a crafted web script.

6.4.2 Find Vulnerability Location

The vulnerability location was determined by mapping information from NVD, Apache Http server bug repository [92] and Apache SVN archive databases [93]. Vulnerabilities are either located in one of the entry points or are located in a function that is called by the entry points directly or indirectly. By performing the following steps, we identified the location of the vulnerabilities.

a) From the NVD, we selected the vulnerability. The following report shows one of the selected vulnerabilities. As can be seen, the vulnerable file and Apache version can be easily determined from the following report description.

b) Using Bugzilla, we mapped the CVE number to the Bug id. Bugzilla is a bug- or issue-tracking system. Figure 6.5 shows the mapping step.

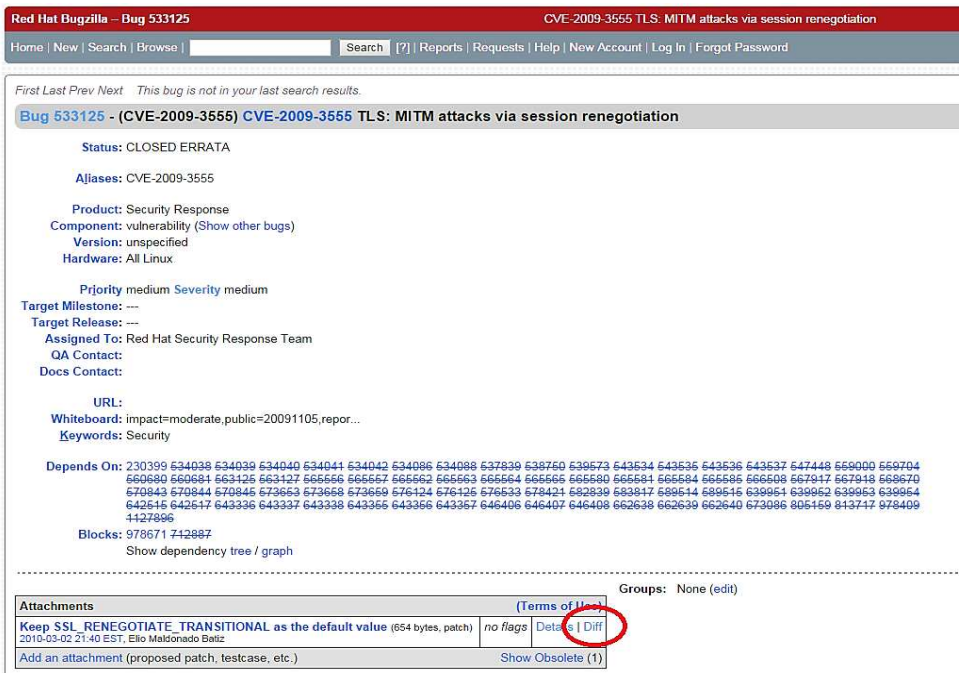


Figure 6.5: Mapping from CVE number to Bug ID

c) From Bugzilla, we accessed the Apache SVN to determine the vulnerable code. Apache SVN is a software used to maintain the current and historical versions of files: such as source code, web pages, and documentation. We used diff link, which is shown circled at the bottom of Figure 6.5, to access the source code and identify the vulnerable code (function). Basically, the diff link shows the difference between before the code has been changed and after the code has been modified. The diff page is shown in Figure 6.6 where the vulnerable function ap_get_mime_headers_core has been circled with red color. The diff link shows the code for a bug fix, which corresponds to a vulnerability fix. Using the (—) symbol indicates the part of the code that has been removed whereas (+++) symbol shows the code that has been added.

Figure 6.6: Identifying vulnerable code (Function)

Once the vulnerable function name is identified, we look at the source code and identify the .c file name that contains the function and annotate its location. We followed the same steps to identify the location of the remaining vulnerabilities. Table 6.6 shows the locations of the chosen vulnerabilities. It has been noticed that some of the vulnerabilities have been located in more than one function.

Table 6.6: Vulnerabilities locations (functions)

Vulnerability	File Name	Vulnerable Function Name
CVE-2012-0053	protocol.c	ap_get_mime_headers_core
CVE-2011-4415	util.c	ap_pregsub
CVE-2011-4317	mod_proxy.c	proxy_handler
	mod_rewrite.c	hook_fixup
CVE-2011-3192	byterange_filter.c	parse_byterange
CVE-2010-0434	protocol.c	clone_headers_no_body
CVE-2010-0408	mod_proxy_ajp.c	ap_proxy_ajp_request
CVE-2009-1891	mod_deflate.c	deflate_out_filter
CVE-2009-1890	mod_proxy_http.c	Stream_reqbody_cl
CVE-2008-2939	mod_proxy_ftp.c	Proxy_send_dir_filter
CVE-2007-6420	mod_proxy_balancer.c	balance_handler
CVE-2006-5752	mod_status.c	Ststus_handler
CVE-2009-1195	config.c	process_resource_config_nofnmatch, ap_process_config_tree
CVE-2007-5000	mod_imagemap.c	menu_header
CVE-2007-4465	mod_autoindex	index_directory
CVE-2010-0010	Proxy_util.c	ap_proxy_send_fb
CVE-2004-0940	mod_include.c	get_tag
CVE-2007-6388	mod_status.c	status_handler
CVE-2005-3352	mod_imap.c	read_quoted
CVE-2004-0488	ssl_util.c	ssl_util_uuencode_binary
CVE-2008-0455	mod_negotiation.c	make_variant_list
CVE-1999-0107	http_request.c	process_request_internal
CVE-2004-0493	protocol.c	ap_get_mime_headers
CVE-2006-3747	mod_rewrite.c	apply_rewrite_rule
CVE-2013-1896	mod_dav.c	dav_method_merge
CVE-2006-3918	http_protocol.c	get_canned_error_string

6.4.3 Apply Reachability Analysis

Once the vulnerable functions and the entry points were identified, reachability analysis can be achieved using the Understand tool. This tool automatically generates a graph contains a chain of all functions that can call the selected function. This graph is known as Called By graph. It shows what calls the selected function. Each line connecting an entity is read as x is called by y. It should be noted that this graph is read from the bottom up. When x is called by y, x is in the bottom and y is in the top. We have added to the generated graph the entry point (EP) label, and vulnerability CVE number to make the graph easily viewable asier. Shown the reachability analysis for all selected vulnerabilities can make the length of the chapter more than it should be, therefore only five vulnerabilities were selected. Those vulnerabilities are representative of the common trends among the studied vulnerabilities.

a) CVE-2012-0053 Looking at Table 6.6, it can be verified that this vulnerability is located in the function ap_get_mime_headers_core which in turn resides in the file protocol.c. To determine whether this vulnerability is reachable from an entry point, we performed the following:

- Generate a called by graph that captures all functions that call the vulnerable function directly or indirectly. Figure 6.7 shows the generated graph with the function name at the top of the rectangle and .c file name in the bottom.

- Verify whether any of these functions is an entry point. Using the entry points in Table 6.5, the attacker can have an access to the vulnerability by two ways:

  - **Directly**: The protocol.c component has two entry points namely: ap_get_mime_headers and ap_read_request that directly can call the vulnerable function. These two entry points are located in the protocol.c file where the vulnerable function is located. None of these entry points have a dangerous system call.

  - **Indirectly**: The component core.c has an entry point function register_hooks that indirectly can call the vulnerable function in the component protocol.c from multiple paths. This entry point does not have a dangerous system call.

As a result, it can be concluded that there is a call relationship between the entry points and the vulnerable functions. The vulnerable function is also indirectly reachable from an entry point located in a different component. Neither of these entry points has DSCs.

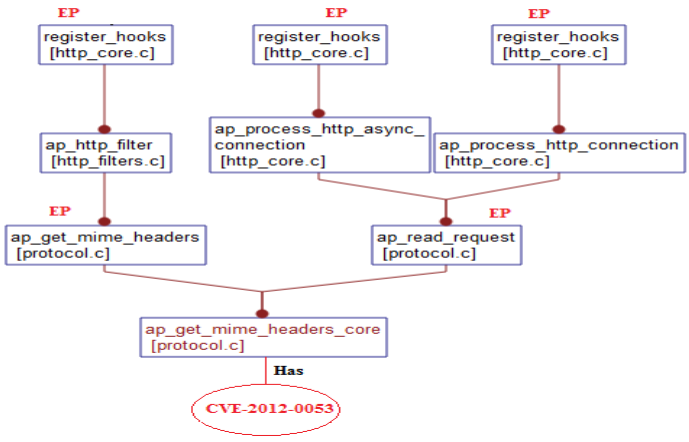


Figure 6.7: Direct and indirect call sequences from the EP to the vulnerable function

b) CVE-2004-0488 Looking at Table 6.6, it can be shown that this vulnerability is located in the function ssl_utill_uuencode_binary which lives in the file ssl_util.c. To determine whether this vulnerability is reachable from an entry point, we performed the following:

- Generate a called by graph that captures all functions that call this function directly or indirectly. Figure 6.8 shows the generated. graph.
- Verify whether any of these functions is an entry point. Using the entry points in Table 6.5, the attacker has no way of manipulating this vulnerability because there are no entry points found in any of these functions including the vulnerable function

As a result, it can be concluded that this vulnerable function is not reachable from an entry point.

c) CVE-2007-4465 From Table 6.6, it can be verified that this vulnerability is located in the function index_directory which resides in the mod_autoindex.c. To determine whether this vulnerability is reachable from an entry point, we performed the following:

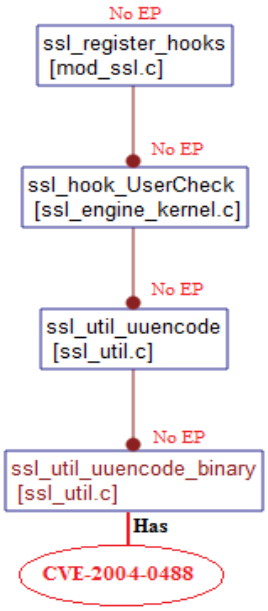


Figure 6.8: No call sequence that reaches the vulnerable function in sll_util.c.

- Generate a called by graph that captures all functions that call this function directly or indirectly. Figure 6.9 shows the generated graph.

- - Verify whether any of these functions is an entry point. Using the entry points in Table 6.5, it has been found that the vulnerable function itself is also an entry point. Besides, register_hooks is an entry point too. Thus, the attacker has an access to this vulnerability by invoking the vulnerable function directly. This entry point has a DSC.

As a result, it can be concluded that this vulnerability is reachable by directly calling the vulnerable function. Besides, this function has a DSC.

d) CVE-2010-0434 From Table 6.6, it can be confirmed that this vulnerability is located in the function clone_headers_no_body which resides in the file protocol.c. To determine whether this vulnerability is reachable from an entry point, we performed the following:

- Generate a called by graph that captures all functions that call the vulnerable function directly or indirectly. Figure 6.10 shows the generated graph.

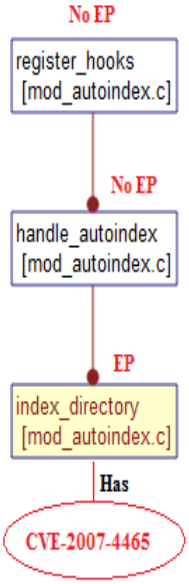


Figure 6.9: Direct call sequence from the EP to the vulnerable function in mod_autoindex.c

- Verify whether any of these functions is an entry point. Using the entry points in Table 6.5, it has been found that the attacker can reach the vulnerable function in two ways:
  - **Directly**: The component protocol.c has an entry point: ap_set_sub_req_protocol that can directly call the vulnerable function. This entry point has no DSCs.
  - **Indirectly**: The component mod_rewrite.c has two entry points namely: register_hooks and hookuri2file. Thus, the attacker has an access to this vulnerability from multiple paths as shown in Figure 6.10. This entry point has NDSCs.

As a result, it can be concluded that this vulnerability is reachable by directly and indirectly calling the vulnerable function from multiple paths. Besides, these functions have NDSCs.

e) CVE-2009-1195 From Table 6.6, it can be decided that this vulnerability is located into two functions namely: ap_process_cnfig_tree and process_resource_config_nofnmatch which both live in the file config.c. To determine whether this vulnerability is reachable from an entry point, we performed the following:

- Generate a called by graph that captures all functions that call the vulnerable function directly or indirectly. Figure 6.11 shows the generated graph.

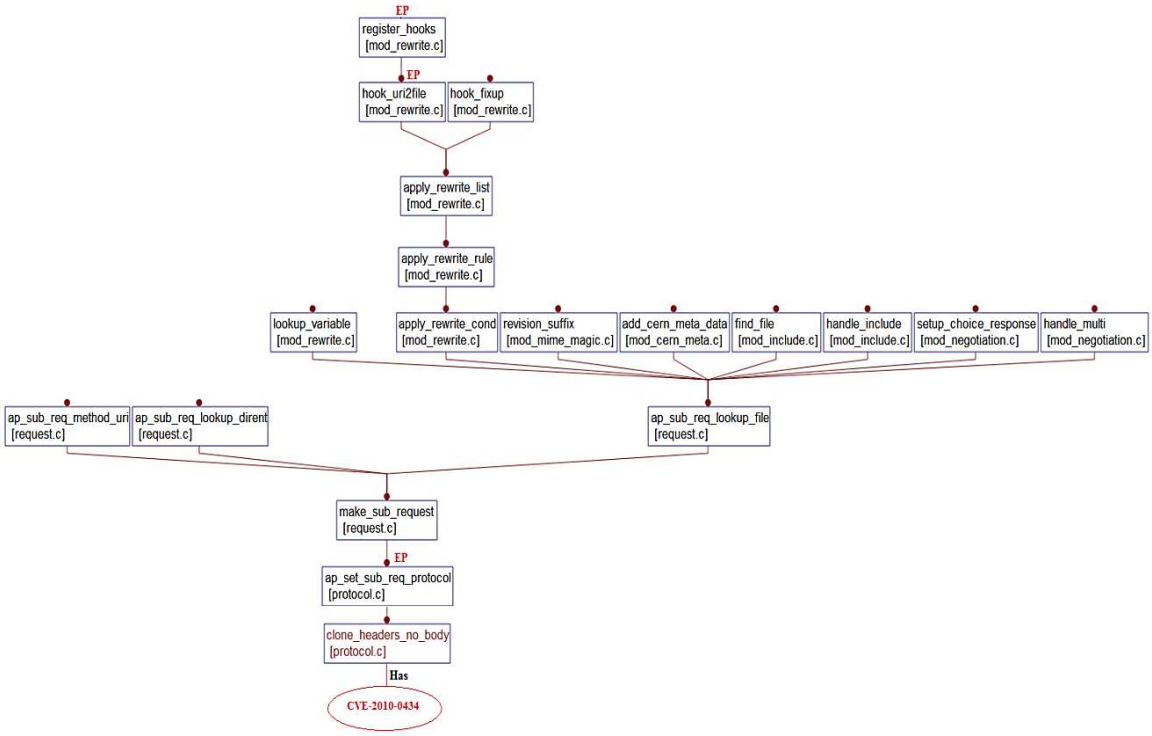


Figure 6.10: Direct and indirect call sequences from the EP to the vulnerable function in prorotocol.c.

- Verify whether any of these functions is an entry point. Using the entry points in Table 6.5, the attacker has an access to this vulnerability by invoking the vulnerable function in two ways:

  - **Indirectly**: The main.c component has an entry point namely: main that can indirectly call the vulnerable function process_resource_config_nofmatch throughout the path: main, ap_read_config, and ap_process_resource_config as shown on the left side of the graph. The entry points have a DSC.

  - **Directly**: There are two ways the vulnerable function can be directly called. First, the main.c component has an entry points namely: ap_process_resource_config that can directly call the vulnerable function. Second, throughout the same entry point main, the vulnerable function ap_process_resource_config_tree can be directly called as shown on the right side of the graph. The entry points have a DSC.

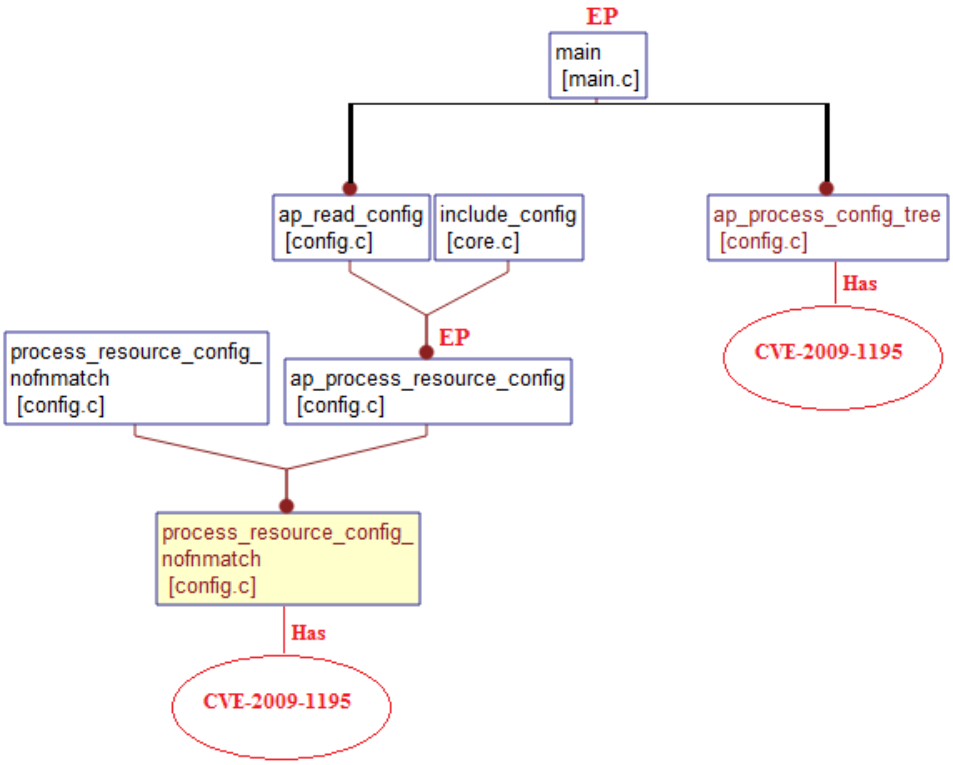


Figure 6.11: Direct and indirect call sequences from the EP to the vulnerable function in config.c

As a result, it can be concluded that there is a call relationship between the entry points in the component main.c and the two vulnerable functions in the component config.c. The vulnerable function is also indirectly reachable from an entry point located in a different component. The entry point has DSCs.

6.4.4 Find Dangerous System Calls

For the identified attack entry points, we have checked whether an entry point has dangerous system calls. These dangerous system calls are shown in Table 6.5.

6.4.5 Assessing Vulnerability Exploitability

After tracing the call sequence from the entry points to the vulnerability location and identifying the DSCs, the individual vulnerability Structural Severity can be evaluated using the results included in Table 6.7. Looking at Table 6.7, a vulnerability is either:

1. Reachable with Dangerous System Calls.

2. Reachable with No Dangerous System Calls.

3. Not reachable.

Table 6.7 gives the structural severity measures of the Apache HTTP Server vulnerabilities using the proposed method. Every vulnerability has its own location and the availability of the path from the entry point next to it. If a path from an entry point to a vulnerable function is found, the vulnerability considered to be reachable; otherwise, it is considered not reachable. Besides, the existence of the DSCs for every entry point has been specified.

6.4.6 Assigning a Vulnerability Structural Severity Value

The Structural Severity is measured using the reachability and DSCs metrics. It is then assigned one of the three values: high, medium, and low as described in section 3. Based on these three values, the chosen vulnerabilities were assessed and compared to the CVSS overall severity scores. In following subsections, we will show the Structural Severity values for the selected vulnerabilities of Apache HTTP server and Linux Kernel datasets respectively. It should be noted that we have used the exploit database (EDB), Open Source vulnerability database, and the Meta exploit Database **??** for gathering the exploits for the chosen vulnerabilities. Besides, the authors are not aware of a database that reports the impact of vulnerabilities exploitation and hence comparing those factors is not visible. The abbreviations used in the tables are explained as follows:

- DSC: Dangerous System Call.
- NDSC: No Dangerous System Call.
- AC: Access Complexity Metric. Its values are Low (L), Medium (M), and High (H).
- AV: Access Vector Metric. Its values are Network (N), Local (L), and Adjacent Network (A).
- AU: Authentication Metric. Its values are None (N), Single (S), and Multiple (M).
- The CVSS impact metrics use C for Confidentiality, I for Integrity, A for Availability. These metrics takes the following values: None (N), Partial (P), and Complete (C).

Table 6.7: The Obtained metrics of the proposed measure

Vulnerability	Function name	Path from Entry Points	Dangerous System Call	Reachability
CVE-2004-0488	ssl_util_uuencode_binary	no path	NDSC	NR
CVE-2004-0940	get_tag	get_tag	NDSC	R
CVE-2005-3352	read_quoted	imap_handler	DSC	R
CVE-2006-5752	Ststus_handler	ststus_handler	NDSC	R
CVE-2007-6420	balance_handler	balance_handler	DSC	R
CVE-2007-5000	menu_header	no path	NDSC	NR
CVE-2007-4465	index_directory	index_directory	DSC	R
CVE-2007-6388	status_handler	status_handler	NDSC	R
CVE-2008-2939	proxy_send_dir_filter	proxy_send_dir_filter	DSC	R
CVE-2008-0455	make_variant_list	do_negotiation→ store_variant_list	DSC	R
CVE-2009-1891	deflate_out_filter	deflate_out_filter	NDSC	R
CVE-2009-1195	process_resource_config _nofnmatch,	main→ap_read_config→ap_process_resources_config	DSC	R
		include_config→ap_process_resources_config	NDSC	
	ap_process_config_tree	Main	DSC	
CVE-2009-1890	stream_reqbody_cl	stream_reqbody_cl	NDSC	R
		ap_proxy_http_request→proxy_ http_handler	DSC	
CVE-2010-0010	ap_proxy_send_fb	ap_proxy_send_fb	NDSC	R
		ap_proxy_http_handler (Proxy_http.c)	DSC	
CVE-2010-0434	clone_headers_no_body	ap_set_sub_req_protocol	NDSC	R
CVE-2010-0408	ap_proxy_ajp_request	ap_proxy_ajp_request	NDSC	R
CVE-2011-4415	ap_pregsub	register_hooks→spot_cookie	NDSC	R
CVE-2011-3192	parse_byterange	register_hooks→ap_ byterange_filter	NDSC	R
CVE-2011-4317	proxy_handler	register_hooks	DSC	R
	hook_fixup	register_hooks	NDSC	
CVE-2012-0053	ap_get_mime_headers_c ore	register_hooks→ap_http_filter →p_get_mime_headers	NDSC	R
CVE-1999-0107	process_request_internal	Standalone_main→makechild→ childmain→ap_process_request	NDSC	R
CVE-2004-0493	ap_get_mime_headers	register_hooks→ap_http_filter	NDSC	R
CVE-2006-3747	apply_rewrite_rule	register_hooks→hook_urlfile→ apply_rewrite_list	NDSC	R
CVE-2013-1896	dav_method_merge	register_hooks→dav_handelr	DSC	R
CVE-2006-3918	get_canned_error_string	ap_read_request→ap_send_erro r_response	NDSC	R
R: Reachable, NR: Not reachable, DSC: Dangerous System Call, NDSC: No Dangerous System Call				

- SS: Total Subscore

Apache HTTP Server

Table 6.8 shows the Structural Severity and the CVSS Metrics values for the Apache HTTP server dataset. The table also shows the availability of an exploit for every vulnerability. The following has been observed from Table 6.8:

- Two out of the 25 vulnerabilities are not reachable and have no exploits.
- Thirteen out of the 25 vulnerabilities are reachable with no existing exploits. More than half of these vulnerabilities have DSCs and hence have been assigned a high Structural Severity value.
- The remaining 11 vulnerabilities are reachable and have exploits. Four out of them have DSCs and thus have been assigned a high Structural Severity value.
- The majority of the vulnerabilities have been assigned high exploitability value by both metrics.
- More than half of the vulnerabilities have been found to have a low impact value by both metrics (CVSS impact SS, and Structural Severity DSC). This could be attributed to the type of software.
- More than half of the vulnerabilities have been assigned a medium severity value by the two metrics.
- It should be noted that all the vulnerabilities require no Authentication whereas only three vulnerabilities require a Local Network Access Vector (L). The majority of the Access Complexity metric values are low and medium and these values have no significant effect on the overall CVSS exploitability subscore.
- The exploitability total subscore is significantly changed only when the Access Complexity metric value is high. Nonetheless, there are only two vulnerabilities that have a high Access Complexity metric value.

- Out of the 11 vulnerabilities that have an exploit, only two vulnerabilities have been assigned a low CVSS exploitability subscore.

It should be notated that the vulnerabilities in Table 6.8 have been grouped into five groups based on their similarity with regard to their reachability, DSCs and availability of exploit. This demonstrates their relationship. The following explains these groups.

a) CVE-2004-0488 and CVE-2007-5000:

According to the CVSS metrics, the CVE-2004-0488 have a low Access Complexity value and a high overall severity, whereas CVE-2007-5000 have a low Access Complexity and a medium overall severity. Considering the network accessibility factor is useful but not sufficient. In order for a vulnerability to be exploited it first has to be reachable regardless of the access conditions. However, based on the software structure analysis, we did not find any call relationship between the vulnerable function and any of the entry point functions. Additionally, no exploit was found for these vulnerabilities in any of the exploit databases. Thus, based on the proposed metrics, they have been assigned a low Structural Severity. Having just a potential vulnerability does not mean that it is going to be exploited.

b) CVE-2008-0455, CVE-2009-1890, CVE-2010-0010, and CVE-2013-1896:

As stated by the CVSS metrics, the CVE-2008-0455, CVE-2010-0010, and CVE-2013-1896 have a medium Access Complexity value and a medium overall severity, whereas CVE-2009-1890 have a medium Access Complexity and a high overall severity. Based on software structure analysis, we found the two vulnerabilities to be reachable and to have a DSC. Additionally, there exists an exploit for these vulnerabilities. However, based on the proposed metrics, they have been assigned a high Structural Severity.

c) CVE-2012-0053, CVE-2004-0940, and CVE-2011-3192:

Based on the CVSS metrics, both CVE-2012-0053 and 2004-0940 have a medium Access Complexity value and a medium overall severity. On the other hand, CVE-2011-3192 have a low Access Complexity value and a high overall severity. Based on software structure analysis, we found these three vulnerabilities to be reachable and to have no DSC. Additionally, there exist an

Table 6.8: The Obtained Structural Severity Metrics Compared to CVSS Metrics of Apache HTTP Server Dataset

No	Vulnerability	Structural Severity			CVSS									Exploit Existence
		Reachability	DSC	Severity	Exploitability				Impact				Severity	
					AC	AV	AU	SS	C	I	A	SS		
1	CVE-2004-0488	NR	NDSC	**L**	L	N	NR	10	P	P	P	6.4	**H**	NEE
2	CVE-2007-5000	NR	NDSC	**L**	M	N	NR	8.6	N	P	N	2.9	**M**	NEE
3	CVE-2008-0455	R	DSC	**H**	M	N	NR	8.6	N	P	N	2.9	**M**	EE
4	CVE-2009-1890	R	DSC	**H**	M	N	NR	8.6	N	N	C	6.9	**H**	EE
5	CVE-2010-0010	R	DSC	**H**	M	N	NR	8.6	P	P	P	6.4	**M**	EE
6	CVE-2013-1896	R	DSC	**H**	M	N	NR	8.6	N	N	P	2.9	**M**	EE
7	CVE-2004-0940	R	NDSC	**M**	M	L	NR	3.4	C	C	C	10	**M**	EE
8	CVE-2011-3192	R	NDSC	**M**	L	N	NR	10	N	N	C	6.9	**H**	EE
9	CVE-2012-0053	R	NDSC	**M**	M	N	NR	8.6	P	N	N	2.9	**M**	EE
10	CVE-1999-0107	R	NDSC	**M**	L	N	NR	10	N	N	P	2.9	**M**	EE
11	CVE-2004-0493	R	NDSC	**M**	L	N	NR	10	N	P	P	4.9	**M**	EE
12	CVE-2006-3747	R	NDSC	**M**	H	N	NR	4.9	C	C	C	10	**H**	EE
13	CVE-2006-3918	R	NDSC	**M**	M	N	NR	8.6	N	P	N	2.9	**M**	EE
14	CVE-2006-5752	R	NDSC	**M**	M	N	NR	8.6	N	P	N	2.9	**M**	NEE
15	CVE-2007-6388	R	NDSC	**M**	M	N	NR	8.6	N	P	N	2.9	**M**	NEE
16	CVE-2009-1891	R	NDSC	**M**	M	N	NR	8.6	N	N	C	6.9	**H**	NEE
17	CVE-2010-0434	R	NDSC	**M**	M	N	NR	8.6	P	N	N	2.9	**M**	NEE
18	CVE-2010-0408	R	NDSC	**M**	L	N	NR	10	N	N	P	2.9	**M**	NEE
19	CVE-2011-4415	R	NDSC	**M**	H	L	NR	1.9	N	N	P	2.9	**L**	NEE
20	CVE-2005-3352	R	DSC	**H**	M	N	NR	8.6	N	P	N	2.9	**M**	NEE
21	CVE-2007-6420	R	DSC	**H**	M	N	NR	8.6	N	P	N	2.9	**M**	NEE
22	CVE-2007-4465	R	DSC	**H**	M	N	NR	8.6	N	P	N	2.9	**M**	NEE
23	CVE-2008-2939	R	DSC	**H**	M	N	NR	8.6	N	P	N	2.9	**M**	NEE
24	CVE-2009-1195	R	DSC	**H**	L	L	NR	3.9	N	N	C	6.9	**M**	NEE
25	CVE-2011-4317	R	DSC	**H**	M	N	NR	8.6	N	P	N	2.9	**M**	NEE

EE: Exploit Exist; **NEE**: No Exploit Exist; **SS**: Subscore

exploit for these vulnerabilities. Thus, based on the proposed metrics, the vulnerabilities have been assigned a medium Structural Severity.

d) CVE-2006-5752, CVE-2007-6388, CVE-2009-1891, CVE-2010-0434, CVE-2010-0408, CVE-2011-4415:

Consistent with the CVSS metrics, the following vulnerabilities: CVE-2006-5752, CVE-2007-6388, and CVE-2010-0434 have a medium Access Complexity and a medium overall severity. Besides, CVE-2009-1891 hve a medium Access Complexity and a high overall severity. Additionally, CVE-2010-0408 have a low Access Complexity and a medium overall severity. Moreover, CVE-2011-4415 have a medium Access Complexity and a high overall severity. However, based on software structure analysis, we found that all of the vulnerabilities are reachable and have no DSC. Furthermore, no exploit was found for these vulnerabilities. Based on the proposed metrics, these vulnerabilities have been assigned a medium Structural Severity.

e) CVE-2005-3352, CVE-2007-6420, CVE-2007-4465, CVE-2008-2939, CVE-2009-1195, CVE-2010-0010, and CVE-2011-4317:

According to the CVSS metrics, all the above vulnerabilities have a medium Access Complexity value except for CVE-2009-1195 which have a low Access Complexity sub-score. Besides, all of them had a medium overall severity. However, based on software structure analysis, we found that all of them are reachable and have a DSC. Additionally, there was no exploit found for these vulnerabilities. Based on the proposed metrics, these vulnerabilities have been assigned a high Structural Severity.

Linux Kernel

Table 6.9 in the Appendix shows the Structural Severity and the CVSS Metrics values for Linux Kernel. The table also shows the availability of an exploit for every vulnerability. The following can be observed from Table 6.9.

- Three out of the 86 vulnerabilities are not reachable and have no exploits. Two of them have DSCs.

- Nineteen out of the 86 vulnerabilities are reachable with no exploit exist for them. More than half of these vulnerabilities have DSCs and hence have been assigned a high Structural Severity value.

- All 64 vulnerabilities that have an exploit have been found to be reachable. More than half of these vulnerabilities have a DSC and thus have been assigned a high Structural Severity value.

- The majority of the vulnerabilities have been assigned a high exploitability value by the Structural Severity Reachability metric. On the other hand, most of the vulnerabilities have been assigned a low exploitability value by the CVSSS exploitability metrics. This could be attributed to the fact that most of the vulnerabilities are accessed locally.

- The majority of the vulnerabilities have been found to have a high impact value by both metrics (CVSS impact SS and Structural Severity DSC).

- More than half of the vulnerabilities have been assigned a high severity value by the Structural Severity metrics whereas only a few have been assigned a low severity value. This can be explained by the previous two observations. On the other hand, the CVSS severity metrics do not have noticeable variation among their values except for a slightly larger number of medium severity values.

- It should be noted that most of the vulnerabilities require no Authentication except for two, which required only a Single System Authentication (SS). The majority of the vulnerabilities requires a Local Network Access Vector (L). However, there are 14 vulnerabilities that require a Network Access Vector (N) and there are only two vulnerabilities that require an Adjacent Network Access Vector (A). The majority of the Access Complexity metric values is low.

- Neither the low nor the medium Access Complexity metric values have a significant effect on the overall CVSS exploitability subscore. The exploitability total subscore is significantly

changed only when the Access Complexity metric value is high. Nonetheless, there are only five vulnerabilities that have a high Access Complexity value.

- Out of the 64 vulnerabilities that have an exploit, seven vulnerabilities have been assigned a high CVSS exploitability subscores and 12 vulnerabilities have been assigned low CVSS exploitability subscores.

It should be notated that the vulnerabilities in Table 6.9 have been grouped into two groups based on the availability of exploit. As the number of the vulnerabilities of Linux Kernel is large, we will only discuss some noteworthy vulnerabilities.

a) CVE-2003-0462, CVE-2004-1235, CVE-2006-2629, CVE-2006-5757, and CVE-2010-4258:

According to the CVSS metrics, these vulnerabilities have a high Access Complexity value and hence low overall exploitability subscore. Four of them have a medium overall severity and only one (CVE-2003-0462) has a low overall severity. Based on software structure analysis, we found that these vulnerabilities are reachable and three of them have a DSC. The other two vulnerabilities (CVE-2003-0462 and CVE-2006-5757) do not have a DSC. Additionally, there exists an exploit for these vulnerabilities. Thus, based on the proposed metrics, the three vulnerabilities that have a DSC have been assigned a high Structural Severity value while the other two vulnerabilities that do not have DSCs have been assigned a medium Structural Severity value.

b) CVE-2003-0619, CVE-2004-1137, CVE-2006-2444, CVE-2007-1357, CVE-2009-0065, CVE-2009-3613 and CVE-2010-1173:

According to the CVSS metrics, these vulnerabilities have a network Access Vector value and thus have been assigned a high overall exploitability subscore. Six vulnerabilities have been assigned a high overall severity and only one vulnerability, CVE-2003-0619, that has been assigned a medium overall severity. Based on the software structure analysis, these vulnerabilities have been found reachable and three of them have a DSC. The other four vulnerabilities (CVE-2003-0619, CVE-2009-0065, CVE-2009-3613, and CVE-2010-1173) do not have a DSC. Besides, there exists an exploit for these vulnerabilities. Thus, based on the proposed metrics, the three vulnerabilities that have DSCs have been assigned a high Structural Severity value while the other four vulnerabilities that do not have DSCs have been assigned a medium Structural Severity value.

Table 6.9: The Obtained Structural Severity Metrics Compared to CVSS Metrics of Linux Kernel Dataset

No	Vulnerability	Structural Severity			CVSS									Exploit Existence
		Reachability	DSC	Severity	Exploitability				Impact				Severity	
					AC	AV	AU	SS	C	I	A	SS		
1	CVE-2002-0499	R	DSC	**H**	L	L	NR	3.9	N	P	N	2.9	**L**	EE
2	CVE-2003-0462	R	NDSC	**M**	H	L	NR	1.9	N	N	P	2.9	**L**	EE
3	CVE-2003-0619	R	NDSC	**M**	L	N	NR	**10**	N	N	P	2.9	**M**	EE
4	CVE-2003-0961	R	DSC	**H**	L	L	NR	3.9	C	C	C	10	**H**	EE
5	CVE-.2003-0985	R	NDSC	**M**	L	L	NR	3.9	C	C	C	10	**H**	EE
6	CVE-2004-0424	R	DSC	**H**	L	L	NR	3.9	C	C	C	10	**H**	EE
7	CVE-2004-1016	R	DSC	**H**	L	L	NR	3.9	N	N	P	2.9	**L**	EE
8	CVE-2004-1073	R	DSC	**H**	L	L	NR	3.9	P	N	N	2.9	**L**	EE
9	CVE-2004-1137	R	DSC	**H**	L	N	NR	**10**	C	C	C	10	**H**	EE
10	CVE-2004-1235	R	DSC	**H**	H	L	NR	1.9	C	C	C	10	**M**	EE
11	CVE-2004-1333	R	DSC	**H**	L	L	NR	3.9	N	N	P	2.9	**L**	EE
12	CVE-2005-0736	R	DSC	**H**	L	L	NR	3.9	N	P	N	2.9	**L**	EE
13	CVE-2005-0750	R	DSC	**H**	L	L	NR	3.9	C	C	C	10	**H**	EE
14	CVE-2005-1263	R	DSC	**H**	L	L	NR	3.9	C	C	C	10	**H**	EE
15	CVE-2005-1589	R	DSC	**H**	L	L	NR	3.9	C	C	C	10	**H**	EE
16	CVE-2005-2709	R	NDSC	**M**	L	L	NR	3.9	P	P	P	6.4	**M**	EE
17	CVE-2005-2973	R	NDSC	**M**	L	L	NR	3.9	P	N	N	2.9	**L**	EE
18	CVE-2005-3257	R	DSC	**H**	L	L	NR	3.9	P	P	P	6.4	**M**	EE
19	CVE-2005-3807	R	DSC	**H**	L	L	NR	3.9	N	N	C	6.9	**M**	EE
20	CVE-2005-3808	R	DSC	**H**	L	L	NR	3.9	N	N	C	6.9	**M**	EE
21	CVE-2005-3857	R	NDSC	**M**	L	L	NR	3.9	N	N	C	6.9	**M**	EE
22	CVE-2006-2444	R	DSC	**H**	L	N	NR	**10**	N	N	C	6.9	**H**	EE
23	CVE-2006-2451	R	DSC	**H**	L	L	NR	3.9	P	P	P	6.4	**M**	EE
24	CVE-2006-2629	R	DSC	**H**	H	L	NR	1.9	N	N	C	6.9	**M**	EE
25	CVE-2006-5757	R	NDSC	**M**	H	L	NR	1.9	N	N	P	2.9	**L**	EE
26	CVE-2007-1000	R	NDSC	**M**	L	L	NR	3.9	N	N	C	10	**H**	EE
27	CVE-2007-1357	R	DSC	**H**	L	N	NR	**10**	N	N	C	6.9	**H**	EE
28	CVE-2007-1388	R	NDSC	**M**	M	L	**SS**	2.7	N	N	C	6.9	**M**	EE
29	CVE-2007-1730	R	DSC	**H**	L	L	NR	3.9	C	N	C	9.2	**M**	EE
30	CVE-2007-1861	R	DSC	**H**	L	L	NR	3.9	N	N	C	6.9	**M**	EE
31	CVE-2008-4113	R	DSC	**H**	M	L	NR	3.9	C	N	N	6.9	**M**	EE
32	CVE-2008-4302	R	NDSC	**M**	L	L	NR	3.9	N	N	C	6.9	**M**	EE
33	CVE-2008-5713	R	NDSC	**M**	L	L	NR	3.9	N	N	C	6.9	**M**	EE
34	CVE-2009-0065	R	NDSC	**M**	L	A	NR	**10**	C	C	C	10	**H**	EE
35	CVE-2009-0676	R	NDSC	**M**	L	L	NR	3.9	P	P	N	2.9	**L**	EE
36	CVE-2009-0746	R	DSC	**H**	L	L	NR	3.9	N	N	C	6.9	**M**	EE
37	CVE-2009-1337	R	DSC	**H**	M	L	NR	3.4	P	P	P	6.4	**M**	EE
38	CVE-2009-1897	R	DSC	**H**	M	L	NR	3.4	C	C	C	10	**M**	EE

c) CVE-2005-0124, CVE-2005-2492, and CVE-2005-2500:

According to the CVSS metrics, these vulnerabilities have a low Access Complexity value. Two of them (CVE-2005-0124 and CVE-2005-2492) have been assigned a low overall exploitability subscore and a low overall severity and this is because they are locally accessed. On the other hand, CVE-2005-2500 has been assigned a high overall exploitability subscore and a high overall severity and this is because it is accessed via a network. Based on software structure analysis, these vulnerabilities have been found not reachable and one of them, CVE-2005-0124, does not have a DSC. Moreover, no exploit was found for these vulnerabilities in any of the exploit databases. Hence, based on the proposed metrics, they have been assigned a low Structural Severity value.

d) CVE-2002-0499, CVE-2003-0462, CVE-2004-1016, CVE-2004-1073, CVE-2004-1333, CVE-2005-0736, CVE-2005-2973, CVE-2006-5757, CVE-2009-0676, CVE-2009-1961, CVE-2010-1636, and CVE-2010-4073:

According to the CVSS metrics, these vulnerabilities have a local Access Vector value and thus have been assigned a low overall exploitability subscore and a low overall severity. All of these vulnerabilities require no authentication. Besides, eight out of these vulnerabilities have a low Access Complexity value. Out of the remaining four vulnerabilities only two vulnerabilities (CVE-2003-0462 and CVE-2006-5757) have a high Access Complexity value whereas the other two have a medium access complexity (CVE-2009-1961 and CVE-2010-4073). Based on the software structure analysis, we found that these vulnerabilities are reachable and seven of them have a DSC. Besides, there exists an exploit for all of them. Hence, based on the proposed metrics, the vulnerabilities that have a DSC have been assigned a high Structural Severity value while the vulnerabilities that do not have DSCs have been assigned a medium Structural Severity value.

6.4.7 Performance Evaluation of the Proposed Metric

Since data is available about the existence of exploits, we can compare the Structural Severity Reachability metric with the CVSS exploitability metrics based on the availability of exploits. To evaluate the performance of these two metrics, we used sensitivity, precision, and F-measure measures. These performance measures are explained using a confusion matrix as shown in Table

Table 6.10: Confusion matrix

	Prediction	
Actual	**Exploitable**	**Not exploitable**
Exploitable	TP= True Positive	FN= False Negative
Not exploitable	FP= False Positive	TN= True Negative

6.10. The confusion matrix table shows the actual vs. the predicted results. For the two class problem (a vulnerability is either exploitable or not exploitable), the following is defined based on Table 6.10.

- True Positive (TP): the number of the vulnerabilities predicted as exploitable, which do in fact have an exploit.
- False Negative (FN): the number of vulnerabilities predicted as not exploitable, which turn out to have an exploit.
- False Positive (FP): the number of vulnerabilities predicted as exploitable when they have no exploit.
- True Negative (TN): the number of vulnerabilities predicted as not exploitable when there is no exploit.

The selected performance measures can be derived as follows.

Sensitivity (Recall)

Sensitivity, which also termed recall, is defined as the ratio of the number of vulnerabilities correctly predicted as exploitable to the number of vulnerabilities that are actually exploitable as shown by the following:

$$Sensitivity = \frac{TP}{TP + FN}$$

Precision

Precision, which is also known as the correctness, is defined as the ratio of the number of vulnerabilities correctly predicted as exploitable to the total number of vulnerabilities predicted as

exploitable as shown by the following:

$$Precision = \frac{TP}{TP + FP}$$

For convenient interpretation, we express these two measures in terms of percentage, where a 100% is the best value and 0% is the worst value. Both precision and sensitivity should be as close to the value 100 as possible (no false positives and no false negatives). However, such ideal values are difficult to obtain because sensitivity and precision often change in opposite directions. Therefore, a measure that combines sensitivity and precision in a single measure is needed. Hence, we will introduce the F-measure in the following section. We believe that it is more important to identify exploitable vulnerabilities even at the expense of incorrectly predicting some not exploitable vulnerabilities as exploitable vulnerabilities. This is because a single exploitable vulnerability may lead to serious security failures. Having said that, we think more weight should be given to sensitivity than precision. Thus, we include F2-measure, which weights sensitivity twice as precision, to evaluate the two metrics.

F-measure

F-measure can be interpreted as the weighted average of sensitivity and precision. It measures the effectiveness of a prediction with respect to a user attaches the times as much importance to sensitivity as precision. The general formula for the F-measure is shown by the following:

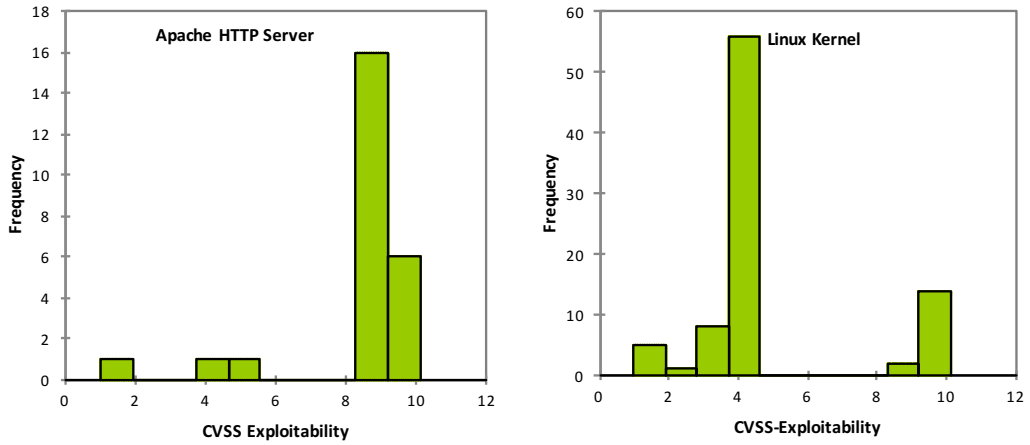
$$F_\beta - Measure = \frac{(1 + \beta^2) \times Precision \times Senetivity}{(\beta^2 \times Precision) + Senetivity}$$

β is a parameter that controls a balance between sensitivity and precision. When β = 1, F-measure becomes to be equivalent to the harmonic mean, whereas when β <1 it becomes more precision oriented. However, when β >1, F-measure becomes more sensitivity oriented. In this paper β has been chosen to be 2.

Comparison of the Performance Evaluation Results

Table 6.8 and Table 6.9 report the total subscore (SS column) for the CVSS exploitability metrics, the values of the Structural Severity Reachability metric, and the availability of the exploits for the Apache HTTP Server and Linux Kernel datasets respectively. The ranges of CVSS

exploitability metrics subscores are 1 to 10, whereas the values of the Structural Severity Reachability metric are R or NR. Figure 6.12 shows the distribution of the CVSS Exploitability subscore for Apache HTTP server and Linux Kernel datasets. Here, the population is split into two: vulnerabilities with CVSS Exploitability subscore >6 are considered exploitable (positive test) and those <6 are considered not exploitable (negative test). The minimum and maximum exploitability subscores for Apache and Linux are 1.9 and 10 respectively, whereas the mean and the standard deviation for Apache are 8.3 and 1.9 respectively. However, the mean and the standard deviation for Linux are 4.8 and 2.4 respectively.



(a) (b)

Figure 6.12: Distribution of CVSS Exploitability Subscores for Apache HTTP Server and Linux Kernel

As can be seen from Figure 6.12, most of the vulnerabilities do turn out to have a high Exploitability subscore for Apache whereas a low Exploitability subscore for Linux. This is because of the Access Vector metric. The majority of the vulnerabilities in Apache is remotely accessed and hence has been assigned a high exploitability subscore (8 out of 10). On the other hand, most of the vulnerabilities in Linux are locally accessed and thus have been assigned a low exploitability subscore (3.9 out of 10). It should be noted that neither the Authentication metric nor the Access Complexity metric values (Low and Medium) have a significant effect on the exploitability total subscore for both datasets. However, when the Access Complexity metric value is high it notably affects the exploitability total subscore. Nonetheless, only two out of 25 vulnerabilities in Apache

Table 6.11: Prediction Performance

Software	Performance Measures	CVSS Exploitability Metrics	Structural Severity Reachability Metric
Apache HTTP Server	Sensitivity	90.91%	100%
	Precision	45.45%	47.83%
	F1-Measure	61.00%	65.00%
	F2-Measure	75.76%	82.09%
Linux Kernel	Sensitivity	10.90%	100.00%
	Precision	44.00%	78.00%
	F1-Measure	18.00%	88.00%
	F2-Measure	13.00%	95.00%

dataset and five out of 86 vulnerabilities in Linux dataset have been found to have a high value Access Complexity.

Table 6.11 compares the two metrics using the Apache HTTP Server and Linux datasets. From the table we can observe that the Structural Severity Reachability metric performs better in terms of all measures than CVSS exploitability metrics for the Apache dataset. However, in the Linux dataset the Structural Severity Reachability metric performs much better than the CVSS exploitability metrics. It has been observed that when the software and the dataset size were changed, the Structural Severity Reachability metric performs better than CVSS exploitability metrics in terms of precision, F1 and F2-measures. This can be attributed to the fact that most of the vulnerabilities in the Linux Kernel require an attacker to have a Local access to exploit them and that makes their exploitation harder, and hence they have been assigned a low exploitability subscore. For the Apache dataset, the false positive rate for the two metrics is 85.71% whereas the false negative rate is 0% for the Structural Severity metric and 0.09% for the CVSS exploitability metrics. Having the 0.09% false negative rate is because the vulnerability CVE-2006-3747 has an exploit and the CVSS exploitability metrics predicted it as being not exploitable. On the other hand, for the Linux Kernel dataset the false positive rate slightly fell for the Structural Severity metric (82%), whereas it notably reduced for the CVSS exploitability metrics (41%). This can be explained by the increase in the false negative rate (89%). This can be attributed to the fact that there are 86 vulnerabilities with an exploit in Linux Kernel that have been assigned a low exploitability subscore.

We argue that it is better to have more nonexploitable vulnerabilities inspected than to have one exploitable vulnerability being left unchecked. It should be noted that the Structural Severity measure captures the exploitability factor by using only one attribute, reachability, whereas the CVSS exploitability metrics uses three attributes, Access Complexity, Access Vector, and Authentication.

6.5 Discussion and Threats to Validity

6.5.1 Discussion of the Case Study Results

Having an approach that does not depend on the availability of the security experts can be of great value. The proposed measure is based on software properties that reflect exploitability factors. We have observed that the proposed approach has not been found to be less restrictive than its counterpart, CVSS, except in two cases, when the vulnerabilities (CVE-2004-0488 and CVE-2007-5000) were found not reachable.

We have also observed that identifying critical functions can be accomplished by looking at the number of functions that are called by the vulnerable function. The more functions called by the vulnerable function, the higher the effect if the vulnerable function is exploited. Figure 7.12 shows the call graphs of the vulnerability CVE-2009-1195. This graph shows the number of functions that are called by the vulnerable function. This vulnerability impacts two functions as explained in subsection 5.3. Figure 7.12 (a) shows the vulnerable function process_resource_config_nofmatch directly calls 10 functions, whereas the other vulnerable function ap_process_config_tree directly calls only two functions as shown in Figure 7.12 (b). Bhattacharya et al. [94] introduced a bug severity metric called Node-Rank. It measures the relative importance of a function or module in the function call or module graphs. It could be interesting if the same can be done for vulnerability severity so the risk of exploitation can be further illustrated.

Sparks et al. [95] studied the penetration depth of reaching a node in a Control Flow Graph. They found that the nodes at greater depths (>10 edges) become increasingly difficult to reach. In other words, it is hard to craft an input that can lead to such a node at a particular depth. If crafting an input that can reach a vulnerable statement for a single method is difficult, we believe that crafting an input to call a method containing a vulnerable statement from other methods could

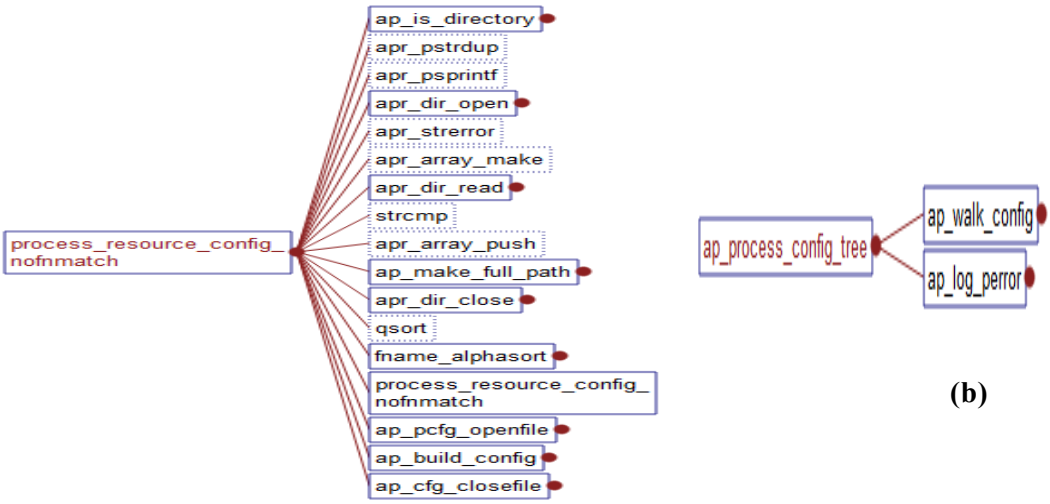


Figure 6.13: Functions that are called by the vulnerable functions (Calls Graph)

be even harder. If we further assume that the target system is a closed system, it gets even harder for the attackers to figure out the sequences of calls and inputs that are needed to trigger them. However, it has been also observed that the degree of a call depth of vulnerable functions varies among vulnerabilities. Some of the vulnerabilities have only one degree of depth while others have 13. Figure 7.13 shows the degree of depth of the vulnerability CVE-2010-0434. Due to page size limitation, we have only showed the depth up to level 5. We believe that the depth degree of a vulnerable function can inform us about the difficulty of exploiting a vulnerability, because the more functions an attacker has to invoke to reach a vulnerable function the harder to get it exploited.

Even though most of the property extraction has been automated, it has been noticed that performing vulnerability exploitation assessment remained dependable on the human. For instance, looking at a table with 25 vulnerabilities and comparing the selected properties to decide the risk of a given vulnerability is fairly attainable. However, having a larger dataset can be challenging and error-prone. It would be helpful if this part of the process can be automated by using some techniques from machine learning. Having the exploitability properties extracted from the source code, the machine learning model can automatically assess the exploitability risk based on the data. This idea could also help in managing scalability as machine learning techniques have shown success

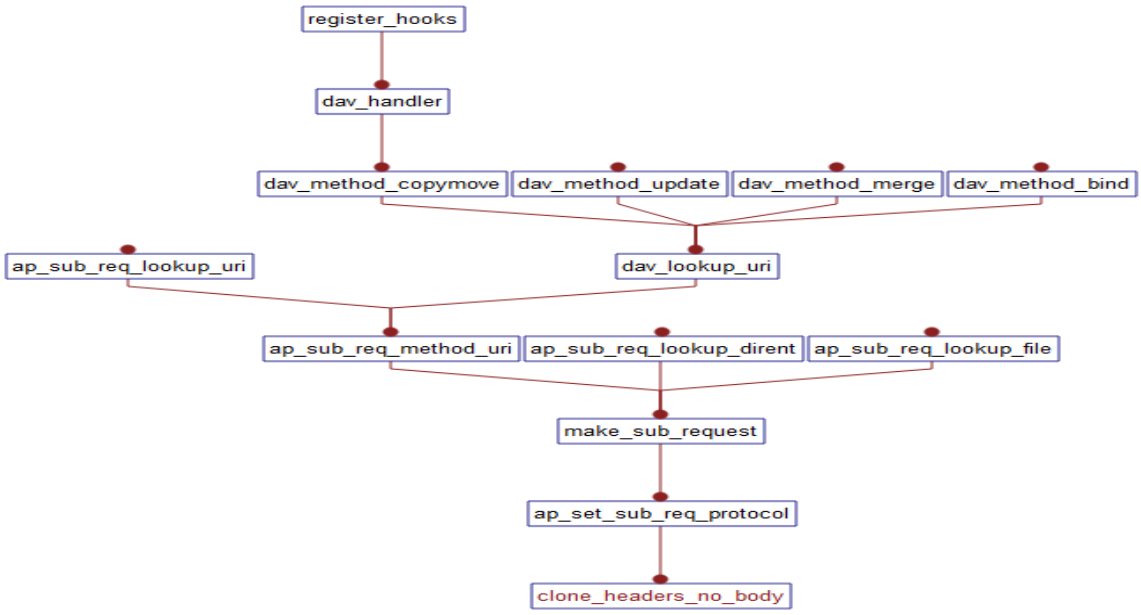


Figure 6.14: Depth of a vulnerable function calls (Called By Graph)

in dealing with sizable data sets. Moreover, it also makes adding added number of exploitability properties (features) handy.

It has also been noted that even though a vulnerability reachability is an important factor of exploitation, considering the exploit mitigation guards at the function level could be a great addition. For instance, having an exploit guards such as GuardStack (GS) can prevent exploits from exploiting a vulnerability even if the exploit has called the vulnerable function. Sparks et al. [95] stated that it is possible to identify functions that have not been compiled to use GS through the use of simple static analysis tools. This can be used as an estimator of exploitation impact. Thus, vulnerabilities (vulnerable functions) with exploit mitigation guards could be considered of a low risk.

6.5.2 Threats to Validity

External validity: The main threat to external validity is choosing two datasets, the Apache HTTP server and Linux Kernel. However, Apache HTTP server, since its release in 1995, has gone through a number of improvements which led to the release of several versions: 1.3.x, 2.0.x, 2.2.x, 2.3.x, and 2.4.x. For the release 1.3.x alone there are 47 versions whereas for the release

2.2 there are 23 versions. The Apache HTTP server has a richer history of public available vulnerabilities, more than 169. Besides, its line of code varies between 50,712 LOC to 358,633 LOC. Moreover, Apache HTTP server has varieties of vulnerabilities: Denial of Service (69), Execute Code (24), Overflow (16), XSS (21), Bypass Something (14), Gain Information (13), Directory Traversal (4), Gain Privilege (6), Memory Corruption (1), Http Response Splitting (1), and CSRF (1) (National Vulnerability Database 2013). Even though Apache HTTP server has mainly been developed using C/C++, languages like HTML, Java script, Python, and XML have been used too. Web servers form a major component of the Internet and they are targeted by attackers (more than 1,000 malicious HTTP requests in six months (Imperva 2012)). According to Usage Statistics and Market Share(2013), Apache web server is used by over 64% among most commonly used web servers. On the other hand, Linux Kernel has larger number of vulnerabilities, more than 1200. Its size in line of code has ranged from 10,239 LOC to 15,803,499 LOC. It also has a greater variety of vulnerabilities: Denial of Service (750), Execute Code (58), Overflow (213), Bypass Something (67), Gain Information (197), Directory Traversal (2), Gain Privilege (149), Memory Corruption (68), and Gain Privilege (149).

Internal Validity: The main threat to internal validity is the chosen factors for measuring the risk of vulnerability exploitation: reachability and DSCs. As an exploit in its basic form is an input, a vulnerability has to be reachable in order for the input to be able to trigger it. Therefore, reachability can be considered as a major contributor. However, not all vulnerabilities that are reachable are exploitable and that is because of the degree of difficulty of reaching vulnerabilities. For instance, vulnerabilities that are hard to reach are also hard to exploit and should be given less priority. This could introduce false positive results. Nevertheless, the degree of difficulty is dependent on whether a vulnerability is reachable or not. Once that is decided, we need to assess whether the code path to the vulnerable location is hard to reach or not. We recognize this threat and we are investigating the possibility of objectively measuring the difficulty of reachability using the following: 1) the number of function calls an attacker has to invoke to reach the vulnerable code, 2) an authentication verification mechanisms along the path to the vulnerable location, and 3) the privilege required to invoke the vulnerable function.

Even though using DSCs help in eliminating subjectivity, we recognize that capturing the technical impact by only considering the existence of a DSC might not be sufficient. Factors such as privilege of the vulnerable function, exploit mitigation, and authentication could be a major contributor too. However, measuring these factors can introduce subjectivity. Ensuring objective measurements of these factors requires further research.

Construct Validity: We used a commercial tool named Understand to automatically generate call graph between the functions. We also used SVN provided by the Apache project foundation to identify the vulnerabilities location. Moreover, we used the cflow and a python script to identify attack entry points. Usage of the third party tools and a python script represent potential threats to construct validity. We verified the results produced by these tools by manually checking randomly selected outputs produced by each tool.

Part of our measurement of exploitability uses the attack entry points as an estimate of the attack resources that an attacker can use to exploit vulnerabilities. Even though a significant concern resides in areas where the system obtains external data, there are vulnerabilities that can be exploited without sending data to the system. This represents a threat to construct validity. However, our vulnerabilities datasets have been collected from NVD that provides a list of the types of vulnerabilities that it uses. There are 19 types of vulnerabilities in the NVD. Based on their Common Weakness Enumeration (CWE) number and the description provided by the CWE and the NVD, we verified that the selected vulnerabilities are directly influenced by a user input.

6.6 Related Work

In this section, we review the work related to vulnerability exploitation risk assessment. We organize this section based on the method used to assess exploitation risk into: measurement-based, model-based, test-based, and analysis-based approaches.

6.6.1 Measurement-Based Approaches

Attack Surface Metric: The attack surface notion was first introduced by Howard in his Relative Attack Surface metric [96]. It was later formally defined by Manadhata and Wing [69].

They proposed a framework that included the notion of Entry and Exit Points and the associated damage potential-effort ratio. They applied their formally defined metric to many systems and the results show the applicability of the notion of the attack surface. Their new metric has been adapted by a few major software companies, such as Microsoft, Hewlett-Packard, and SAP. Manadhdata et al. [72] related the number of reported vulnerabilities for two FTP daemons with the attack surface metric along the method dimension. Younis and Malaiya [8] compared the vulnerability density of two versions of Apache HTTP server with the attack surface metric along the method dimension. However, attack surface metric does not measure the risk of exploitation of individual vulnerabilities. Rather, it measures the exploitability for the whole system and as a result it cannot help in prioritizing among vulnerabilities. Besides, neither Manadhata et al. [72] nor Younis and Malaiya [8], however, related entry points with the location of the vulnerability to measure its exploitability.

CVSS Metrics: CVSS metrics are the de facto standard that is currently used to measure the severity of vulnerabilities [4]. CVSS Base Score measures severity based on the exploitability (the ease of exploiting vulnerability) and impact (the effect of exploitation). Exploitability is assessed based on three metrics: Access Vector, Authentication, and Access Complexity. However, CVSS exploitability measures have come under some criticism. First, they assign static subjective numbers to the metrics based on expert knowledge regardless of the type of vulnerability, and they do not correlate with the existence of known exploit [97]. Second, two of its factors (Access Vector and Authentication) have the same value for almost all vulnerabilities [83]. Third, there is no formal procedure for evaluating the third factor (Access Complexity) [4]. Consequently, it is unclear if CVSS considers the software structure and properties as a factor.

6.6.2 Model-Based Approaches

Probabilistic Model: Joh and Malaiya [49] formally defined a risk measure as a likelihood of adverse events and the impact of this event. On one hand, they utilized the vulnerability lifecycle and applied the Markov stochastic model to measure the likelihood of vulnerability exploitability for an individual vulnerability and the whole system. On the other hand, they used the impact re-

lated metrics from CVSS to estimate the exploitability impact. They applied their metric to assess the risk of two systems that had known unpatched vulnerabilities using actual data. However, the transition rate between vulnerability lifecycle events has not been determined and the probability distribution of lifecycle events remains to be studied. Moreover, the probability of being in an exploit state requires information about the attacker behavior which might not be available. Additionally, the probability of a patch being available but not applied requires information about the administrator behavior which has not been considered by the proposed model and also hard to be obtained. In contrast, we assess vulnerability exploitability for individual vulnerabilities based on the source code properties regardless of the availability or unavailability of a patch.

Logistic Model: Vulnerability density metric assesses the risk of potential exploitation based on the density of the residual vulnerabilities [71]. The density of residual vulnerabilities is measured based on the number of known reported vulnerabilities and the total number of vulnerabilities. However, the total number of vulnerabilities is unknown but can be predicted using vulnerability discovery models (VDMs). Alhazmi and Malaiya [28] proposed a logistic vulnerability discovery model, termed the AML model. AML examines the reported known vulnerabilities of a software system to estimate the total number of vulnerabilities and their rate of discovery. However, considering the number of vulnerabilities alone is insufficient in assessing the risk of an individual vulnerability exploitation. Because different vulnerabilities have different opportunity of being exploited based on their properties such as reachability.

Machine Learning based Metric: Bozorgi et al. [97] aimed at measuring vulnerability severity based on likelihood of exploitability. They argued that the exploitability measures in CVSS Base Score metric cannot tell much about the vulnerability severity. They attributed that to the fact that CVSS metrics rely on expert knowledge and static formula. To that end, the authors proposed a machine learning and data mining technique that can predict the possibility of vulnerability exploitability. They observed that many vulnerabilities have been found to have a high severity score using CVSS exploitability metric although there were no known exploits existing for them. This indicates that the CVSS score does not differentiate between exploited and non-exploited vulnerabilities. This result was also confirmed by [83, 98]. However, unlike their work, ours relies

on software properties such as the attack surface entry points, the source code structure, and the vulnerabilities location to estimate vulnerability exploitability. This is particularly important for newly released applications that do not have a large amount of historical vulnerabilities.

6.6.3 Test-Based Approaches (proof of concept)

Automated exploit-generation system (AEG): Avgerinos et al. [99] proposed an automated exploit-generation system (AEG) to assess the risk of vulnerability exploitation. AEG rst uses the static analysis to nd a potential bug locations in a program, and then uses a combination of static and dynamic analysis to nd an execution path that reproduces the bug, and then generates an exploit automatically. AEG generates exploits, which provides an evidence that the bugs it nds are critical security vulnerabilities. However, generating an exploit is expensive and does not scale. AEG has only been applied to a specific type of vulnerabilities and software.

Black Box Fuzz Testing: Sparks et al. [95] extended the black box fuzzing using a genetic algorithm that use the past branch proling information to direct the input generation in order to cover specied program regions or points in the control ow graph. The control ow is modeled as Markov process and a tness function is dened over Markov probabilities that are associated with a state transition on the control ow graph. They generated inputs using the grammatical evolution. These inputs are capable of reaching deeply vulnerable code which is hidden in a hard to reach locations. In contrast to their work, ours relies on the source code analysis, a link between vulnerability location and attack surface entry points, and the DSC analysis that were specifically intended for measuring vulnerability exploitability.

6.6.4 Analysis-Based Approaches

Black Market Data Analysis: Allodi and Massacci [83, 98] proposed the black market as an index of risk of vulnerability exploitation. Their approach assesses the risk of vulnerability exploitation based on the volumes of the attacks coming from the vulnerability in the black market. It first looks at the attack tools and verifies whether the vulnerability is used by such tool or not. It also analyzes the attacks on the wild to verify whether the vulnerability has been a target of such

attacks or not. If the vulnerability is being used by one of the attack tools or being a target of real attacks, they consider this vulnerability as a threat for exploitation. This approach has introduced a new view of measuring the risk of exploitation by considering the history of attacks at the vulnerabilities. This approach does not require spending large amount of technical resources to thoroughly investigate the possibility of vulnerability exploitation. However, this approach requires a vulnerability intelligence provider as the information about the attacks and tools are dynamic in nature. Moreover, if the vulnerability right now is not used by a tool or it is not a target of an attack, it does not mean that it is going to be so continually. Our approach, on the other hand, relies only on software properties and does not make any assumption about the attacks and attackers resources.

Source Code Analysis: Brenneman [100] introduced the idea of linking the attack surface entry point to the attack target to prioritize the effort and resource required for software security analysis. Their approach is based on the path-based analysis, which can be utilized to generate an attack map. This helps visualizing the attack surfaces, attack target, and functions that link them. This is believed to make a significant improvement in software security analysis. In contrast to their work, we do not only utilize the idea of linking the attack surface entry point with the reported vulnerability location to estimate vulnerability exploitability, but also check for the DSCs inside every related entry point to estimate the impact of exploitation. The use of the DSCs is helpful for inferring an attackers motive in invoking the entry point method.

System Calls Analysis: Massimo et al. [91] presented a detailed analysis of the UNIX system calls and classify them according to their level of threat with respect to the system penetration. To control these system calls invocation, they proposed the Reference Monitor for UNIX System (REMUS) mechanism to detect an intrusion that may use these system calls which could subvert the execution of privileged applications. Nevertheless, our work applies their idea to estimate the impact of exploitation, as attackers usually look to cause more damage to targeted systems. Thus, our work is not about intrusion detection but rather measuring the exploitability of a known vulnerability.

6.7 Conclusion and Future Work

Assessing the severity of a vulnerability requires evaluating the potential risk. Existing measures do not consider software access complexity and tend to rely on subjective judgment. In this paper, we have proposed an approach that uses system related attributes such as attack surface entry points, vulnerability location, call function analysis, and the existence of DSCs. This approach requires us to examine some of the structural aspects of security such as the paths to the vulnerable code starting from the entry points. We have demonstrated the applicability of the proposed approach and have compared resulting measures with overall CVSS severity metrics. Our results show that this approach, involving assessment of the system security based on systematic evaluation and not subjective judgment, is feasible.

While the main parts of the analysis have been automated, providing a framework that can automate the entire analysis will be helpful in reducing the effort. We plan to develop techniques to reduce human involvement and thus enhance scalability in assessing exploitability risk by using machine learning techniques. We plan to examine the effectiveness of machine learning for automatically assessing the risk of vulnerability exploitation using the proposed properties as features. Given a vulnerable function and their exploitability features the machine learning model can predict whether it is an exploitable function and estimate the impact of its exploitation.

Even though measuring the possibility of reaching a vulnerability is important, quantifying the degree of difficulty of reaching a vulnerability is also valuable for comparing the severity among similar vulnerabilities, and thus needs to be examined. We plan to utilize the idea of the function call graph depth which has been presented in the discussion section. Devising a way of estimating the impact of reachable vulnerabilities will be valuable for estimating the overall risk of individual vulnerabilities and the whole system. We plan to further study the Node-Rank proposed by Bhattacharya et al. [94] as an estimator of the vulnerability impact. Finally, identifying whether a vulnerable function is guarded by security control can help better understand the impact of exploitation. We intend to study how function exploitation properties proposed by Skape [101] can give more information about the risk of vulnerability exploitation.

Chapter 7

Characterizing Vulnerability Exploitability

In this chapter, internal software attributes (metrics) that can be used to predict vulnerability exploitability risk will be examined. We characterize the vulnerable functions that have no exploit and the ones that have an exploit using eight metrics: Source Line of Code, Cyclomatic complexity, CountPath, Nesting Degree, Information Flow, Calling functions, Called by functions, and Number of Invocations. We first test the discriminative power of the individual selected metrics using the Welch t-test. Then we select a combination of the metrics using three feature selection methods and evaluate their predictive power using four classifiers.

7.1 Introduction

Identifying and addressing software vulnerabilities is important before software release because a single software vulnerability can lead to a breach with a high impact to an organization. However, identifying and addressing potential vulnerabilities can take considerable expertise and effort. Recently, researchers [102, 103, 104, 35] have started investigating ways to predict code areas which are more likely to be vulnerable so security testers can focus on them.

Software vulnerabilities, pose different levels of potential risk. A vulnerability with an exploit written for it presents more risk than the one without an exploit because the existence of an exploit allows an attacker to take advantage of a vulnerability and potentially compromise the affected systems. Allodi and Massacci in [83] have shown that out of the 49599 vulnerabilities reported by the National Vulnerability Database, only 2.10% are in fact exploited. Younis and Malaiya in [7] have also found that only 6.8% out of 486 vulnerabilities of Microsoft Internet Explorer have reported exploits. K. Nayak et al. [105] have reported that combining all of the products they have studied only 15% of disclosed vulnerabilities are ever exploited. Thus, identifying what characterizes a vulnerability having an exploit is needed; it can identify code that are more likely than others to have exploits and help security testers focus on areas of highest risk, thus saving

limited resources and time. It should be noted that having a reported exploit does not necessarily mean some company or individuals have suffered a real attack. It means that a proof for exploiting a vulnerability exists. Obtaining data on real attacks is challenging because such data is generally kept confidential. Therefore, we will use the presence of an exploit as the ground truth for characterizing exploited vulnerabilities.

Discriminating between a vulnerability that has no exploit from the one that has an exploit is challenging because both of them have similar characteristics. Besides, the number of vulnerabilities with a reported exploit are few compared to the vulnerabilities without a reported exploit. Although vulnerability exploitability can be characterized by external factors such as attacker profile, software market share, etc., the focus of this study is on predicting vulnerability exploitability using internal attributes. This can help software developers predict vulnerabilities exploitability on the development side rather than the deployment side.

The objective of this research is to investigate what could characterize a code containing a vulnerability with an exploit. To address this objective, we have studied 183 vulnerabilities from the National Vulnerability Database [16] for the Linux Kernel and Apache HTTP server. The two software systems have been selected because of their rich history of publicly available vulnerabilities, availability of reported exploits, the existence of an integrated repository, availability of the source code, and their diversity in size, functionalities, and domain. For every selected vulnerability, we verify whether it has an exploit reported in the Exploit Database or not [22]. Eighty-two vulnerabilities have been found to have an exploit. Ten of them are for Apache HTTP server and 72 for Linux Kernel. We then mapped these vulnerabilities to their locations at the function granularity level.

After that, we characterize the vulnerable functions with and without an exploit using the selected eight software metrics: Source Line of Code, Cyclomatic complexity, CountPath, Nesting Degree, Information Flow, Calling functions, Called by functions, and Number of Invocations. The reasons why these metrics have been selected are discussed in the hypotheses and methodology section. Based on the metrics values of the vulnerable functions with and without an exploit, we first test the individual selected metrics discriminative power using Welch t-test [106]. Next,

we select a combination of these metric using three feature selection methods: correlation-based, wrapper, and principal component analysis. Then, we test the predictive power of the selected subset of metrics using four classifiers: Logistic Regression, Nave Base, Random Forest, and Support Vector machine.

The results demonstrate that vulnerabilities having an exploit can be characterized. The investigation also shows that predicting exploitation of vulnerabilities is more complex than predicting the presence of vulnerabilities and hence further research that considers metrics from security domain is needed to improve the predictability of vulnerability exploits.

This paper is organized as follows. In section 8.2, the background of the vulnerabilities, vulnerability databases, exploit database, software metrics, confusion matrix, and feature subset selection methods are discussed. In the next section, the hypotheses are examined and the methodology of testing them is presented. In sections 8.4, the case studies along with the results are introduced. Section 8.5 presents the discussion whereas section 8.6 presents the related work. Finally, concluding comments is given along with the issues that need further research.

7.2 Related Topics and Concepts

7.2.1 Software Metrics

A software metric is a measure of some property of a piece of software. Table 7.1 summarizes the eight selected metrics. We have selected these metrics based on prior research on fault and vulnerability prediction.

7.2.2 Confusion Matrix

The confusion matrix table shows the actual vs. the predicted results. For the two class problem a vulnerability is either has an exploit or has no exploit. The following terms are defined based on Table 7.2.

True Positive (TP): the number of the vulnerabilities predicted as having an exploit, which do in fact have an exploit. False Negative (FN): the number of vulnerabilities predicted as not having an exploit, which turn out to have an exploit. False Positive (FP): the number of vulnerabilities

Table 7.1: Software Metrics

Metrics	Description
Source Line of Code	SLOC measures the size of a code [107]. A higher value of SLOC indicates that an entity is to be difficult to test.
Cyclomatic complexity	CYC measures the number of independent paths through a program unit [108]. The higher this metric the more likely an entity is to be difficult to test.
CountPath	CountPath measures the number of unique decision paths. A higher value of the CountPath metric represents a more complex code structure [107].
Nesting Degree	ND measures the maximum nesting level of control structures in a function. The higher this metric the more likely an entity is to be difficult to test [109].
Information Flow	”Fan-In measures information flow which represents the number of inputs a function uses [110]. The more inputs from external sources the harder to trace where they came from. ”
Calling Functions	”In-Degree measures the number of functions that call the function corresponding to the node [111]. The more dependent upon a peace of code the higher the chance it has a defect.”
Called by Functions	”Out-Degree measures the number of functions that the function corresponding to the node calls [111]. The more depends upon other code the higher the chance to have a defect. ”
Number of Invocations	It measures the number of functions that needed to be called before invoking the vulnerable function [12]. The higher this metric the more difficult to reach the vulnerable code.

Table 7.2: Confusion matrix

	Prediction	
Actual	**Has an Exploit**	**Has no Exploit**
Has an Exploit	TP= True Positive	FN=False Negative
Has no Exploit	FP= False Positive	TN= True Negative

predicted as having an exploit when they have no exploit. True Negative (TN): the number of vulnerabilities predicted as not having an exploit when there is no exploit.

7.2.3 Feature Subset Selection

Two commonly known types of feature subset selection methods are the filter and the wrapper approach. In the filter approach, the feature selection is performed independently of the learning algorithm and it selects a subset based only on the data characteristics. The wrapper approach, however, conducts a search for a good subset using the learning algorithm as part of the evaluation function [65].

Correlation-based feature selection

Correlation-based feature selection (CFS) evaluates subsets of attributes instead of individual attributes [112]. This technique uses a heuristic to evaluate subset of attributes. The heuristic balances how predictive a group of features are and how much redundancy is among them. In this study, CFS is used with the Greedy stepwise forward search through the space of attribute subsets.

Wrapper Subset Evaluation

The wrapper feature subset evaluation conducts a search for a good subset using a learning algorithm (classifier) as part of the evaluation function. In this study, repeated five-fold cross-validation is used as an estimate for the accuracy of the classifier while a greedy stepwise forward search is used to produce a list of attributes, which are ranked according to their overall contribution to the accuracy of the attribute set with respect to the target learning algorithm [113].

Principal Component Analysis

Principal Component Analysis (PCA) is a statistical technique that transforms a set of possibly correlated variables into a set of linearly uncorrelated variables [114]. These linearly uncorrelated variables are called principal components. The transformation is accomplished by first computing the covariance matrix of the original variables and after that finding its Eigen vectors, principal components. The principal components have the property that most of their information content

is stored in the first few features so that the remainder can be discarded. It should be noted that in this paper, PCA is used with the ranker search method that ranks attributes by their individual evaluations.

7.3 Hypotheses and Methodology

In this section, we provide our hypotheses that are needed to be researched and then provide the methods to test them.

7.3.1 Hypotheses

It should be noted that the metrics in section 2.3 have been classified into four classes: size (SLOC), structure (CYC, CountPath, ND), ease of access (Number of Invocations), and communication (Fan-In, In-Degree, Out-degree) so to ease the analysis and observation. The rationale behind using the selected metrics to derive our hypothesis is explained as follows.

In the software security field, experts argued that complexity is the enemy of security [102]. It is believed that complexity can be the cause of subtle vulnerabilities that are hard to test and analyze [115] and hence providing a chance to attackers to exploit them. However, we consider the fact that predicting a presence of a vulnerability is different from predicting its exploitation because the latter involves the attacker behavior. As the measures of the complexity vary, we consider the possibility that the potential exploit writers would prefer to exploit less complex code [116]. Besides, in [117], the researchers have observed that the complexity measures for Windows 7 and 8 were negative and they argued that the reason could be that attackers favor simpler vulnerable targets. Based on these reasons, we set up the following research hypotheses related to code complexity (size and structure metrics):

H1*: The values of the size metric for vulnerabilities with an exploit are lower than for vulnerabilities without an exploit.*

H2*: The values of the structure metrics for vulnerabilities with an exploit are lower than for vulnerabilities without an exploit.*

Sparks et al. [95] studied the penetration depth of reaching a node in a control flow graph. They found that the nodes at greater depths (¿10 edges) become increasingly difficult to reach. If crafting an input that can reach a vulnerable statement for a single method is difficult, we believe that crafting an input to call a method containing a vulnerable statement from other methods could be even harder. If we further assume the target system is a closed system, it gets even harder for the attackers to figure out the sequences of calls and inputs that are needed to trigger them. Younis et al. [12] observed that the degree of a call depth of vulnerable functions varies among vulnerabilities. Some of the vulnerabilities have only one degree of depth, while others have 13. Thus, we set up the following research hypothesis for the ease of access (Number of Invocations metric):

H3*: The values of the ease of access metric for vulnerabilities with an exploit are lower than for vulnerabilities without an exploit.*

On the other hand, Younis et al. [12] argued that the more functions are called by the vulnerable function, the higher the effect if the vulnerable function is exploited. They have shown that some vulnerable functions call more than 10 functions while other functions call only one or two functions. Based on the attack surface concept [96], however, the more a function is exposed to the outside environment the larger attack surface. Thus, we argue that the higher the communication a function has, the larger its attack surface gets. From this reasoning, we set up the following research hypothesis for the communication (Fan-In, In-Degree, Out-degree metrics):

H4*: The values of communication metrics for vulnerabilities with an exploit are higher than for vulnerabilities without an exploit.*

As argued by Manadhata and Wing [69], no single metric can be an indicator of software quality for all types of software. Besides, according to [19], a feature (or two features) that is completely useless by itself (themselves) can be useful when taken with others (together). Therefore, determining the combination of multiple features (metrics) is important. Thus, we set up the following research hypothesis:

H5*: There is a combination of metrics that significantly predicts vulnerabilities with an exploit.*

7.3.2 Evaluation Strategy for the Hypotheses

We first investigate the discriminative power of the proposed individual metrics and then test their predictive capability when they are combined. A metric has a discriminative power if it can discriminate between high-quality software components and low-quality software components [118]. In this reserach, a vulnerable function is classified as exploited if there exists a reported exploit for it and as not exploited if there exists no reported exploit for it at the time of our study. To evaluate the discriminative power of the proposed hypotheses, H1, H2, H3, and H4, we test their null hypotheses. Because our data has unequal sample size, unequal variances and is skewed, we used the Welch t-Test [106]. The Welch t-test is an adaptation of t-test to compare the means of two samples when the two samples have unequal variances and unequal sample sizes and it also is known to provide a good performance for skewed distributions. The difference between the means of the two group is considered discriminative when the result from the Welch t-test are statistically significant at the $p < 0.05$ level.

However, to evaluate the predictive power of the hypothesis H5 (combined metrics), on the other hand, two challenges have to be addressed. First, how can we select the subset metrics? Second, how can we evaluate the predictive power of the selected subset? To address the first challenge, we use feature subset selection methods. There are two commonly used feature subset selection methods: the filter and the wrapper approach. In this paper, we use correlation-based feature selection (CFS) and wrapper subset evaluation (WRP) methods. The first method has been selected because it selects the subset features based on only the characteristics of the data while the second method selects the features based on the learning algorithm and this can help us observe an advantage of one method over the other. However, according to [111], combining several metrics can be affected by the multicollinearity and that is due to the inter-correlation among metrics. To account for the multicollinearity problem, we use the Principal Component Analysis (PCA) and compare its result with two selected methods.

To address the second challenge, we evaluate the performance of a binary classification technique to assess the predictive power of the selected subset metrics. A binary classifier can make two possible types of errors: false positives and false negatives. These two types of errors are

defined in section 2.4. To measure the performance of a classifier, we mainly use recall, precision and false positive ratio. In addition, we also report the harmonic mean of the precision and recall, using the F-measure, and the accuracy. It should be noted that a high recall value is required even at the cost of the precision and that is because of the significant impact of one exploited vulnerability. It is also desirable to have a low false positive rate because that helps avoiding a waste of an inspection effort. P. Morrison et al. in [117] suggested that the value 0.7 of the recall and precision measures are considered reasonable for the prediction models in the realm of software quality.

Recall is defined as the ratio of the number of vulnerabilities correctly predicted as having an exploit to the number of. Apache HTTP Server Vulnerabilities Measures vulnerabilities that actually have an exploit as shown by the following: Recall = TP / TP+FN. Precision, on the other hand, is defined as the ratio of the number of vulnerabilities correctly predicted as having an exploit to the total number of vulnerabilities predicted as having an exploit as shown by the following: Precision = TP / TP+FP. False positive rate is defined as the ratio of the vulnerabilities incorrectly predicted as having an exploit to the total number of vulnerabilities that have no exploit: FP rate = FP / FP+TN.

We have considered four classifiers for this study and they are namely: Logistic Regression (LR), Nave Bayes (NB), Random Forests (RF), and Support Vector Machine (SVM). LR has been selected because it is a standard statistical classification technique whereas NB has been selected because it is a simple classifier and it has often outperformed more sophisticated classifiers [35]. Besides, RF has been selected because it is more robust to noise such as inter-correlated features while SVM has been selected because it is less prone to overfitting.

7.4 Experimentation

The purpose of the experiment is to investigate whether there is a difference in characteristics between a vulnerable function without exploits and a function with exploits. We have selected two software systems namely: Linux Kernel and Apache HTTP Server. The two software systems have been selected because of their rich history of publicly available vulnerabilities, availability of reported exploits, existence of an integrated repository (which enables us to map vulnerabilities to

Table 7.3: Vulnerabilities and Exploits

Software	EE	NEE	Total Each
Linux Kernel	72	81	**153**
Apache	10	20	**30**
Total All	**82**	**101**	**183**

their location in the source code), availability of the source code (which enables us to collect the measures of the proposed metrics), and their diversity in size, functionalities, and domain.

7.4.1 Data Collection

In this study, the data about vulnerabilities and exploits of Linux kernel and Apache HTTP Server were collected from NVD [16] and the EDB [22] respectively from the period 2002 to 2014. Table 7.3 shows the number of the selected vulnerabilities and their exploits. It should be noted that we have considered all reported vulnerabilities that have an exploit for the two selected software system. We only considered some of the vulnerabilities that do not have an exploit. These vulnerabilities have been mainly selected based on their age and information about their locations. We tried to select the vulnerabilities that are at least 3 or 4 years old, so that their lack of exploit is not due to their recent discovery.

7.4.2 Computing the Metrics

To collect the selected metrics, we use a function as a logical unit for analysis. Before we can take the measures of the selected metrics, the location of the vulnerable function is needed to be identified. The location can be found by looking at the report in the vulnerability database. The following steps have been followed to identify the location:

- From the vulnerability database, identify the vulnerability.
- From the Bug Repository (Bugzilla) and Version Archive:
  - Identify the vulnerable version (e.g., Apache 1.3.0)
  - Identify files by mapping CVE number to Bug ID
  - Identify the vulnerable function

Table 7.4: Apache HTTP Server Vulnerabilities' Measures

Vulnerability	In-Degree	Out-Degree	Count Path	ND	CYC	Fan-In	No of Invocation	SLOC	Exploit Existence
CVE-2002-0839	2	2	3	2	3	7	2	8	NEE
CVE-2003-0016	1	10	9999	6	63	35	3	318	NEE
CVE-2004-0492	4	8	146	4	11	16	4	48	NEE
CVE-2004-0488	1	0	10	2	5	5	3	22	NEE
CVE-2004-0940	12	4	134	5	35	17	2	63	EE
CVE-2006-3747	2	2	12	4	16	5	3	34	EE
CVE-2009-1890	1	8	158	4	15	21	3	60	EE
CVE-2009-1891	1	9	9000	6	68	45	2	211	NEE
CVE-2010-0010	4	9	145	4	11	16	4	38	EE
CVE-2013-1896	26	5	8	1	5	37	3	29	EE

Once the vulnerable version and the vulnerable function have been identified, we can now compute the metrics at the function level from the source code [79] and [119].There are different tools that can be used to compute these metrics. We have chosen the commercial tool Understand [89]. This tool has been chosen because it is user-friendly and it has a good set of APIs that allows interaction with programming languages such as Python, Perl, and C++. For the selected vulnerable version we have performed the following using our own python script:

- Search inside all folders in the main folder and find all .c files and store them in a list
- From the list, select the .c files that contain the vulnerabilities
- For every selected file, find the vulnerable function(s)
- Using the commercial tool Understand, compute the selected metrics.

Showing the whole measures for the selected software is limited by the number of pages, thus Table 7.4 shows the measures of the selected metrics for some of Apache HTTP Server vulnerabilities. As can be seen, the measures of the vulnerabilities are distinguished by the availability of exploit as either an exploit exist (EE) or no exploit exist (NEE).

7.4.3 Discriminative Power Test

Apache Dataset

Table 7.5 shows the results of testing the hypotheses H1, H2, H3, and H4 for the discriminative power of the individual metrics for the Apache HTTP Server dataset. It should be noted that we have performed an outliers test for all computed metrics to avoid their effect on the mean. Only one metric, calling functions, has been found to have outliers.

H1: As can be seen, the size metric has shown to have smaller values for vulnerabilities with exploits than those without an exploit and this difference is statistical significant at p-value 0.021. Therefore, the Welch t-test result suggests that the vulnerabilities with an exploit tend to have a smaller size than the vulnerabilities without an exploit.

H2: Looking at the structure metrics values, however, only the CountPath metric has shown statistical significant difference, the p-value is 0.011, for vulnerabilities with exploits compared to those without an exploit. Thus, the Welch t-test result implies that the vulnerabilities with an exploit tend to have smaller CountPath than the vulnerabilities without an exploit. On the other hand, while the Cyclomatic complexity values are smaller for vulnerabilities with an exploit, though the difference is not significant, Nesting Degree values are higher for vulnerabilities with an exploit and this is a contrary to what we anticipated. However, according to [120], the recommended maximum for Nesting Degree is 5 and those two values are less than this number. This suggests that Nesting Degree is smaller for both vulnerabilities with an exploit and those without an exploit.

H3: The ease of access metric has shown slightly higher values for vulnerabilities with exploits which is not what we anticipated. The median for the vulnerabilities with and exploits and without an exploit is 3.0 and 2.5 respectively. Therefore, the Welch t-test result suggests that the vulnerabilities with an exploit tend to have a slightly more number of invocations than those without an exploit but this difference is not significant. However, Sparks et al. in [95] found that the nodes at greater depths ($>$10 edges) become increasingly difficult to reach. Therefore, we conclude that the degree of depth (around 3.0) for both groups is smaller.

H4: As can be seen from the communication metrics values, only calling functions metric has shown to have higher values for vulnerabilities with exploits than for the vulnerabilities without

Table 7.5: Result of Discriminative Power Test for Apache HTTP Server Dataset

			EE (Observations = 10)		NEE (Observations = 20)			
Class of Metrics	**Metrics**		**Mean**	**Variance**	**Mean**	**Variance**	**t-value**	**P-Value**
Size	SLOC		54.2	134.4	698.4	18148.2	-2.49	**0.021**
Structure	Cyclomatic complexity		13.6	27.3	83.8	766.6	-1.95	0.063
	Nesting Degree		3.7	2.5	3.4	3.7	0.43	0.669
	CountPath		45.4	2462.3	3072	13840876.5	-2.83	**0.011**
Ease of Access	Number of Invocations		2.9	0.4	2.5	0.6	1.55	0.136
Communication	Information Flow		14.69	93.5	17.5	150.6	-0.73	0.472
	Calling functions	**Outliers**	5.7	70.3	1.4	0.7	1.53	0.163
		No outliers	5.2	64.6	1.2	0.3	1.6	0.151
	Called by functions		4.9	9.1	8.4	37.5	-2.01	**0.054**

an exploit. However, this difference is not statistically significant. On the other hand, information flow and called by functions metrics have shown a smaller values for vulnerabilities with an exploit and that is not what we anticipated.

Linux Kernel Dataset

We also obtained the results of testing the hypotheses H1, H2, H3, and H4 for the discriminative power of the individual metrics for the Linux Kernel dataset, as shown in Table 7.6. We have performed an outlier test for all computed metrics so that to avoid its effect on the mean. All metrics values have been found to have outliers except for one metric, number of invocations. We have run the Welch t-test on both data with and without outliers.

H1: The size metric values for the data without outliers show that the size of the vulnerabilities with an exploit and without an exploit is almost the same and this is not what we anticipated. Therefore, based on the p-value, we accept that there is no difference between the vulnerabilities with an exploit and without an exploit.

H2: The values of the structure metrics without outliers are smaller for vulnerabilities with an exploit than for those without an exploit except for the CountPath metric where its values have been found to be higher, which is not what we anticipated. However, these differences are not statistically significant. Therefore, the Welch t-test result implies that none of these differences are statistically significant.

Table 7.6: Result of Discriminative Power Test for Linux Kernel Dataset

			EE (Observations = 72)		NEE (Observations = 81)			
Class of Metrics	**Metrics**		**Mean**	**Variance**	**Mean**	**Variance**	**t-value**	**P-Value**
Size	SLOC	**Outliers**	64.1	10496.8	60.7	4958.5	0.2	0.816
		No outliers	49.2	2715.2	50.0	1980.9	-0.1	0.922
Structure	Cyclomatic Complexity	**Outliers**	20.1	1278.5	20.3	1555.5	-0.02	0.984
		No outliers	13.3	220.1	16.3	389.4	-1.0	0.296
	Nesting Degree	**Outliers**	2.3	1.8	2.6	3.3	-1.1	0.284
		No outliers	2.2	1.4	2.4	2.2	-0.9	0.377
	CountPath	Outliers	3.3E+07	2.9E+16	2.8E+07	2.5E+16	0.2	0.851
		No outliers	4.7E+06	1.5E+15	2.7E+06	2.9E+14	0.4	0.703
Ease of Access	Number of Invocations		2.1	1.0	2.2	2.0	-0.6	0.574
Communication	Information Flow	Outliers	18.3	336.1	26.3	8125.5	-0.8	0.442
		No outliers	15.4	116.3	16.4	433.5	-0.4	0.705
	Calling functions	Outliers	6.5	265.5	14.4	6756.3	-0.8	0.405
		No outliers	3.4	51.8	5.5	465.0	-0.8	0.428
	Called by functions	Outliers	12.4	151.1	11.8	77.2	0.4	0.707
		No outliers	11.2	96.1	11.7	76.6	-0.3	0.748

H3: The values of the ease of access metric show that the mean of the vulnerabilities with an exploit are almost the same compared to the vulnerabilities without an exploit and this is not what we anticipated. Therefore, the Welch t-test result suggests that there is no significant difference between the vulnerabilities with an exploit and those without an exploit. However, as it was discussed in section 4.3.1 under the H3 paragraph, the mean of the measures of these two groups are considered to be small.

H4: The communication metrics have shown to have smaller values. This is not what we anticipated. However, looking at the p-values, we can see that the differences are not statistically significant. Thus, we cannot reject the null hypothesis that there is no difference between the vulnerabilities with an exploit and those without an exploit.

7.4.4 Predictive Power Test

To test predictive power of the metrics, we need to select a subset of the proposed metrics. To do that, we implemented CFS, WRP, and PCA feature selections techniques using Waikato Environment for Knowledge Analysis (WEKA) [121]. WEKA is a popular open source toolkit implemented in Java for machine learning and data mining tasks. Once the metrics subsets have

been selected, we implemented the four selected classifiers: LR, NB, RF, and SVM using WEKA. It should be noted that the parameters for the chosen feature selection techniques and classifiers are initialized with the default settings of the WEKA toolkit. It should be also noted that the results are obtained by performing 10-fold cross-validation so that the variability in prediction are reduced. Cross-validation is a technique for assessing how accurately a predictive model will perform in practice [122]. However, it is more important to identify exploited vulnerabilities even at the expense of incorrectly predicting some not exploited vulnerabilities as exploited vulnerabilities. This is because a single exploited vulnerability may lead to serious security failures. Thus, we use recall as our main performance measure to compare among the classifiers performance. The results of testing metrics predictive power are shown in the following two subsections.

Apache Dataset

Table 7.7 shows predictive power results for the Apache dataset. Column one and two contain the classifiers and their performance measures respectively. We first start with testing every classifier using the whole selected metrics and collect the performance measures provided by WEKA, as shown in column three. For convenient interpretation, we express the performance measures in Table 6 shows predictive power results for the Apache dataset. Column one and two contain the classifiers and their performance measures respectively. We first start with testing every classifier using the whole selected metrics and collect the performance measures provided by WEKA, as shown in column three. For convenient interpretation, we express the performance measures in terms of percentage, where a 100 % is the best value and 0 %is the worst value. Then, we select a subset of those metrics using CFS, WRP, and PCA feature selection techniques and test the chosen classifier using the selected subsets and provide the performance measures as shown in column four, five, and six respectively.

Considering the value 0.7 (70%) of recall and precision measures as reasonable [117], we show for every metrics subset the highest recall for a classifier in bold. Now, we will compare the classifiers performance measures using different subset metrics. We will first start with using the whole metrics. As can be seen, only NB and RF report the best recall and precision value.

Moreover their accuracy and precision performance measures are either 70 or more. Even though NB reports the best precision, the reported harmonic mean, F-measure, by RF is better. It should be also notated that FPR reported by NB is lower than the one reported by RF. However, when the subset metrics selected by the correlation-based feature selection, we see that not only LR has improved but it has done better than the other classifiers. Neither the BN nor the SVM has shown any improvement. Table 6. Result of Predictive Power Test for Apache HTTP Server Dataset

On the other hand, when the other subset of the metrics, selected by the wrapper subset selection method was used, RF has reported the best performance compared to the other classifiers and the best among any the other features selection technique the RF used. It should be noted that WRP conducts a search for a good subset using a classifier. We applied the four classifiers as a part of WRP in order to select the best combination of metrics. However, as the other classifiers, SVM and LR, did not show better results than NB and RF, we only reported the NB and RF results. When the subset selected by the PCA method was used, however, LR has reported the best performance.

We have observed that the feature selection technique has an impact on the classifiers performances. More precisely, the LR has its best performance when PCA technique has been used while the NB, RF, and SVM has their best performance when the WRP has been applied. In addition, we have observed that the RF recalls value did not score below 70%. Besides, even though the SVM recall value has improved when the metrics subset selected by the WRP was used, its FPR remains above 50. Moreover, when the PCA technique was used, only LR reports a good performance and the other three classifiers either remained the same (RF and SVM) or performed their worst (NB).

H5: It can be concluded that there is a combination of metrics that significantly predicts vulnerabilities that have an exploit using Apache dataset.

Linux Kernel Dataset

The metrics predictive power results for the Linux Kernel dataset show that none of the classifiers has a 70 % recall value. However, let us investigate if any of the classifiers has a recall score of at least 50%. When the whole metrics have been used, none of the classifiers has a recall score of 50%. However, when the CFS feature selection technique has been used, RF has a 50% recall

Table 7.7: Result of Predictive Power Test for Apache HTTP Server Dataset

Model	Performance Measures	All metrics (%)	Correlation-Based (CFS) (%)	Wrapper Subset (WRP) (%)		PCA (%)
				BN	RF	
Logistic Regression	Recall	60	**73**	73	67	**74**
	Precision	60	**73**	72	64	**75**
	F-Measure	53	**73**	73	64	**74**
	Accuracy	60	**73.3**	73.3	66.6	**73.3**
	FPR	50	**33**	38	52	**28**
Naïve Bayes	Recall	70	70	**77**	80	63
	Precision	84	84	**79**	81	72
	F-Measure	70	70	**77**	80	64
	Accuracy	70	70	**76.7**	80	63.3
	FPR	15	15	**22**	20	28
Random Forest	Recall	**77**	70	73	**83**	73
	Precision	**76**	69	72	**84**	73
	F-Measure	**76**	70	73	**84**	73
	Accuracy	**76.6**	70	73.3	**83.3**	73.3
	FPR	**32**	40	38	**18**	33
Support Vector Machine	Recall	67	67	73	73	67
	Precision	44	44	81	81	44
	F-Measure	53	53	67	67	53
	Accuracy	66.7	66.7	73.3	73.3	66.7
	FPR	67	67	53	53	67

score. It should be noted that while the NB and LR have improved when the CFS was used, compared when using the whole metrics, SVM performed worse than when it used the whole metrics. On the other hand, when the WRP has been used, only NB and RF has their recall score improved. When the PCA technique has been used, however, only NB and SVM have shown a recall score above 50%. Though using the feature selection techniques has slightly improve the performance of the chosen classifiers, their FPRs have a score close to or greater than 50%. This shows that the classifiers have difficulty to learn from this dataset and hence behaved almost randomly. It should be noted that the SVM, when the PCA was used, has performed the best compared to the other classifiers. H5: We can conclude that there is a combination of metrics that significantly predicts vulnerabilities that have an exploit using Linux Kernel dataset but at low recall score.

7.4.5 Threats to Validity

In this paper we have considered the datasets for only two products, the Apache HTTP server and Linux Kernel. However, they are both very significant examples. The Apache HTTP server has more than 169 belonging to different categories. Besides, its line of code varies between 50,712 LOC to 358,633 LOC. We recognize that the number of vulnerabilities that have an exploit reported for them in Apache HTTP server is low. However, we are just scratching the surface based on what is available. On the other hand, Linux Kernel has larger number of vulnerabilities,

more than 1200. Its size in line of code has ranged from 10,239 LOC to 15,803,499 LOC. It also has a greater variety of vulnerabilities. As there are other potential factors that can influence the probability of development of an exploit for a vulnerability that have not been examined in this study. Finally, exploits that have not yet been publically reported were not considered in our study.

7.5 Discussion

One of our main observations is that some metrics have a good discriminative and predictive power for Apache dataset. However, they do not have significant discriminative and predictive power for the Linux Kernel dataset. Moreover, we also observed that, unlike in Apache dataset, some metrics such as CountPath, values have been found to be higher for vulnerabilities with exploit in Linux dataset. One possible reason could be the value of the target. The exploits developers may be willing to exploit vulnerabilities in an operating system even if it requires more effort because having a root access could be worth the effort. Besides, when compared to Apache, Linux Kernel has more exploits. This shows that this might be because Linux Kernel is a more valuable target for the attackers. To verify this, we looked into the initial release dates of the both products and we found that the difference in age is not very significant (Linux is 23 years old and Apache 20 years old), and also looked into the usage statistics and market share data and we found Apache market share is around 57% [123] whereas Linux Kernel is about 52.4 % [123].

Based on the Apache dataset, we have also observed that when a function is vulnerable and has an exploit, its SLOC, CYC, and CountPath values have been found to be lower than the vulnerable functions without an exploit. A similar result has been observed by [117] when they try to predict the existence of a vulnerability. It seems that the attackers favoring simpler vulnerable targets, especially when the goal is to deny a service, such as the one provided by Apache, using the least effort. Using the WRP as a method to select a combination of metrics has the best impact on the classifiers performance. However, in [117], security domain knowledge metrics have to be considered in order for the vulnerability prediction performance to be enhanced.

7.6 Related Work

In this section, we summarize related works based on their approach: works that address vulnerabilities exploitability, the studies that use software metrics to predict vulnerability location and existence, and works that use the graph-based metrics.

CVSS Metrics: CVSS metrics are the de facto standard for measuring the severity of vulnerabilities [4]. CVSS Base Score measures severity based on exploitability (the ease of exploiting vulnerability) and impact (the effect of exploitation). However, CVSS exploitability measures have some limitations. First, they assign static subjective numbers to the metrics based on expert knowledge. In contrast, we focus on reducing subjectivity in assessing vulnerability exploitation by basing our analysis on software attributes that can be objectively derived from the source code. Second, two of CVSSs factors (Access Vector and Authentication) have the same value for almost all vulnerabilities [83]. Third, there is no formal procedure for evaluating the third factor (Access Complexity) [[4]. Consequently, it is unclear if CVSS considers the software structure and properties as a factor.

Assessing vulnerability exploitability: Gegick et al. [124] argued that vulnerabilities that are located in low risk areas of the code should be prioritized differently from the ones that are located in high risk areas of the code. Their results show that the combined usage of the internal metrics has predicted the attack-prone components with a high accuracy and zero false negative rates. While the authors granularity analysis was at the component level ours is at the function level, which might reveal some more important information [125]. Moreover, we used different internal code-level metrics. In addition, they used reported security failures to identify a component as an attack prone and a non-attack-prone. In contrast, we used the availability of exploit to identify a vulnerable function as either exploited or not exploited function.

Bozorgi et al. [97] proposed a Machine Learning and Data mining technique that uses features mined from known vulnerability reports to predict the possibility of vulnerability exploitability. They compare their results with the CVSS Exploitability metrics and found that their approach can classify vulnerability exploitability better than the CVSS. We consider the relationship between

some software internal metrics, which are extracted from the source code, and the availability of exploits. This is particularly important for newly released software where vulnerability reports are not available.

Allodi and Massacci in [83, 98] have proposed the black market as an index of risk of vulnerability exploitation. Their approach assesses the risk of vulnerability exploitation based on the volumes of the attacks due to the vulnerability exploits sold in the black market. In contrast, in this paper we try to investigate the relationship between software metrics and the availability of exploits, as data about vulnerabilities attacked in the wild is not always available. This could help software developers predict vulnerabilities exploitation on the development side instead of the deployment side.

Using software metrics: Several researchers have studied the possibility of using software metrics to predict vulnerable entities (components, classes, modules, files, and functions/methods) or the existence of vulnerabilities. Shin and Williams [102, 103] investigated the possibility of using complexity metrics as predictors for the location of security problems. Chowdhury and Zulkernine [104] investigated the usability of complexity, coupling and cohesion metrics as predictors of vulnerabilities location. However, in [35], Zimmermann et al. studied several software metrics including code complexity and dependency as predictors for the existence of vulnerabilities. Our work, on the other hand, focuses on predicting the exploitability of a vulnerability.

Using Graph-based Metrics: Bhattacharya et al. [94] studied the possibility of applying graph-based approaches to software engineering tasks. Using the source code, they construct a graph model and use graph metrics to measure some properties. Their results show that graph metrics can detect significant structural changes, and can help estimate bug severity, prioritize debugging efforts, and predict defect-prone releases. In this paper, we use different graph metrics and investigate whether they are correlated with a vulnerability severity rather than a bug severity. Besides, we use the availability of an exploit to identify the severity of a vulnerability.

7.7 Conclusions and Future Work

In this study, we investigated the possible relationship between the metrics and the existence of a vulnerability exploit. We studied 183 vulnerabilities and mapped them to their locations at the function level. We then characterized these functions using eight software metrics. The metrics have been evaluated for their discriminative and predictive power. The results show that the difference between a vulnerability that has no exploit and a vulnerability that has an exploit can be characterized to some extent using software metrics known for characterizing the presence of vulnerabilities for some of the products. However, the study shows that predicting exploitation of vulnerabilities is more complicated than predicting the presence of vulnerabilities and thus using metrics that consider security domain knowledge is important for enhancing the performance of a vulnerability exploitation prediction effort.

Even though the two selected applications have a rich history of reported vulnerabilities, considering software with a different Even though the two selected applications have a rich history of reported vulnerabilities, considering software with a different domain, such as an open source browser like Firefox, increases the size of the dataset and that might reveal significant information. Improving the classifiers performance and capturing vulnerabilities exploitability may require further empirical investigations of software metrics specifically applicable to the security realm. Thus, further research is needed which considers the metrics related to attack surface [69], reachability and dangerous system calls metrics [12, 9], graph-based metrics [94], and static analysis tool warnings metrics [124]. Moreover, using an alternative approach such as a text mining technique [126] to predict vulnerability exploitability might lead to an interesting results.

Identification of previously unknown (i.e. zero-day) vulnerability take considerable expertise and effort using fuzzers, and many of the resulting vulnerabilities may pose little risk if no exploits are written for them. If the methods considered here can be extended for identifying code which is more likely to have an exploited vulnerability, the vulnerability testers can save considerable time.

Chapter 8

Validating CVSS Using Availability of Exploits

In this chapter, the exploitability factor of CVSS Base metrics and Microsoft Rating system will be studied. Using the presence of actual exploits, the two exploitability metrics performance has been compared and evaluated. The results show that exploitability metrics in CVSS and Microsoft do not correlate strongly with the existence of exploits (ground truth), and have a high false positive rate. The high false positive rate (many number of high severity vulnerabilities without a reported exploits or attacks) could be a result of applying defensive mechanism such as Vulnerability reward programs or exploit mitigation techniques.

8.1 Introduction

Evaluating the risk associated with software vulnerabilities is crucial. Risk evaluation involves assessment of appropriate metrics. A security metric is a quantifiable measurement that indicates the level of security for an attribute of a system [67]. In addition to FIRST, an international confederation, some of the major software developers have developed rating systems for assessing the risk of software vulnerabilities. The rating systems include: CVSS [4] by FIRST, Microsoft [127], IBM ISS X-Force [128], Symantec [129], etc. Each one of them rates vulnerabilities risk (severity) based on varieties of metrics and obtain a single overall score by assessing these metrics.

The accuracy of such rating systems is very important as they are intended to help decision makers in resource allocation, patch prioritization, program planning, risk assessment, and product and service selection. According to Verendel [70], the lack of validation and comparisons among such metrics makes their usability risky. To that end, in this study, we compare and evaluate the performance of the CVSS Base metrics and Microsoft Rating systems. The system using the CVSS Base metrics has been selected because it is the de facto standard that is currently widely

used to measure the severity of individual vulnerabilities. On the other hand, the latter has been chosen because Microsoft software vulnerabilities have been evaluated using both Microsofts own metrics and CVSS Base metrics and that makes their comparison feasible. Besides, it performs in-depth technical analysis of the vulnerabilities and this helps in comparing the effectiveness of the technical analysis approach (Microsoft) with the expert opinions approach (CVSS). The presence of actual exploits is used for their comparison.

In this study, we have examined 813 vulnerabilities of Internet Explorer browser and Windows 7 operating system. The two software systems have been selected because their vulnerabilities severity has been measured using the two selected rating systems, their rich history of publicly documented vulnerabilities, and their diversity in size and functionality.

This chapter is organized as follows. Section 9.2 presents the related work. In Section 9.3, the background of the vulnerabilities, vulnerability databases, and the exploit database are discussed. In the following section, the selected vulnerabilities rating systems are discussed. In sections 9.5, the selected datasets are presented. In section 9.6, the applicability of CVSS and MS-Exploitability metrics is examined. Section 9.7 presents the discussion. Finally, concluding comments are given and the issues that need further research are identified.

8.2 Related Work

A few researchers have started to examine CVSS critically. Bozorgi et al. [97] have studied the exploitability metrics in CVSS Base Score metric. They have argued that the exploitability measures in CVSS Base Score metric cannot tell much about the vulnerability severity. They attributed that to the fact that CVSS metrics rely on expert knowledge and static formulas. They have proposed a machine learning and data mining technique that attempts to predict the possibility of vulnerability exploitation. Bozorgi et al. have used the distributions resulting from the two approaches for evaluation. In contrast, in this paper, we evaluate the performance of CVSS exploitability metrics using well defined performance measures and using the presence of actual exploits to compare it with the performance of MS-Exploitability metric. In addition, we take into

consideration the type of software when comparing the performance of the two metrics, as that might reveal some significant insight.

Allodi and Massacci in [[83, 98] have proposed the black market as an index of risk of vulnerability exploitation. Their approach assesses the risk of vulnerability exploitation based on the volumes of the attacks due to the vulnerability exploits sold in the black market. In their study, they conducted a thorough analysis of the CVSS Base metrics: exploitability and impact metrics. They compared CVSS metrics performance against the existence of exploits in the EDB, Symantecs Attack Signature (wild), and OSVDB by using sensitivity and specificity measures. In contrast, in this paper, we compare CVSS exploitability metrics performance with MS-Exploitability metric.

Eiram in [130] has reviewed the value of the rating systems of the Microsoft Exploitability Index, Adobe Priority Rating system, and CVSS exploitability Temporal metric. The aim was determining if these rating systems are meeting the goal of easing prioritization of applying security updates. To evaluate the rating systems, Eiram counted the number of vulnerabilities that have a high severity value (1) and used it as a method of evaluation. He has suggested that this number should not be high. He only used vulnerabilities that were reported in 2012. In this paper, however, we choose to evaluate CVSS exploitability metrics instead of CVSS exploitability temporal metric because the former is the one that is always reported in well-known vulnerability databases such as NVD. Moreover, we do not only use a larger dataset, but also use well defined performance measures. Moreover, we are using the existence of the exploit as a method of evaluation instead of counting the vulnerabilities that have a high severity value.

8.3 Vulnerability Rating Systems

Recently, several rating systems for assessing the severity of computer system security vulnerabilities have been developed. The rating systems include: CVSS [4], Microsoft [127], IBM ISS X-Force [128], Symantec [129], etc. Each one of them rates vulnerabilities severity based on varieties of metrics and assigns a single overall score by assessing these metrics. They are intended to help decision makers to patch prioritization and risk assessment. In this section, CVSS Base metrics and Microsoft Rating systems that are selected for this study are briefly introduced. The

two approaches differ significantly. CVSS depends on the opinions of experts whereas Microsofts approach depends on the technical factors it considers significant.

8.3.1 CVSS Base Metrics

CVSS Base Score measures severity based on exploitability (the ease of exploiting a vulnerability) and impact (the effect of exploitation) as shown by the following [4]:

$$\textbf{Base score } = Roundto1decimal[(0.6Impact) + (0.4Exploitability) - 1.5]f(Impact)$$

The formula for the base score, as well as for the exploitability and impact sub-scores are based on expert opinion and are not based on formal derivations. The constants are chosen to yield the maximum values of 10. The base score is rounded to one decimal place and it is set to zero if the impact is equal to zero regardless of the formula. The CVSS scores for known vulnerabilities are readily available in the majority of public vulnerability databases. The CVSS score is a number in the range [0.0, 10.0]. This score represents the intrinsic and fundamental characteristic of a vulnerability and thus the score does not change over time. The two CVSS sub-scores exploitability and impact also range between [0.0, 10.0]. CVSS score from 0.0 to 3.9 corresponds to Low severity, 4.0 to 6.9 to Medium severity and 7.0 to 10.0 to High severity.

The impact sub-score measures how a vulnerability will directly affect an IT asset as the degree of losses in Confidentiality (IC), Integrity (II), and Availability (IA) as is shown by the following:

$$\textbf{Impact} = 10.41 \times (1 - (1 - I\_C) \times (1 - I\_I) \times (1 - I\_A))$$

The impact sub-scores, on the other hand, are all assessed in terms of None (N), Partial (P), or Complete (C) by security experts and assigned one of the mentioned qualitative letter grades. Exploitability, on the other hand, is assessed based on three metrics: Access Vector (AV), Authentication (AU), and Access Complexity (AC) as is shown by the following:

$$\textbf{Exploitability} = AV \times AU \times AC$$

The AV reflects how a vulnerability is exploited in terms of local (L), adjacent network (A), or network (N). The AC measures the complexity of an attack required to exploit the vulnerability

(once an attacker has gained an access to a target system) in terms of High (H), Medium (M), or Low (L). The AU counts the number of times an attacker must authenticate to reach a target (in order to exploit a vulnerability) in terms of Multiple (M), Single (S), or None (N). The CVSS system uses lookup tables to provide the numerical values needed for the subscores.

8.3.2 Microsoft Rating System

Microsoft (MS) rating system measures vulnerabilities risk based on two variables: Impact and Probability [127]. The latter means the potential effect of a vulnerability being exploited, and the former means the likelihood of that exploitation taking place. These two variables are assessed using severity rating and Exploitability Index.

The impact factor, on one hand, is captured using the severity rating that indicates the ”worst case” scenario of an attack that exploits a vulnerability. The Impact factor can take one of the following values: Critical (A vulnerability whose exploitation could allow code execution without user interaction), Important (A vulnerability whose exploitation could result in compromise of the confidentiality, integrity, or availability of user data, or of the integrity or availability of processing resources), Moderate (Impact of the vulnerability is mitigated to a significant degree by factors such as authentication requirements or applicability only to non-default configurations), and Low (Impact of the vulnerability is comprehensively mitigated by the characteristics of the affected component).

On the other hand, the probability factor is assessed using the Exploitability Index that measures the likelihood that a specific vulnerability would be exploited within the first 30 days after bulletin release. The Exploitability Index can be assigned a score 1, 2, and 3 for any vulnerability with a severity rate of Important or Critical. Here, 1 means Consistent exploit code is likely, 2 means Inconsistent exploit code is likely, and 3 means Function exploit code unlikely.

8.4 Datasets

In this study, the data about vulnerabilities and exploits of Microsoft Windows 7 and Internet Explorer (IE) during the period January 2009 to October 2014 were collected. The reason why

Table 8.1: Internet Explorer and Windows 7 Vulnerabilities

Software	Exploit Exist	No Exploit Exist	Total Each
IE	33	436	459
Windows 7	52	302	354
Total All	85	738	813

Table 8.2: Unselected Vulnerabilities in Microsoft Internet Explorer

CVE-2010-0808	CVE-2010-3327	CVE-2010-3342	CVE-2010-3348
CVE-2012-0010	CVE-2011-1244	CVE-2011-1246	CVE-2011-1258
CVE-2011-1962	CVE-2011-2383	CVE-2011-3404	CVE-2011-0038
CVE-2012-0168	CVE-2012-1872	CVE-2012-1882	CVE-2013-3126
CVE-2013-3186	CVE-2013-3192	CVE-2014-2817	CVE-2014-4123

we considered collecting data starting from 2009 is because the Microsoft Exploitability Index is introduced in October 2008 and has been included in Microsoft Bulletin starting 2009. This data was collected as follows. First, the vulnerabilities and their metrics' values were collected from NVD [16] and Microsoft Security Bulletin [131]. Second, the exploits were collected from EDB [22]. Table 8.1 shows the number of the selected vulnerabilities and their exploits.

It should be noted that the total number of the IE vulnerabilities is 482. However, out of the 482, 23 vulnerabilities were not selected because we could not find information about their CVSS Base metrics values or MS Exploitability Index values.

- Three out of the 23 vulnerabilities (CVE-2014-4066, CVE-2014-4112, and CVE-2014-4145) could not be found in the NVD even though they have CVE number.
- For the remaining 20 vulnerabilities, we could not find their MS Exploitability Index values. These vulnerabilities are shown in Table 8.2.
- None of these unselected vulnerabilities has an exploit.

It should also be noted that the total number of vulnerabilities of Windows 7 is 380. However, out of the 380 vulnerabilities, 26 vulnerabilities were unselected because we could not find their MS Exploitability Index values. Out of the 26 vulnerabilities, four vulnerabilities that have an exploit were removed (CVE-2010-1890, CVE-2010-1887, CVE-2010-2554, and CVE-2010-3227). Table 8.3 shows these 26 vulnerabilities.

Table 8.3: Unselected Vulnerabilities in Microsoft Windows 7

CVE-2010-0252	CVE-2010-0481	CVE-2010-0811	CVE-2010-1890
CVE-2010-1887	CVE-2010-2554	CVE-2010-3227	CVE-2010-0811
CVE-2011-1971	CVE-2011-1978	CVE-2011-2002	CVE-2011-2004
CVE-2011-3415	CVE-2012-0156	CVE-2012-0174	CVE-2012-1850
CVE-2012-1851	CVE-2012-2531	CVE-2013-0013	CVE-2013-1291
CVE-2013-1293	CVE-2013-1336	CVE-2013-1337	CVE-2013-3172
CVE-2013-2556	CVE-2014-0295		

The attributes of every selected vulnerability were collected using the following steps.

1. From Microsoft Security Bulletin, the vulnerabilities CVE numbers for Windows 7 and IE were collected.

2. Next, for every existed CVE number in Microsoft Security Bulletin, we collected the vulnerability severity rating and Exploitability Index values.

3. Then, we searched for the same vulnerabilities CVE numbers found in Microsoft Security Bulletin in the NVD.

4. After that, for every vulnerabilitys CVE number found in the NVD, the CVSS impact Subscore and exploitability Subscore values were collected.

5. Lastly, for every selected vulnerability we used the CVE number to verify whether it has an exploit reported in the EDB or not.

Table 8.4 shows a part of the selected vulnerabilities because showing the whole datasets is limited by the number of pages allowed. Unlike CVSS Base Score, Microsoft rating system does not provide the total value of a vulnerability risk, but rather it provides an individual value and let the person in charge to make the decision based on the provided values. It has been noticed that for some vulnerabilities the severity rating and the Exploitability Index are assigned a combination of values instead of one value, for example Critical/Moderate or Important / Low, or, 1/2 or 1/3. This is because some vulnerabilities exist in older versions of Microsoft products that are no longer supported by Microsoft security updates, and hence are assigned a higher severity rating

Table 8.4: The obtained measures of Microsoft Rating System and CVSS Base Score Metrics

CVE	MS Security Bulletin	Microsoft		CVSS Base Score			Exploit Existence
		Severity Rating	*Exploitability Index*	*Impact Subscore*	*Exploitability Subscore*	*Total*	
CVE-2009-1547	MS09-054	Critical	2	10	8.6	9.3	EE
CVE-2010-0492	MS10-018	Critical/Moderate	1	10	8.6	9.3	NEE
CVE-2011-1992	MS11-099	Important /Low	3	2.9	8.6	4.3	NEE
CVE-2012-1858	MS12-037	Important /Low	3	2.9	8.6	4.3	EE
CVE-2013-1312	MS13-037	Critical/Moderate	2	10	8.6	9.3	NEE
CVE-2014-4141	MS14-056	Critical/Moderate	1	10	8.6	9.3	NEE

or Exploitability Index values. On the other hand, the new releases of Microsoft products have a regular security updates and hence are assigned a lower severity rating or Exploitability Index values.

8.4.1 Analysis of Vulnerabilities and Exploits

Internet Explorer Dataset

Figure 8.1 shows the distribution of the vulnerabilities and their exploits of IE from 2009 to 2014. As can be seen, the year 2010 has the highest percentage of exploit, which is a result of dividing the number of reported exploits per year by the total number of vulnerabilities reported in that year. Besides, a noticeable increase in the number of reported vulnerabilities has been noticed in 2013 and 2014. This could be attributed to Microsoft Bounty Programs that has started in 2013. It should be noted that there are more vulnerabilities than exploits. According to [132], Cybercriminals are more likely to exploit vulnerabilities that dont need any special conditions, or that offer particularly dangerous opportunity to execute malware on the compromised computer.

The total percentage of the exploits in relation to the number of reported vulnerabilities is 6.8. To evaluate this number based on IE counterparts web browsers, we collected the vulnerabilities and exploits from 2009 to 2014 for Google Chrome, Firefox, and Apple Safari from NVD and EDB and the result is shown in Figure 8.2. As can be seen from Figure 8.2, the total number of reported vulnerabilities in Chrome and Firefox are the highest and this could be attributed to their

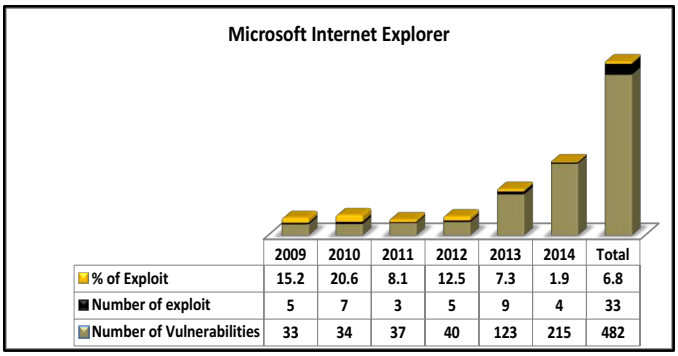


Figure 8.1: Distribution of the number of reported vulnerabilities and exploits and percentage of exploits.

vulnerability Bounty Program. Even though Chrome has the highest number of vulnerabilities, it has the lowest exploit percentage. This could be attributed to the quick patching, which in turn complicate the exploitation process. Although investigating this observation is really important, it is considered beyond the scope of this research.

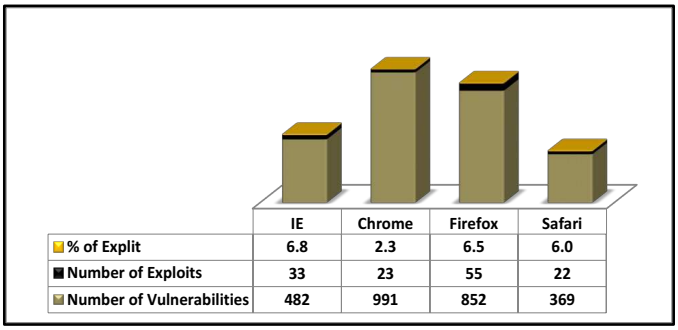
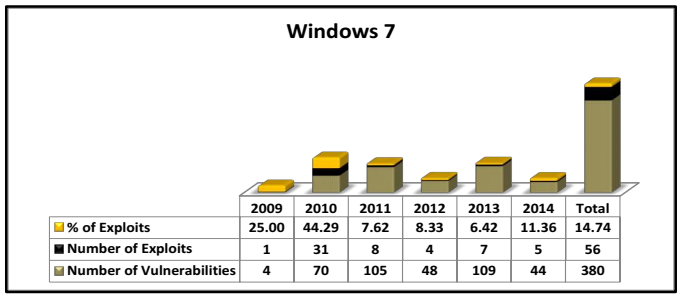


Figure 8.2: Web Browsers distribution of the number of reported vulnerabilities and exploits and the percentage of exploits.

Windows 7 Dataset

The distribution of the vulnerabilities and exploits for Windows 7 is shown in Figure 8.3. As can be seen, 2010 has the most number of exploits, 31. We did some investigation and found out that some of the vulnerabilities that were discovered in Windows Vista are inherited by Windows 7 and this is because of the code reuse concept. Hence, Windows Vista vulnerabilities exploit can also be used to exploit the inherited vulnerabilities in Windows 7. However, more vulnerability was reported in 2011 and 2013. According to [133], the reason behind the increase in 2011 is

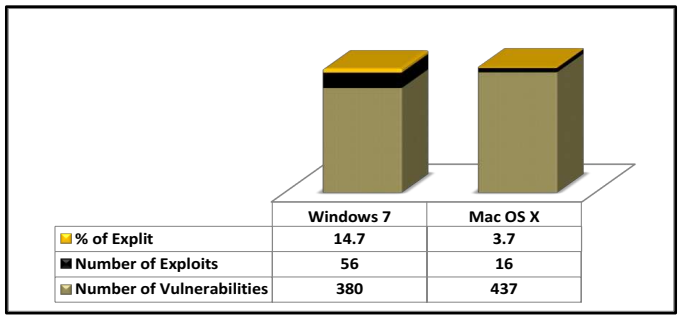
an effort of one researcher who looked at win32k.sys and discovered 20 vulnerabilities in 2010 and 59 in 2011. On the other hand, the increase in 2013 could be explained by Microsoft Bounty Programs.



	2009	2010	2011	2012	2013	2014	Total
% of Exploits	25.00	44.29	7.62	8.33	6.42	11.36	14.74
Number of Exploits	1	31	8	4	7	5	56
Number of Vulnerabilities	4	70	105	48	109	44	380

Figure 8.3: Windows 7 distribution of the number of reported vulnerabilities and exploits and the percentage of exploits.

The total percentage of the exploits in Windows 7 is 14.74. To evaluate this number, we collected the reported vulnerabilities and exploits from 2009 to 2014 for the Mac OS X from NVD and EDB databases. The reason why we selected Mac OS X is because of its market share and popularity. Figure 8.4shows that even though the number of the reported vulnerabilities in Mac OS X is more than Windows 7, the former has a higher number of exploit percentages. One possible explanation could be that Windows 7 has larger market share (40.81%) than Mac OS X (6.64%) in Desktop and Laptop computers [134], which could cause more impact and in turn attracts attackers as a target.



	Windows 7	Mac OS X
% of Explit	14.7	3.7
Number of Exploits	56	16
Number of Vulnerabilities	380	437

Figure 8.4: Microsoft Windows 7 and Mac OS X number of reported vulnerabilities and exploits and the percentage of exploit.

8.4.2 Analysis of Vulnerabilities' Rating Systems

The selected rating systems, CVSS Base Score and Microsoft Rating system, measure vulnerability severity based on two factors the impact of exploitation and the possibility of exploitation. In this section we will first explore their exploitability values and then investigate their impact measures using the two selected datasets.

Exploitability Factor

Figure 8.5 shows the distribution of the exploitability values for CVSS and Microsoft for the IE dataset. For CVSS exploitability score, almost all vulnerabilities have a value between eight and nine (0.97, relative frequency), whereas for Microsoft Exploitability Index the values are almost one (0.82, relative frequency). Even though Microsoft Exploitability Index shows some variations, looking at the boxplot in Figure 8.5, it is clear that the distribution of the two metrics is indistinguishable for IE dataset. The median is 8.6 for CVSS and 1 for Microsoft. This shows that the exploitability factor for both metrics is almost a constant and not a variable. It should be noted that in the CVSS boxplot the minimum and maximum points are plotted as outliers by the software we used to create them (XLSTAT) whereas in Microsoft boxplot only the maximum point is plotted as an outlier.

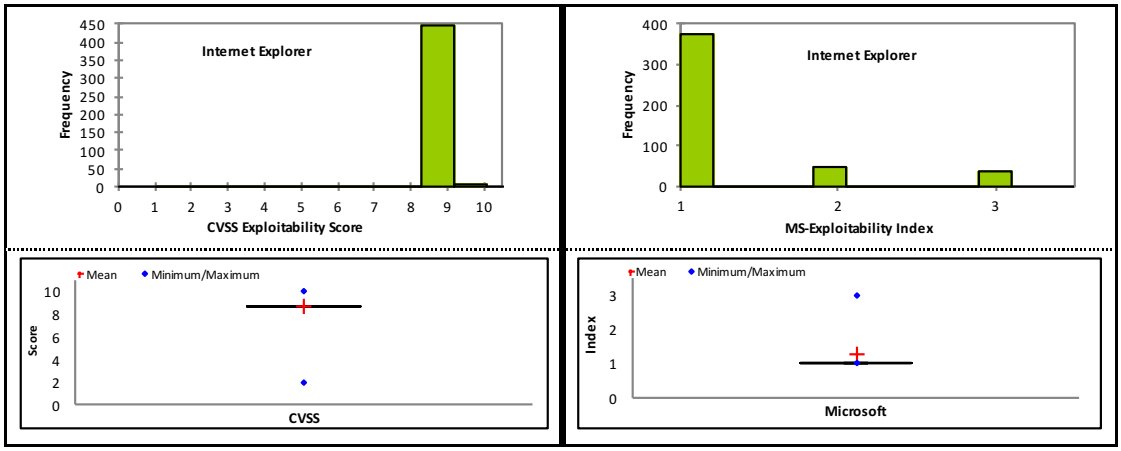


Figure 8.5: The histograms on the top represent the frequency distribution of the CVSS and Microsoft Exploitability values. The boxplots on the bottom describe the distribution of values around the median, which represented by a horizontal line.

Figure 8.6 illustrates the distribution of the exploitability values for CVSS and Microsoft for Windows 7 dataset. Unlike IE dataset, the distribution varies for the two metrics. For the CVSS Exploitability scores, almost half of vulnerabilities have a Low (0 to 3.9) exploitability values, whereas the other half has a High (7 to 10) exploitability values. Only a few vulnerabilities have a Medium (4 to 6.9) exploitability values. On the other hand, for the Microsoft Exploitability Index almost all vulnerabilities have an exploitability value between 1 and 2, which means exploit is likely. Only a few have an exploit value 3 (around 59), which means exploit is unlikely. The variability of the exploitability factor for both metrics is apparent in the reported boxplot.

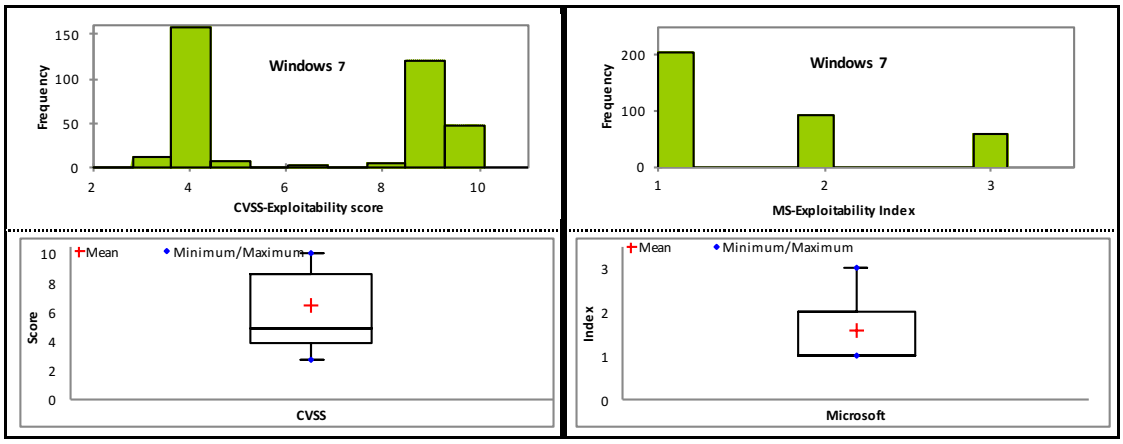


Figure 8.6: The histograms on the top represent the frequency distribution of the CVSS and Microsoft Exploitability values. The boxplots on the bottom describe the distribution of values around the median, which represented by a horizontal line.

To understand why the two metrics show completely different results when applied to different software types datasets, we decompose the CVSS exploitability metrics. As can be seen in Table 8.5, we find that almost half of the vulnerabilities (172) in the Windows 7 dataset have a Local AV value and that has led to a Low CVSS exploitability subscore. On the other hand, there are only three vulnerabilities in the IE dataset that have a Local AV value. It has also been noted that the AV has a significant effect on the CVSS exploitability subscore, regardless of the values of the AU and AC. This can be clearly observed from IE dataset where almost all the vulnerabilities have a Network AV value and hence have been assigned High CVSS exploitability Subscore. It has also been observed that almost all the vulnerabilities in Windows 7 (94.19%) that have a Local

Table 8.5: CVSS Exploitability Metrics Subscore for IE and Windows 7

Exploitability Metrics	Value	IE (%)	Windows 7 (%)
AV	Network	99.35	51.41
	Adjacent	0	0
	Local	0.65	48.59
AU	None	100	95.76
	Single	0	3.95
	Multiple	0	0
AC	High	1.31	1.98
	Medium	98.04	37.29
	Low	0.65	60.45

AV value also have a Low AC value. This explains the increase in the number of vulnerabilities (60.45%) that have a Low AC.

Impact Factor

Figure 8.7 shows the distribution of the Impact values for CVSS and Microsoft for the IE dataset. For CVSS Impact score, almost all vulnerabilities have a value between nine and ten (0.90, relative frequency). On the other hand, Microsoft Impact rating score values are one for almost all vulnerabilities (0.87, relative frequency). It should be noted that, for the sake of comparison, Microsoft Impact values have been mapped into numbers as follows: Critical=1, Important=2, Moderate=3 and Low=4. The medians as shown in boxplot are ten for the CVSS impact values and one for Microsoft Impact values. This shows that the impact values for both metrics for almost all vulnerabilities are high. This can be explained by the fact that 428 (93.24%) vulnerabilities are of the type Execute Code.

As it can be seen from Figure 8.8, more than half of the vulnerabilities have been assigned a high (10) CVSS impact values (0.72, relative frequency), whereas more than half of the vulnerabilities have been assigned an Important (2) Microsoft impact values (0.68, relative frequency). To understand the difference in the impact values between the two metrics, we decomposed the CVSS impact Metrics as shown in Table 8.6. The CIA values are dominated by the value complete for the majority of vulnerabilities especially for IE. We also find that around 270 vulnerabilities (76.27%) that have been assigned a value complete for their C, I, and A metrics in Windows 7. We also find

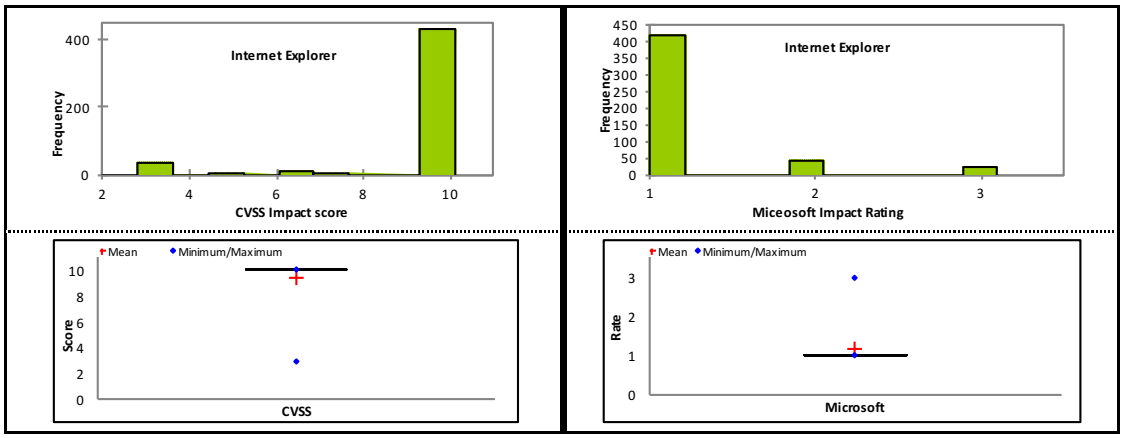


Figure 8.7: The histograms on the top represent the frequency distribution of the CVSS and Microsoft Impact values. The boxplots on the bottom describe the distribution of values around the median, which represented by a horizontal line.

that there are 180 vulnerabilities of the type Gain Privilege and 169 of them have Microsoft impact value 2 (93.89%). The reason why Microsoft has relaxed the impact value of this type of vulnerabilities is possibly because this type of vulnerabilities depends on user configurations, which is most of the time have fewer user rights on the system.

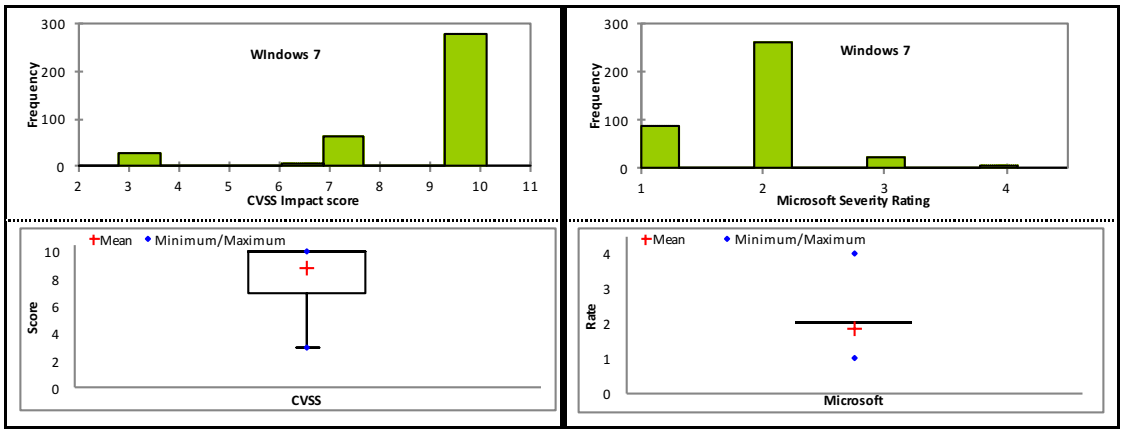


Figure 8.8: The histograms on the top represent the frequency distribution of the CVSS and Microsoft Impact values. The boxplots on the bottom describe the distribution of values around the median, which represented by a horizontal line.

Table 8.6: CVSS Impact factor Metrics for IE and Windows 7

Software	Value	Complete %	Partial %	None %
IE	**Confidentiality (C)**	92.81	5.66	1.53
	Integrity (I)	92.59	3.05	4.36
	Availability (A)	92.81	1.96	5.23
Windows 7	**Confidentiality (C)**	87.29	3.67	8.19
	Integrity (I)	76.27	2.54	20.34
	Availability (A)	81.92	2.54	14.69

8.5 Validation of CVSS and MS-Exploitabilty Metrics

Since data is available about the existence of exploits, we can only evaluate the Microsoft Exploitability Index metric with the CVSS exploitability metrics based on the availability of exploits. Figure 8.9 shows the CVSS and Microsoft Exploitability measures for IE and Windows 7 vulnerabilities against the Existence of Exploit (EE) and No-Exploit Existence (NEE). For the IE dataset, the CVSS exploitability median is 8.6 for both vulnerabilities with EE and for those with NEE. Besides, the median for the Microsoft Exploitability Index is 1 for both classes (EE, NEE). On the other hand, for the Windows 7 dataset, the CVSS exploitability median is 3.9 for vulnerabilities with NEE and 8.6 for vulnerabilities with EE. Moreover, the median for the Microsoft Exploitability Index is 1 for both classes (EE, NEE).

8.5.1 Methodology

To evaluate the performance of these two metrics, we used statistical measures termed sensitivity, precision, and F-measure. These measures are explained using a confusion matrix as shown in Table 8.7. The confusion matrix table shows the actual vs. the predicted results. It should be noted that a vulnerability is considered exploitable if it has a reported exploit, but that does not mean the otherwise. For the two class problem (a vulnerability is either exploitable or not exploitable), the following is defined based on Table 8.7.

- True Positive (TP): the number of the vulnerabilities predicted as exploitable, which do in fact have an exploit.

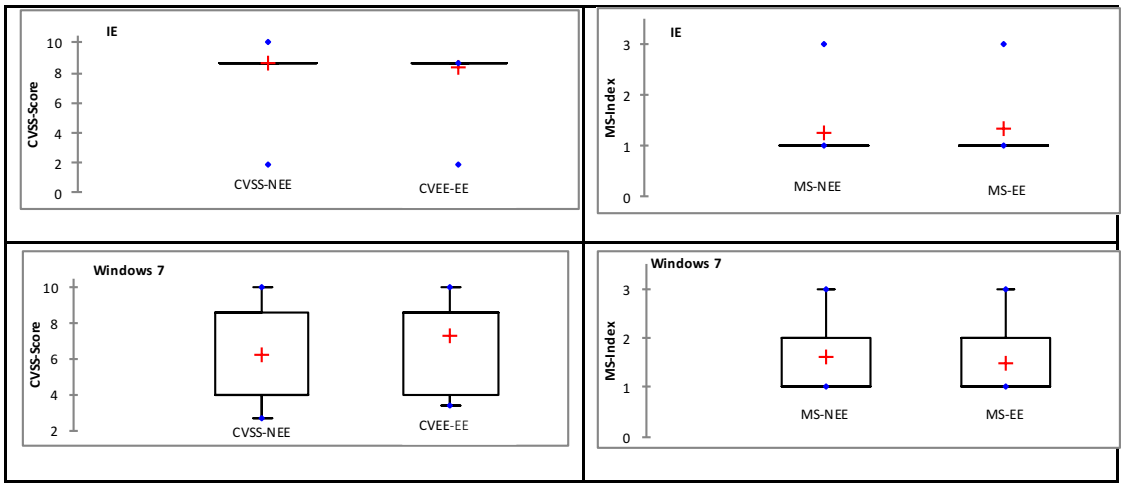


Figure 8.9: Vulnerability Exploitability measures using CVSS in the Left column and MS-Exploitability Index in the right Column compared to Exploit Exist (EE) and No-Exploit Exist (NEE) for IE and Windows 7 datasets.

Table 8.7: Confusion Matrix

Actual	**Prediction**	
	Exploitable	Not Exploitable
Exploitable	TP= True Positive	FN= False Negative
Not Exploitable	FP= False Positive	TN= True Negative

- False Negative (FN): the number of vulnerabilities predicted as not exploitable, which turn out to have an exploit.
- False Positive (FP): the number of vulnerabilities predicted as exploitable when they have no exploit.
- True Negative (TN): the number of vulnerabilities predicted as not exploitable when there is no exploit.

The selected performance measures can be derived as follows.

Sensitivity (Recall)

Sensitivity, which also termed recall, is defined as the ratio of the number of vulnerabilities correctly predicted as exploitable to the number of vulnerabilities that are actually exploitable as

shown by the following:

$$Sensitivity = \frac{TP}{TP + FN}$$

Precision

Precision, which is also known as the correctness, is defined as the ratio of the number of vulnerabilities correctly predicted as exploitable to the total number of vulnerabilities predicted as exploitable as shown by the following:

$$Sensitivity = \frac{TP}{TP + FP}$$

For convenient interpretation, we express these two measures in terms of percentage, where a 100% is the best value and 0% is the worst value. Both precision and sensitivity should be as close to the value 100 as possible (no false positives and no false negatives). However, such ideal values are difficult to obtain because sensitivity and precision often change in opposite directions. Therefore, a measure that combines sensitivity and precision in a single measure is needed. Hence, we will introduce the F-measure in the following section. We believe that it is more important to identify exploitable vulnerabilities even at the expense of incorrectly predicting some not exploitable vulnerabilities as exploitable vulnerabilities. This is because a single exploitable vulnerability may lead to serious security failures. Having said that, we think more weight should be given to sensitivity than precision. Thus, we include F2-measure, which weights sensitivity twice as precision, to evaluate the two metrics.

F-measure

F-measure can be interpreted as the weighted average of sensitivity and precision. It measures the effectiveness of a prediction with respect to a user attached times as much importance to sensitivity as precision. The general formula for the F-measure is shown by the following:

$$F_\beta - Measure = \frac{(1 + \beta^2) \times Precision \times Senetivity}{(\beta^2 \times Precision) + Senetivity}$$

β is a parameter that controls a balance between sensitivity and precision. When β = 1, F-measure becomes to be equivalent to the harmonic mean, whereas when β <1 it becomes more

Table 8.8: Confusion matrix of CVSS exploitability metrics and Microsoft Exploitability Index

Internet Explorer			
CVSS		**Prediction**	
		Exploitable	Not Exploitable
Actual	Exploitable	TP= 32	FN= 1
	Not Exploitable	FP= 423	TN= 3
Microsoft Exploitability Index		**Prediction**	
		Exploitable	Not Exploitable
Actual	Exploitable	TP= 28	FN= 5
	Not Exploitable	FP= 395	TN= 31
Windows 7			
CVSS		**Prediction**	
		Exploitable	Not Exploitable
Actual	Exploitable	TP= 34	FN= 18
	Not Exploitable	FP= 141	TN= 161
Microsoft Exploitability Index		**Prediction**	
		Exploitable	Not Exploitable
Actual	Exploitable	TP= 43	FN= 9
	Not Exploitable	FP= 253	TN= 49

precision oriented. However, when $\beta > 1$, F-measure becomes more sensitivity oriented. In this paper β has been chosen to be 2.

8.5.2 Results

To calculate the above mentioned performance measures we need to obtain the confusion matrix for the two datasets. Using the data about the availability of exploits and the exploitability measures for CVSS exploitability metrics and Microsoft Exploitability Index, the confusion matrix was determined as shown in Table 8.8. Using the values in this matrix, the performance measures have been calculated as shown in Table . It should be noted that there is an imbalance in the two datasets. For instance, there are 33 vulnerabilities with an exploit compared to 436 vulnerabilities without an exploit in the IE dataset, whereas there are 52 vulnerabilities with an exploit and 302 vulnerabilities without an exploit in the Windows 7 dataset. We considered this imbalance in our performance analysis.

For every dataset, we selected all vulnerabilities that have an exploit and at the same time we randomly (using random with replacement technique) selected the same number of vulnerabilities

Table 8.9: Prediction Performance of CVSS Exploitability Metrics and Microsoft Exploitability Index

Software	Performance Measures	CVSS Exploitability Metrics		MS-Exploitability Index	
		Whole Sample (%)	Balanced Sample (%)	Whole Sample (%)	Balanced Sample (%)
IE	**Sensitivity**	97	97	85	85
	Precision	7	50	7	57
	F1-Measure	13	33	12	34
	F2-Measure	27	82	25	77
	False Positive Rate	99.3	97	92.7	64
Windows 7	**Sensitivity**	65.38	65	82.69	82
	Precision	19.43	50	14.53	50
	F1-Measure	29.96	29	24.71	31
	F2-Measure	44.39	62	42.66	73
	False Positive Rate	46.69	62	83.77	81

from those that have no exploit and calculated the performance measures and the results are shown in Table 8.9.

From the IE dataset and when the whole sample is considered, it is clear that the two metrics have a high sensitivity values and that comes at the cost of having a very low precision. It is also apparent that the false positive rate is very high, which is almost a 100% for CVSS exploitability metrics and around 93% for Microsoft Exploitability Index. This makes the two metrics behave like a random predictor. Looking at CVSS exploitability metrics values, the high positive rate can be explained by the fact that almost all the vulnerabilities in IE dataset have a Network AV value and hence have been assigned a high CVSS exploitability value and that has led to assessing those vulnerabilities as exploitable. On the other hand, when the imbalanced sample is considered, it can be seen that the precision of the two metrics has dramatically changed and that is in turn has changed the F1 and F2 measures too, which expected as both of them rely on the precision value. Besides, the false positive rate for Windows Exploitability Index has noticeably reduced.

Looking at the performance of the two metrics using the whole sample of Windows 7 dataset, it is clear that both metrics performed differently when compared to their performance using IE dataset. This difference is more apparent in CVSS exploitability metrics. First, the false positive rate has significantly dropped to less than 50%. This could be explained by the fact that (unlike

the IE dataset where almost all the vulnerabilities have a Network AV value and hence have been assigned a high exploitability value) in Windows 7 almost half of the vulnerabilities have been assigned Local AV values and hence have been assigned a low exploitability value. In other words, CVSS exploitability factor is highly influenced by the AV values. Second, the sensitivity has noticeably dropped and this is 6because there are 18 vulnerabilities that have been assessed as not exploitable and they turn out to have an exploit. However, when the imbalanced sample is taken into account, it can be seen that the precision of the two metrics has noticeably changed and that is in turn has also changed F2 measure. Besides, F1 measure slightly decreased and that is because of the drop in the sensitivity of the CVSS.

From the performance analysis, 6the following has been observed.

- The sensitivity measure of the Microsoft Exploitability Index has not been affected by the change of the type of software.
- The sensitivity measure of CVSS exploitability measure has been noticeably affected by the change of software type. This has led to a change in AV values and that in turn has made CVSS exploitability measure predicts 18 vulnerabilities as not exploitable where is in fact there are exploits for those vulnerabilities.
- Both metrics are very sensitive and that has led to a lower precision and a high false positive rate and this could lead to a waste of resources and effort.
- CVSS exploitability factor is highly influenced by the AV values regardless of the other two factors, AC and AU.
- Taking into consideration the imbalance in the datasets between the number of vulnerabilities with an exploit and those without an exploit has shown an improvement in the precision and F2 measures for the two metrics.
- It is unclear how the Microsoft Exploitability Index is assessed.

8.5.3 Threats to Validity

In this chapter, we have considered the datasets for only two products, Internet Explorer and Windows 7. However, the two selected software have a rich history of reported vulnerabilities and exploits (IE has 436 reported vulnerabilities, 33 with reported exploits, and Windows 7 has 354 reported vulnerabilities, 52 with reported exploits). One of our observations is that the false positive rate is high for both metrics. This could be a result of other factors that have not been considered in this study. Only publicly reported vulnerabilities and exploits have been considered here.

8.6 Discussion

An important question that arises from the results of this study is why both the technical approach (Microsoft) and the expert opinion approach (CVSS) did not perform well? One possible reason could be that some of the chosen metrics do not correlate well with the factors that contribute to vulnerability exploitability in actual reality and hence new metrics are needed to be identified and added to the two rating systems. It will require extensive investigations to identify new metrics that well correlated. There may be some randomness in how potential exploit developers identify the vulnerabilities for which exploit development may be worthwhile. It is also possible that the software developers may work more aggressively towards developing patches for highly exploitable vulnerabilities making it less attractive for external exploit developers to develop exploits. However the results suggest that these may not provide the complete explanation for why the exploitability measures have not performed well. Another possible reason could be that the two metrics did not carefully consider the threat (the external factor) and mainly focus on the internal factors, which are alone not enough to capture the whole risk presented by a vulnerability. Looking at the formal theory of risk, we find that risk is defined as [84]:

$$Risk = Likelihood of an adverse event \times Impact of the adverse event$$

$$Likelihood of an Adverse Event = Threat R\_Vulnerability$$

$$Risk = Threat \times R\_Vulnerability \times Impact$$

What the two rating systems have considered are the R_vulnerability (in the risk theory the term vulnerability is a number given by the probability of an attacks success or an ease of exploitation) and the impact (losses that occur given a successful exploitation) and they assume the threat as either there or not adaptive, whereas a threat (an adversary), unlike accidents or acts of nature, is intelligent and may dynamically adapt to the used defensive measures. Therefore, vulnerabilities are dangerous if and only if someone (adversary) is interested in exploiting them (motive) and has the means (capability) to do it and that is shown by [135].

$$Threat(attacker) = Motive \times Capability$$

A motive is a measure of how far an adversary is willing to go and what he is willing to risk to reach his objectives. A motive can be influenced by the sensitivity of data, desire for monetary gain, or the potential publicity effects of an attack. One way of measuring a motive in cyber security is based on the following factors [136]: a cost of attempting an attack, a payoff of a successful attack, a probability of successfully completing an attack, and a probability of detection. A capability, on the other hand, is the degree to which an adversary is able to execute an attack. To execute an attack, the skills, knowledge, tools, and techniques should be possessed by an adversary. According to [137], the following factors are a key in measuring a capability of a hacker or cracker: group size, history of relevant activity, technical expertise, and target selection. Some of the capability factors have been considered to a limited extent, especially by MS rating system. Quantifying and including the attackers motive and some other capability factors as a part of the two studied rating systems may significantly advance the vulnerability risk assessment by increasing the accuracy and reducing the false positive rate.

8.7 Conclusion and Future Work

This study compares and evaluates the performance of the CVSS Base metrics and Microsoft rating system using 813 vulnerabilities of IE and Windows7. The results show that the two mea-

sures have a very high false positive rate. It was observed that the sensitivity measure of CVSS exploitability metrics is noticeably affected by the software type. Besides, CVSS Exploitability factor is highly influenced by the AV values regardless of the other two factors (AC and AU). However, unlike the CVSS Base metrics where the metrics (factors) used for measuring vulnerabilities risk are provided, Microsoft rating system does not provide such metrics but rather provides the values and their definition. Hence it was hard to conduct a thorough investigation trying to correlate the two sets of metrics.

Even though the two selected software have a rich history of reported vulnerabilities, considering other Microsoft products could increase the size of the dataset and that might reveal significant information. The study suggests that a simple measure of vulnerabilities exploitability using few metrics may not be sufficient. Hence, identifying new metrics that capture attributes that have not been yet considered and adding them to the two rating systems is needed. Younis et al. in [12] have proposed some distinctive metrics based on the software structure. In addition, identifying and including the external factors, such the attacker behavior, to the two selected rating system could improve their precision and reduce their false positive rate.

Chapter 9

Validating CVSS Using Vulnerability Rewards Program

This chapter introduces the vulnerability reward programs (VRPs) as a novel ground truth to evaluate the CVSS Base scores. Having more eyes on the code means that VRPs uncovered many more vulnerabilities and that makes finding and exploiting vulnerabilities more difficult for malicious actors. The results show that the fact that there are more number of vulnerabilities with a high CVSS scores and have no exploits or attacks is because vulnerabilities that are discovered by VRPs result in prioritized fixing.

9.1 Introduction

Vulnerability rewards programs (VRPs) are programs adopted by software vendors to pay security researchers, ethical hackers and enthusiasts for exchange of discovering vulnerabilities in their software and responsibly disclosing the findings to the vendors [52]. Having more eyes on the code means that VRPs uncovered many more vulnerabilities and that makes finding vulnerabilities more difficult for malicious actors and hence ensure the security of software. Besides, vulnerabilities found by VRPs results in a coordinated disclosure and patch that minimizes the risk of vulnerabilities discovery and exploitation [138]. Our approach is inspired by the economics of exploitation model proposed by Miller et al. in [139]:

$$AttackerReturn = (Gain\ per\ use \times Opportunity\ to\ use) -$$
$$(Cost\ to\ acquire\ vulnerability + Cost\ to\ weaponize)$$

The authors argue that an attacker must invest resources to acquire vulnerabilities and develop weaponized exploit for it. While mitigation techniques have shown to increase the cost and complexity of developing an exploit and hence cost of weaponize [97], we argue that VRPs can also

be an important factor in this equation. As software vendors invest significantly to find vulnerabilities the cost of an attacker acquiring a vulnerability is going to be increased and that reduces the likelihood of vulnerabilities discovery and exploitation. Many software vendors such as Google, Mozilla, Facebook, PayPal, and recently Microsoft have adopted using VRPs. They realized that attackers are finding vulnerabilities faster and thus adapting VRPs will help put more sets of eyes looking for vulnerabilities and that makes all vulnerabilities shallow.

There are a number of VRPs and each one of them have their rules and criteria. Among these programs are Mozilla Firefox VRP [140] and Google Chrome VRP [64]. Mozilla Firefox and Google Chrome VRPs determine the reward amount of a vulnerability based on its severity and proof of its exploitation. Both VRPs classify the severity of vulnerabilities as critical, high, medium and low. The details about the description of every severity level for Firefox and Chrome VRPs can be found respectively in [141] and [142] respectively. While Firefox VRP rewards amount ranges from 500 - 10,000, Chrome VRP rewards ranges from 500 - 60000. Firefox VRP pays only for vulnerabilities that has been rated by VRP Commit- tee as a critical or a high and some moderate vulnerabilities, while Chrome VRP rewards critical, high, medium, and some low vulnerabilities.

Recently, there have been efforts to validate CVSS Base score. Some of the researchers have evaluated CVSS Base score using reported exploits [7] and [98] and attacks [143]. The results show that CVSS Base score has a poor correlation with the reported exploits [7, 98] and with the reported attacks [143]. Thus, CVSS Base scores have been considered not a good risk indicator [143] because the majority of vulnerabilities have high scores and have no reported exploits or attacks. Hence, it is hard to use those scores to prioritize among vulnerabilities. However, the lack of exploits or attacks may be a result of prioritized fixing of vulnerabilities that are uncovered by VRPs and that makes finding vulnerabilities more difficult for malicious actors.

In this research, we propose using independent scales used by VRPs to evaluate CVSS Base score. VRPs use their own vulnerability severity rating systems that use a very thorough technical analysis and security experts opinions to assign a severity to vulnerabilities. The severity ratings are then used to pay money ranging from 500$ to 60,000$ or even more. Hence, comparing CVSS

Base score with VRPs severity ratings could explain whether CVSS high scores are reasonable, and why having many high sever vulnerabilities with no reported exploit or attacks.

To conduct this study, we examine 1559 vulnerabilities of Mozilla Firefox and Google Chrome browsers. The two software has been selected because of their rewarding programs maturity and their rich history of publicly documented rewarded vulnerabilities. Besides, the examined vulnerabilities have been assessed by both VRPs rating systems and the CVSS Base score which makes their comparison feasible.

The chapter is organized as follows. Section 2 presents the related work. In Section 3, the selected datasets are presented. In section 4, the validity of CVSS Base score is examined. Section 5 presents the discussion. In section 6, concluding comments are given and the issues that need further research are identified.

9.2 Related Work

Bozorgi et al. [97] have studied the exploitability metrics in CVSS Base metrics. They argued that the exploitability measures in CVSS Base metrics do not differentiate well between the exploited and not exploited vulnerabilities. They attributed that to the fact that many vulnerabilities with a CVSS high score have no reported know exploit and many vulnerabilities with low CVSS scores have a reported know exploit. However, in this paper, we evaluate the performance of CVSS Base score considering both the exploitability and the impact factors using vulnerability rewards programs and we provide an insight into why many vulnerabilities with a high CVSS score and have no exploits.

Allodi and Massacci in [98] have used a case-control study methodology to evaluate whether a high CVSS score or the existence of proof of concept exploit is a good indicator of risk. They use the attacks documented by Symantecs AttackSignature as the ground truth for the evaluation. Their results show that CVSS Base score performs no better than randomly picking vulnerabilities to fix. Besides, they also show that there are many vulnerabilities that have a high CVSS score and are not attacked. However, in this paper, we seek to find an explanation of why the majority of

vulnerabilities have a high CVSS score and have no reported exploits. Thus, we use VRPs instead of an attack in the wild to conduct this study.

Younis and Malaiya in [7] have compared Microsoft rating systems with CVSS Base metrics using the availability of exploits as a ground truth for the evaluation. In addition to finding that both rating systems do not correlate very well with the availability of exploit, they also find that many vulnerabilities have a high CVSS score and have no reported exploits. However, in this study we try to use different ground truth for the evaluation so that an explanation for why many vulnerabilities have a high CVSS scores and have no reported exploits may be provided.

Finifter et al. in [52] have examined the characteristics of Google Chrome and Mozilla Firefox VRPs. The authors find that using VRPs helps improving the likelihood of finding latent vulnerabilities. They also find that monetary rewards encourage security researchers not to sell their result to the underground economy. Besides, they find that patching vulnerabilities found by the VRPs increases the difficulties and thus the cost for the malicious actors to find zero-day vulnerabilities or exploit them. However, in this study, we examine using VRPs as ground truth to evaluate CVSS Base score.

Swamy et al. in [97] at the Microsoft Security Response Center examine the impact of using exploit mitigation techniques that Microsoft has implemented to address software vulnerabilities. One of their result shows that stack corruption vulnerabilities that were historically the most commonly exploited vulnerability class are now rarely exploited. However, in this research, we focus on the impact of using VRPs on the availability of exploits and on the relationship between VRPs measures and CVSS Base score.

9.3 Datasets

In this section, we first provide the source of the data. Then we show how the data were collected and analyzed. In this research, the data about vulnerabilities, exploits, and vulnerabilities rewards program data have been collected from the National Vulnerability Database [16], Exploit Database (EDB) [22], and Mozilla Firefox [144] and Google Chrome bug databases [63] receptively. Table 9.1 shows the number of the examined vulnerabilities and their exploits.

Table 9.1: Firefox and Chrome Vulnerabilities

Software	Vulnerabilities	Exploit Exist
Firfox	547	22
Google Chrome	1012	5

It should be noted that the total number of the Firefox vulnerabilities is 742. Out of this number, a 195 vulnerabilities were not examined because we could not find information about them and that is explained as follows. First, 71 vulnerabilities have no direct mapping between the Common Vulnerabilities and Exposures (CVE) number and the Firefox Bug ID. Second, a 122 vulnerabilities could not be accessed due to the unauthorized access permission (You are not authorized to access this data); Third, two vulnerabilities have no data recorded in the Firefox bug database. We found that the VRP data in the Firefox bug database have started to be recorded starting 2009. Thus, all vulnerabilities and exploits of Firefox during the period 2009 to October 2015 were collected. On the other hand, it should also be noted that the total number of vulnerabilities of Chrome is 1084. A 72 vulnerabilities were not examined because we could not find information about them because of the unauthorized access permission"You are not authorized to access this data". We also found that the VRP data in the Chrome bug database have started to be recorded starting 2010. Therefore, all vulnerabilities and exploits of Chrome during the period 2010 to October 2015 were collected. The data of every examined vulnerability of Firefox and Chrome were collected using the following steps. First, from NVD, the vulnerability is first identified. Next, for every existing link in NVD to vendors' bug database, we collected the vulnerability's severity rating and rewards data assigned by the VRPs. After that, for every vulnerabilitys CVE number found in the vendors bug database, the CVSS scores and severity values were collected. Lastly, for every examined vulnerability we used the CVE number to verify whether it has an exploit reported in the EDB or not.

9.3.1 Firefox Vulnerabilities Analysis.

Table 9.2 shows only three of the Firefox vulnerabilities because showing the whole vulnerabilities is limited by the number of pages allowed. Firefox VRP does not provide data about the amount of the reward paid and rather it uses: 1) + symbol to indicate the bug has been accepted and

Table 9.2: The obtained measures of Firefox and CVSS Base Score

CVE	Mozilla Firefox VRP		CVSS Base Score		Exploit Existence
	Reward	VRP Severity	Severity	Score	
CVE-2011-2371	3000-7500	sec-critical	High	10	EE
CVE-2013-1727	500-2500	sec-moderate	Medium	4	NEE
CVE-2015-0833	3000-5000	sec-high	Medium	6.9	NEE

payment will be made, 2) - symbol to indicate the bug does not meet the criteria and payment will not be paid, and 3) ? symbol to indicate the bug is nominated for review by the bounty committee [141].

The CVSS Base score assigns a score in the range [0.0, 10.0]. This score represents the intrinsic and fundamental characteristic of a vulnerability and thus the score does not change over time. CVSS score from 0.0 to 3.9 corresponds to Low severity, 4.0 to 6.9 to Medium severity and 7.0 to 10.0 to High severity. Mozillas security ratings are see-critical: vulnerabilities allow arbitrary code execution, sec-high: vulnerabilities allow obtain confidential data, sec-moderate: vulnerabilities which can provide an attacker additional information, sec-low: minor security vulnerabilities such as leaks or spoofs of non-sensitive information. We have found that 13 vulnerabilities did not meet the criteria for rewarding and hence have been assigned ”–” symbol. We have also found that 11 of them have a low and a moderate severity (five are low and six are moderate) and two are high and critical.

Table 9.3 shows the number of the rewarded and not rewarded vulnerabilities and their severity values for Firefox dataset. It should be noted that the Not Rewarded vulnerabilities are most likely have been discovered by internal discoverers (41.13%) whereas Rewarded vulnerabilities have been discovered by external discoverers (58.87%) [52]. While the Firefox bug database does not clearly provide information about whether the vulnerabilities have been discovered internally or externally, this was very clear in the Chrome bug database where the name and the team the discoverer works with is provided. 9.3 also shows the severity values and their frequency for rewarded and on rewarded data. It should be noted that the majority of the medium severity vulnerabilities and all low severity vulnerabilities were discovered internally.

Table 9.3: Firefox Dataset

Vulnerabilities	**Rewarded**	**Not Rewarded**
547	225	322
VRP Severity	**Rewarded**	**Not Rewarded**
Critical & High	210	202
Medium	15	89
Low	0	31

Figure 9.1 shows the vulnerabilities severity values of CVSS Base score and Firefox VRP ratings of Firefox dataset. There are 412 vulnerabilities that have been assessed as critical or high severity by VRP rating system whereas there are 312 vulnerabilities that have been assessed as high severity by CVSS Base score. It should be notated that Firefox VRP severity rating is the baseline that we are comparing CVSS scores with. It should also be noted that Shared means the same vulnerabilities, which have the same CVE number, that have been assigned the same severity value by Firefox VRP rating system and CVSS Base score. On the other hand, Not Shared means the same vulnerabilities, but have been assigned different severity values by CVSS Base score.

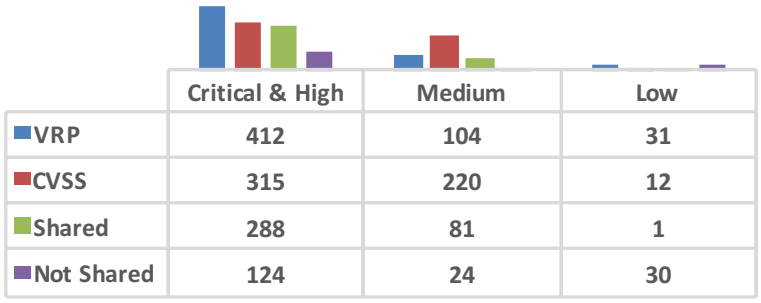


Figure 9.1: Comparing Firefox VRP and CVSS Severity Values

Almost 70% (69.9) of the vulnerabilities that have been assessed by Firefox VRP as critical or high severity have also been assessed as high severity by CVSS. Using Common Weakness Exposure (CWE) [20], which is used to identify vulnerabilities types, we have found that the majority of the Shared vulnerabilities are of the vulnerabilities that execute code. However, the 124 vulnerabilities that are Not Shared have all been assigned a high or critical severity by VRPs rating system, whereas CVSS Base score has assigned to 7 of them a low severity and to 117 of them a moderate severity.

Table 9.4: Vulnerabilites Mismatched by CVSS Base score

VRP	**Crtical**		**High**		**Moderate**	**Low**	
CVSS	Low	Medium	Low	Medium	Low	Medium	High
Total	3	**24**	4	**93**	4	**22**	8

On the other hand, almost 78% (77.88) of the Shared vulnerabilities that have been assessed by Firefox VRP as a medium severity have also been assessed as a medium severity by CVSS. However, there are 24 Not Shared vulnerabilities that have been assigned a medium severity by the VRP rating system. Out of these, 19 vulnerabilities have been assigned a high severity and five have been assigned a low severity. However, only one vulnerability that has been assessed as a low severity by CVSS base score and VRP rating system. While 30 vulnerabilities that have been assessed as a low severity by the VRP rating system, eight have been assessed as a high severity and 22 as a medium severity.

Table 9.4 shows the vulnerabilities that have been mismatched by CVSS Base score. As can be seen, seven vulnerabilities have been assessed as low severity by CVSS whereas three of them have been assessed as critical and four of them has been assessed as high severity by the VRP. It has noticed that those seven vulnerabilities have been assigned critical and high severity values by the Firefox VRP during the debate time, but the vulnerabilities severity first assignments were later changed [24]. We have found that the majority of those vulnerabilities requires unusual users interactions. We have also noticed that CVSS version 3, which has not been used yet, have consider using user interaction factor when assessing exploitability factor [144]. It is clear that the medium range of CVSS scores makes the main part of the mismatch compared to the high and low ranges.

9.3.2 Chrome Vulnerabilities Analysis.

The Chrome vulnerabilities have been examined similar to the Firefox vulnerabilities as shown in Table 9.2. The only difference is that Chrome bug database provides the amount rewarded. Chrome security ratings are similar to that of Mozilla. Unlike Firefox where low severity vulnerabilities are not rewarded, seven low severity vulnerabilities have been rewarded by Chrome VRP.

Table 9.5: Chrome Dataset

Vulnerabilities	**Rewarded**	**Not Rewarded**
1012	584	428
VRP Severity	**Rewarded**	**Not Rewarded**
Critical & High	441	175
Medium	136	137
Low	7	116

The seven low severity vulnerabilities have been found to effect non-critical browser features, crash inside the sandbox, or hang the browser.

Table 9.5 shows the number of the rewarded and not rewarded vulnerabilities and their severity values for Chrome dataset. We have found nine vulnerabilities have been classified as TBD (To Be Determined) and thus considered them as not rewarded. It should be noted that the majority of the Not Rewarded vulnerabilities have been discovered by Google internal discoverers and they represent around 41.4%, whereas the Rewarded vulnerabilities have been discovered by external discoverers and represents around 57.7%. Table 9.5 also shows the severity values and their frequency for the rewarded and not rewarded data. It should be noted that while the majority of the critical and high vulnerabilities have been discovered externally, the majority of the low vulnerabilities have been discovered internally.

The frequency of the amount paid is shown in Figure 9.2. As can be seen, the majority of the rewarded vulnerabilities (404) have been paid either 500$ or 1000$.

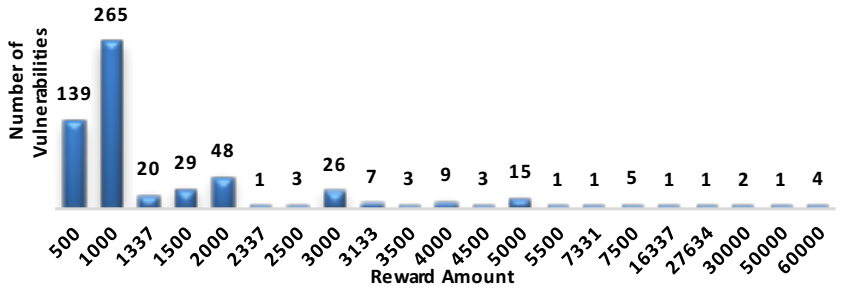
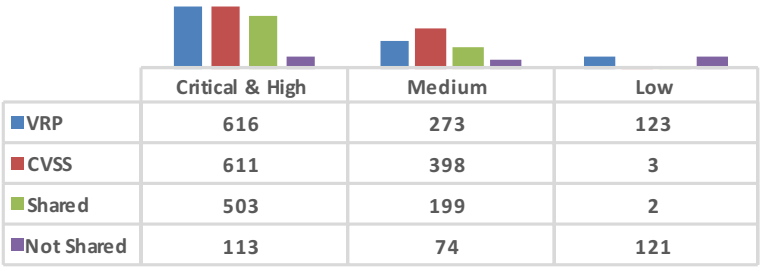


Figure 9.2: Rewarded Amount of Chrome Rewarded Vulnerabilities

We have noticed that 70.79% (286) of those vulnerabilities have been assigned a high severity, 27.47% (111) have been assigned a medium severity, and only 1.73% (7) have been assigned a low severity by VRP. Looking at the point number 3-5 under the Reward amounts section in [7],

we can see that establishing exploitability or providing a Proof of Concept (PoC) or with a poor quality of PoC could be the reason for paying less for many severe vulnerabilities.

Figure 9.3 shows the vulnerabilities severity of CVSS Base score and Chrome VRP rating system of Chrome dataset. Almost 82% (81.65) of the vulnerabilities that have been assessed by Chrome VRP as critical or high severity have also been assessed as high severity by CVSS. However, the 113 vulnerabilities that are Not Shared have all been assigned a high severity by CVSS, whereas VRPs rating system have assigned to 35 of them a low severity and to 78 of them a medium severity.



	Critical & High	Medium	Low
VRP	616	273	123
CVSS	611	398	3
Shared	503	199	2
Not Shared	113	74	121

Figure 9.3: Comparing Chrome VRP and CVSS Severity Values

On the other hand, almost 73% (72.89) of the Shared vulnerabilities that have been assessed by Chrome VRP as a medium severity have also been assessed as a medium severity by CVSS. However, there are 74 Not Shared vulnerabilities that have been assigned a medium severity by CVSS. Out of these, 73 vulnerabilities have been assigned a high severity and only one has been assigned a low severity. However, only two vulnerabilities that have been assessed as a low severity by CVSS Base score and VRP rating system, whereas out of the 121 that have been assigned a low severity by VRP, 35 have been assigned a high severity and 86 have been assigned medium severity by CVSS Base score. Table 9.6 shows the vulnerabilities that have been mismatched by CVSS Base score. We have noticed that out of the 113 vulnerabilities, 75 vulnerabilities have been assigned a high medium score 6.8.

Table 9.6: Vulnerabilites Mismatched by CVSS Base score

VRP	**Crtical**		**High**		**Moderate**	**Low**	
CVSS	Low	Medium	Low	Medium	Low	Medium	High
Total	0	**1**	0	**112**	1	**86**	35

True Positive (TP)	When the CVSS Base score assigns a high severity and VRPs assign critical or high, or, When CVSS Base assigns medium and VRPs assign medium.
True Negative (TN)	When the CVSS Base score assigns low severity and VRPs assign low.
False Negative (FN)	When the CVSS Base score assigns low and VRPs assign critical or high, or, When the CVSS Base score assigns medium and VRPs assign critical or high, or, When the CVSS Base score assigns low and VRPs assign medium.
False Positive (FP)	When the CVSS Base score assigns and VRPs assign medium or low. Or when the CVSS Base score assign medium and VRPs assign low.

9.4 Validation of CVSS Base Score

In this section, we compare CVSS Base score severity with VRPs severity ratings. We assume that VRPs severity rating values are the ground truth because of the through technical analysis and security experts opinions used and the fact that the severity rating are used to pay money. To evaluate the performance of CVSS Base score, we describe when a condition (true or false) is positive or negative as follows:

Since CVSS Base score uses an ordinal range: 0-3.9 = Low, 4 - 6.9 = Medium, and 7-10 = High, the possibility of overlapping between the ranges could make high Low vulnerability such as 3.9 close to medium and high Medium such as 6.9 close to high. To take this into consideration, we used a cluster algorithm to group severity ranges based on the distance between their values. We implemented the K-Means clustering algorithm provided by R language [145] to cluster CVSS Base score for Firefox and Chrome dataset. The result for Firefox vulnerabilities show that Low is in the range from 1.9-5.4, Medium from 5.8-7.6, and High from 8.3-10, whereas the results for

Chrome vulnerabilities show that Low is in the range from 2.6-5.1, Medium from 5.8-7.1, and High from 7.5-10.

We used statistical measures, termed sensitivity, precision, and F-measure to evaluate the performance of CVSS Base score severity. Sensitivity, which also termed recall, is defined as the ratio of the number of vulnerabilities correctly assessed as high or medium to the number of vulnerabilities that are actually high or medium as shown by the following: Sensitivity = TP / TP+FN. Precision, which is also known as the correctness, is defined as the ratio of the number of vulnerabilities correctly assessed as high or medium to the total number of vulnerabilities assessed as high or medium as shown by the following: Precision = TP / TP+FP. For convenient interpretation, we express these two measures in terms of percentage, where a 100% is the best value and 0% is the worst value. Both precision and sensitivity should be as close to the value 100 as possible (no false positives and no false negatives). However, such ideal values are difficult to obtain because sensitivity and precision often change in opposite directions. Therefore, a measure that combines sensitivity and precision in a single measure is needed. F-measure can be interpreted as the weighted average of sensitivity and precision. It measures the effectiveness of a prediction with respect to a user attached β times as much importance to sensitivity as precision. The general formula for the F-measure is shown by the following:

$$F_\beta - Measure = \frac{(1+\beta^2) \times Precision \times Senetivity}{(\beta^2 \times Precision) + Senetivity}$$

β is a parameter that controls a balance between sensitivity and precision. When β = 1, F-measure becomes to be equivalent to the harmonic mean, whereas when β <1 it becomes more precision oriented. However, when β >1, F-measure becomes more sensitivity oriented. In this paper β has been chosen to be 2. Due to their importance, we have also used the FP rate measure: FP rate = FP / FP + TN and the FN rate measure: FN rate = FN / TP + FN.

9.4.1 Result

To calculate the above mentioned performance measures, we need to obtain the confusion matrix for the two datasets. Using the severity ratings assigned by VRPs and CVSS Base score, the confusion matrix was determined as shown in Table 9.7. We have also determined the CVSS Base

Table 9.7: CVSS Base score compared to VRPs Rating Systems

Condition	CVSS Vs. Actual VRPs	Firefox	Chrome
True Positive	When the CVSS High and VRPs Critical or High	288	503
	When CVSS Medium and VRPs Medium	81	199
True Negative	When CVSS Low and VRPs Low	1	2
False Negative	When CVSS Low and VRPs Critical or High	7	0
	When CVSS Says Medium and VRPs Critical or High	117	113
	When CVSS Low and VRPs Medium	4	1
False Positive	When CVSS High and VRPs Low or Medium	27	108
	When CVSS M and VRPs Low	22	86

Table 9.8: Performance Measures for CVSS before and after clustering

Software	Performance Measures	CVSS Scores before clustering (%)	CVSS Scores after clustering (%)
Firefox	Sensitivity	74.25	56
	Precision	88.25	93
	F1-Measure	80.66	70.21
	F2-Measure	57.99	46.94
	False Positive Rate	98	50
	False Negative Rate	25.75	43.54
Chrome	Sensitivity	86	66
	Precision	78	82
	F1-Measure	82	73
	F2-Measure	63	52
	False Positive Rate	99	60
	False Negative Rate	14	34

score ranges obtained by the clustering algorithm and due to the limited pages allowed we only show the results for the CVSS Base score original ranges. It should be noted that we add up the number of every condition, for instance True Positive for Fire fox = 369. As can be seen, CVSS scores before the clustering have a very high FP rate. Using the values in Table 9.7, the performance measures for CVSS Base score original ranges, clustering ranges and our mismatching analysis ranges have been calculated as shown in Table 9.8. We also used Spearman correlation measure to assess the correlation between CVSS scores before clustering and after clustering as shown in Table 9.9. As can be seen, CVSS score correlate with VRPs rating values with p-value less than 0.0001. Clustering score and using mismatching analysis have shown a slight improve-

Table 9.9: Spearman Correlation between CVSS Base score and VRPs Rating System

Software	Correlation	CVSS Scores before clustering	CVSS Scores after clustering
Firefox	Value	0.65	0.47
	P-value	0.0001	0.0001
Chrome	Value	0.53	0.59
	P-value	0.0001	0.0001

ment on the correlation value for the Chrome vulnerabilities whereas no effect have been noticed on the correlation value for Firefox vulnerabilities. However, we also looked at the percentage of the vulnerabilities that have been assigned high and medium severity by CVSS scores and VRPS ratings to verify which measure is more aggressive. For the whole dataset, VRPs have assigned 66% of the vulnerabilities a high severity, whereas CVSS have assigned 59% of the vulnerabilities a high severity. On the other hand, VRPs have assigned 24% of the vulnerabilities a medium severity, whereas CVSS have assigned 46% a medium severity.

9.4.2 Threats to Validity

In this research, we have considered two datasets of two software of the same domain, internet browsers. We consider extending our analysis as long as the data about software from different domains are publicly available and accessible. We are also aware that there are other factors that can affect the vulnerabilities exploitation. Thus, we in no way imply that VRPs should be the only consideration when trying to assess CVSS Base score.

9.5 Discussion

Results have shown that CVSS scores have a higher FP rate. This is mainly because of the number of True Negatives. Out of the 131 vulnerabilities that have been assigned as Low by chrome VRP only two (True Negative) vulnerabilities have been assessed as Low by CVSS. We have found that 86 of these vulnerabilities have been assigned medium and 35 have been assigned high. On the other hand, out of the 31 vulnerabilities that have been assessed as Low by Firefox

VRP, only one (True Negative) vulnerabilities have assessed as Low by CVSS. We have found that 22 of these vulnerabilities have been assigned medium and 8 have been assigned high.

There are more Execute Code vulnerabilities in Firefox than in Chrome. This could be explained by the effect of defensive mechanism, Sandbox, used by Chrome. Furthermore, based on the amount paid, the data from Chrome show that proving exploitability is more valuable than discovering vulnerabilities.

9.6 Conclusion and Future work

This study evaluates CVSS Base Scores as a prioritization metric by comparing it with VRP reward levels, which are arguably more direct measures. We used 1559 vulnerabilities from Mozilla Firefox and Google Chrome browsers to conduct this study. The performance measures and the correlation results show that CVSS Base Score is suitable for prioritization. The fact that there are more vulnerabilities with a high CVSS scores and have no exploits or attacks have been explained by the effect of VRPs on vulnerabilities exploitation. Besides, considering that CVSS score assess most of the vulnerabilities as severer, data show that VRPs have assessed even more vulnerabilities as severe more than CVSS Base score.

Still, there appears to be a need for continued updating of the CVSS metrics and measures. CVSS should highly consider including the Likelihood of Exploit factor (not only the availability of exploit, but also how likely it is that functioning exploit code will be developed) as CWSS [6] and Microsoft [146] rating systems did. Besides. The two chosen VRPs rating systems have shown that Likelihood of Exploit is the main factor that determine the amount of the reward paid for the discoverers and that was very evident in Chrome dataset. As the two datasets considered represent two software of the same domain, examining data from different domains can be valuable as long as their data is publicly available and accessible.

Chapter 10

Conclusions and Future Work

This dissertation has focused on the quantitative assessment of software security risk. Quantitative assessment has been conducted in variety of research fields such as performance assessment, functional evaluation, or statistical modeling. However, it has only begun recently to be applied for software security. In this research, we have examined the following topics i) assessing vulnerability discovery risk, ii) assessing vulnerabilities exploitability risk, and iii) validating CVSS Base metrics.

10.1 Assessing Software Vulnerability Discovery Risk

In this research, we have formally defined and investigated the Folded vulnerability discovery model based on folded normal distribution which is asymmetric by definition and can represent a learningless discovery process. Its model fitting and prediction capabilities have been tested and compared with the AML model for four popular software systems. While both Folded and AML models have been found to fit the vulnerabilities datasets of Windows 7, OSX 5.x, Apache Web server 2.0.x and Internet Explorer 8 well, they differ significantly in the prediction capability. The short learning phase is apparently captured by the Folded model much better than the AML logistic model for the four datasets. The folded model consistently outperforms the AML model in terms of the prediction capabilities for the datasets with no learning phase.

The Folded model needs to be further investigated by applying it to as many software systems as possible and comparing it with other competing models. That will allow development of guidelines as to when this model would be most suitable. In addition, predicting the residual number of vulnerabilities for a new released software without a historical data is a challenge. Thus, using the source code as a source of data to predict the residual number of vulnerabilities could be effective. Some researchers [147] have started to investigate this challenge and further research is required.

10.2 Assessing Individual Vulnerability Discovery Risk

Assessing the risk presented by a software vulnerability is very important for decision makers to prioritize their actions. A vulnerability may remain undiscovered, and hence unremedied, for many successive releases of a software. Here, we proposed a metric to assess a software vulnerability termed Time To Vulnerability Disclosure (TTVD). TTVD is the time taken from when the version containing the vulnerability was first released until the time a vulnerability is discovered and hence disclosed to the public. TTVD can be influenced by extrinsic factors such as discoverers' skills and effort and/or intrinsic attributes of a vulnerability and types. The vulnerabilities rewards program (VRP) data of Google Chrome and the vulnerabilities of Apache HTTP server that does not use VRP showed us the effect of effort and skills on the TTVD. Examining the relationship of TTVD with CVSS Base metrics and vulnerabilities' types showed us that some CVSS metrics relate to TTVD.

Even though the vulnerabilities types have shown a great insight, finding a way to order them in a particular manner is very important for assessing the risk of vulnerability disclosure and discovery. Considering other type of software that use VRPs, such Firefox, and finding a way to include the VRPs as an attribute is very helpful. We have also noticed that there are more types of vulnerabilities in Chrome than in Apache HTTP server. Whether that is because of using VRP requires a further research. Finally, looking for other attributes that can be related to the TTVD is in the top of the list of our priority.

10.3 Assessing The Risk of Vulnerabilities Exploitation

Assessing the severity of a vulnerability requires evaluating the potential risk. Existing measures do not consider software access complexity and tend to rely on subjective judgment. In this paper, we have proposed an approach that uses system related attributes such as attack surface entry points, vulnerability location, call function analysis, and the existence of DSCs. This approach requires us to examine some of the structural aspects of security such as the paths to the vulnerable code starting from the entry points. We have demonstrated the applicability of the pro-

posed approach and have compared resulting measures with overall CVSS severity metrics. Our results show that this approach, involving assessment of the system security based on systematic evaluation and not subjective judgment, is feasible.

Providing a framework that can automate the entire analysis will be helpful in reducing the effort. Thus developing techniques to reduce human involvement and thus enhance scalability in assessing exploitability risk by using techniques such as machine learning is required. We plan to examine the effectiveness of machine learning for automatically assessing the risk of vulnerability exploitation using the proposed properties as features. Given a vulnerable function and their exploitability features, the machine learning model can predict whether it is an exploitable function and estimate the impact of its exploitation.

Quantifying the degree of difficulty of reaching a vulnerability is also valuable for comparing the severity among similar vulnerabilities, and thus needs to be examined. Utilizing the idea of the function call graph depth which has been presented in the discussion section is important. Devising a way of estimating the impact of reachable vulnerabilities will be valuable for estimating the overall risk of individual vulnerabilities and the whole system. Investigating the Node-Rank proposed by Bhattacharya et al. [94] as an estimator of the vulnerability impact is crucial. Finally, identifying whether a vulnerable function is guarded by security control can help better understand the impact of exploitation. Investagating the function exploitation properties proposed by Skape [101] may be helpful in understanding the impact of exploitation when the security controls are present.

10.4 Characterizing Vulnerability Exploitability

In this study, we investigated the possible relationship between the metrics and the existence of a vulnerability exploit. We then characterized these functions using eight software metrics. The metrics have been evaluated for their discriminative and predictive power. The results show that the difference between a vulnerability that has no exploit and a vulnerability that has an exploit can be characterized to some extent using software metrics known for characterizing the presence of vulnerabilities for some of the products. However, the study shows that predicting exploitation of

vulnerabilities is more complicated than predicting the presence of vulnerabilities and thus using metrics that consider security domain knowledge is important for enhancing the performance of a vulnerability exploitation prediction effort.

Improving the classifiers performance and capturing vulnerabilities exploitability may require further empirical investigations of software metrics specifically applicable to the security realm. Thus, further research is needed which considers the metrics related to attack surface [69], reachability and dangerous system calls metrics [12, 9], graph-based metrics [94], and static analysis tool warnings metrics [124]. Moreover, using an alternative approach such as a text mining technique [126] to predict vulnerability exploitability might lead to an interesting results.

10.5 Validating CVSS Base metrics

To validate CVSS Base metrics, this research used two approaches. First, using the availability of exploits we evaluated the performance of the CVSS Exploitability factor and compared its performance to Microsoft (MS) rating system measures. The results showed that exploitability metrics in CVSS and MS do not correlate strongly with the existence of exploits (ground truth), and have a high false positive rate. The high false positive rate result makes me think about exploring different ground truth to explain why too many vulnerabilities have no exploit for them. To address this challenge, we introduced the vulnerability reward programs (VRPs) as a novel ground truth to evaluate the CVSS Base scores. Having more eyes on the code means that VRPs uncovered many more vulnerabilities and that makes finding and exploiting vulnerabilities more difficult for malicious actors. The results show that the fact that there are more number of vulnerabilities with a high CVSS scores and have no exploits or attacks is because vulnerabilities that are discovered by VRPs result in prioritized fixing.

10.5.1 Validating CVSS Base metrics Using Availability of Exploits

This study compares and evaluates the performance of the CVSS Base metrics and Microsoft rating system. The results show that the two measures have a very high false positive rate. It was observed that the sensitivity measure of CVSS exploitability metrics is noticeably affected

by the software type. Besides, CVSS Exploitability factor is highly influenced by the AV values regardless of the other two factors (AC and AU). However, unlike the CVSS Base metrics where the metrics (factors) used for measuring vulnerabilities risk are provided, Microsoft rating system does not provide such metrics but rather provides the values and their definition. Hence it was hard to conduct a thorough investigation trying to correlate the two sets of metrics.

The study suggests that a simple measure of vulnerabilities exploitability using few metrics may not be sufficient. Hence, identifying new metrics that capture attributes that have not been yet considered and adding them to the two rating systems is needed. Younis et al. in [12] have proposed some distinctive metrics based on the software structure. In addition, identifying and including the external factors, such the attacker behavior, to the two selected rating system could improve their precision and reduce their false positive rate.

10.5.2 Validating CVSS Base metrics Using vulnerability rewards program (VRP)

This study evaluates CVSS Base Scores as a prioritization metric by comparing it with VRP reward levels, which are arguably more direct measures. The performance measures and the correlation results show that CVSS Base Score is suitable for prioritization. The fact that there are more vulnerabilities with a high CVSS scores and have no exploits or attacks have been explained by the effect of VRPs on vulnerabilities exploitation. Besides, considering that CVSS score assess most of the vulnerabilities as severer, data show that VRPs have assessed even more vulnerabilities as severe more than CVSS Base score.

Still, there appears to be a need for continued updating of the CVSS metrics and measures. CVSS should highly consider including the Likelihood of Exploit factor (not only the availability of exploit, but also how likely it is that functioning exploit code will be developed) as CWSS [6] and Microsoft [146] rating systems did.

Besides, the two chosen VRPs rating systems have shown that Likelihood of Exploit is the main factor that determine the amount of the reward paid for the discoverers and that was very evident in Chrome dataset. As the two datasets considered represent two software of the same

domain, examining data from different domains can be valuable as long as their data is publicly available and accessible.

References

[1] "Key figures and facts on vulnerabilities from a global information security perspective." https://secunia.com/?action=fetch&filename=secunia_vulnerability_review_2015_pdf.pdf, 2015. [Online; accessed 29-March-2016].

[2] C. P. Pfleeger and S. L. Pfleeger, *Security in computing*. Prentice Hall Professional Technical Reference, 2002.

[3] S. Frei, B. Tellenbach, and B. Plattner, "0-day patch-exposing vendors(in) security performance," *BlackHat Europe*, 2008.

[4] P. Mell, K. Scarfone, and S. Romanosky, "A complete guide to the common vulnerability scoring system version 2.0," in *Published by FIRST-Forum of Incident Response and Security Teams*, pp. 1–23, 2007.

[5] "OWASP Risk Rating Methodology." https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology. [Online; accessed 09-May-2016].

[6] "Common Weakness Scoring System." https://cwe.mitre.org/cwss/cwss_v1.0.1.html. [Online; accessed 30-March-2016].

[7] A. A. Younis and Y. K. Malaiya, "Comparing and evaluating cvss base metrics and microsoft rating system," in *Software Quality, Reliability and Security (QRS), 2015 IEEE International Conference on*, pp. 252–261, IEEE, 2015.

[8] A. A. Younis and Y. K. Malaiya, "Relationship between attack surface and vulnerability density: A case study on apache http server," in *Proceedings on the International Conference on Internet Computing (ICOMP)*, p. 1, The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2012.

[9] A. A. Younis and Y. K. Malaiya, "Using software structure to predict vulnerability exploitation potential," in *Software Security and Reliability-Companion (SERE-C), 2014 IEEE Eighth International Conference on*, pp. 13–18, IEEE, 2014.

[10] A. Younis, Y. K. Malaiya, and I. Ray, "Evaluating cvss base score using vulnerability rewards programs," in *to be presented at 31th Int. Information Security and Privacy Conf., IFIP SEC, Ghent, Belgium, May 2016*.

[11] A. A. Younis, Y. K. Malaiya, and I. Ray, "Using attack surface entry points and reachability analysis to assess the risk of software vulnerability exploitability," in *High-Assurance Systems Engineering (HASE), 2014 IEEE 15th International Symposium on*, pp. 1–8, IEEE, 2014.

[12] A. Younis, Y. K. Malaiya, and I. Ray, "Assessing vulnerability exploitability risk using software properties," *Software Quality Journal*, vol. 24, pp. 1–44, 2015.

[13] A. Younis, H. Joh, and Y. Malaiya, "Modeling learningless vulnerability discovery using a folded distribution," in *Proc. of SAM*, vol. 11, pp. 617–623, 2011.

[14] A. Younis, Y. Malaiya, C. Anderson, and I. Ray, "To fear or not to fear that is the question code characteristics of a vulnerable functionwith an existing exploit," in *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, pp. 97–104, ACM, 2016.

[15] A. Younis, M. Lee, Y. K. Malaiya, and I. Ray, "Measuring and analyzing time to vulnerability disclosure," 2016.

[16] "National Vulnerability Database." `https://nvd.nist.gov/`. [Online; accessed 29-March-2016].

[17] "Open Source Vulnerability Database." `https://blog.osvdb.org/`. [Online; accessed 29-March-2016].

[18] "BugTraq." `http://www.securityfocus.com/`. [Online; accessed 29-March-2016].

[19] "Common Vulnerabilities and Exposures." `https://cve.mitre.org/`. [Online; accessed 29-March-2016].

[20] "Common Weakness Enumeration." `https://cwe.mitre.org/`. [Online; accessed 29-March-2016].

[21] "Common Vulnerability Scoring System." `https://www.first.org/cvss`. [Online; accessed 29-March-2016].

[22] "Exploit Database." `https://www.exploit-db.com/`. [Online; accessed 29-March-2016].

[23] "Open Source Software." `https://opensource.com/resources/what-open-source`. [Online; accessed 29-March-2016].

[24] "Bugzilla." `https://www.bugzilla.org/`. [Online; accessed 29-March-2016].

[25] "Bug Tracking Syaytem." `https://en.wikipedia.org/wiki/Bug_tracking_system`. [Online; accessed 29-March-2016].

[26] E. Rescorla, "Is finding security holes a good idea?," *Security & Privacy, IEEE*, vol. 3, no. 1, pp. 14–19, 2005.

[27] R. Anderson, "Security in open versus closed systemsthe dance of boltzmann, coase and moore," tech. rep., Technical report, Cambridge University, England, 2002.

[28] O. Alhazmi, Y. Malaiya, and I. Ray, "Security vulnerabilities in software systems: A quantitative perspective," in *Data and Applications Security XIX*, pp. 281–294, Springer, 2005.

[29] O. H. Alhazmi and Y. K. Malaiya, "Measuring and enhancing prediction capabilities of vulnerability discovery models for apache and iis http servers," in *Software Reliability Engineering, 2006. ISSRE'06. 17th International Symposium on*, pp. 343–352, IEEE, 2006.

[30] J. Y. Kim, "Vulnerability discovery in multiple version software systems: open source and commercial software systems, masters thesis, computer science department, colorado state university," 2007.

[31] H. Joh, J. Kim, and Y. K. Malaiya, "Vulnerability discovery modeling using weibull distribution," in *Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on*, pp. 299–300, IEEE, 2008.

[32] H. Joh and Y. Malaiya, "Modeling skewness in data with s-shaped vulnerability discovery models," in *Proc. Int. Symp. Software Reliability Eng.(ISSRE)*, pp. 406–407, 2010.

[33] J. A. Ozment, *Vulnerability discovery & software security*. PhD thesis, University of Cambridge, 2007.

[34] P. Anbalagan and M. Vouk, "On reliability analysis of open source software-fedora," in *Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on*, pp. 325–326, IEEE, 2008.

[35] T. Zimmermann, N. Nagappan, and L. Williams, "Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista," in *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on*, pp. 421–428, IEEE, 2010.

[36] O. H. Alhazmi and Y. K. Malaiya, "Modeling the vulnerability discovery process," in *Software Reliability Engineering, 2005. ISSRE 2005. 16th IEEE International Symposium on*, pp. 10–pp, IEEE, 2005.

[37] O. H. Alhazmi and Y. K. Malaiya, "Quantitative vulnerability assessment of systems software," in *Proc. annual reliability and maintainability symposium*, pp. 615–620, 2005.

[38] A. Ozment and S. E. Schechter, "Milk or wine: does software security improve with age?," in *Usenix Security*, 2006.

[39] K. Chan, D. Feng, P. Su, C. Nie, and X. Zhang, "Multicycle vulnerability discovery model for prediction," *Journal of Software*, vol. 21, no. 9, pp. 2367–2375, 2010.

[40] S.-W. Woo, H. Joh, O. H. Alhazmi, and Y. K. Malaiya, "Modeling vulnerability discovery process in apache and iis http servers," *Computers & Security*, vol. 30, no. 1, pp. 50–62, 2011.

[41] C. Daniel, "Use of half-normal plots in interpreting factorial two-level experiments," *Technometrics*, vol. 1, no. 4, pp. 311–341, 1959.

[42] F. Leone, L. Nelson, and R. Nottingham, "The folded normal distribution," *Technometrics*, vol. 3, no. 4, pp. 543–550, 1961.

[43] Y. K. Malaiya, N. Karunanithi, and P. Verma, "Predictability of software-reliability models," *Reliability, IEEE Transactions on*, vol. 41, no. 4, pp. 539–546, 1992.

[44] W. A. Arbaugh, W. L. Fithen, and J. McHugh, "Windows of vulnerability: A case study analysis," *Computer*, vol. 33, no. 12, pp. 52–59, 2000.

[45] S. Frei, M. May, U. Fiedler, and B. Plattner, "Large-scale vulnerability analysis," in *Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense*, pp. 131–138, ACM, 2006.

[46] S. Frei, D. Schatzmann, B. Plattner, and B. Trammell, "Modeling the security ecosystem-the dynamics of (in) security," in *Economics of Information Security and Privacy*, pp. 79–106, Springer, 2010.

[47] M. A. McQueen, T. A. McQueen, W. F. Boyer, and M. R. Chaffin, "Empirical estimates and observations of 0day vulnerabilities," in *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on*, pp. 1–12, IEEE, 2009.

[48] B. Schneier, "Full disclosure and the window of exposure," *Crypto-Gram Newsletter*, 2000.

[49] H. Joh and Y. K. Malaiya, "Defining and assessing quantitative security risk measures using vulnerability lifecycle and cvss metrics," in *The 2011 international conference on security and management (sam)*, pp. 10–16, 2011.

[50] T. Sommestad, H. Holm, and M. Ekstedt, "Effort estimates for vulnerability discovery projects," in *System Science (HICSS), 2012 45th Hawaii International Conference on*, pp. 5564–5573, IEEE, 2012.

[51] H. Holm, M. Ekstedt, and T. Sommestad, "Effort estimates on web application vulnerability discovery," in *System Sciences (HICSS), 2013 46th Hawaii International Conference on*, pp. 5029–5038, IEEE, 2013.

[52] M. Finifter, D. Akhawe, and D. Wagner, "An empirical study of vulnerability rewards programs.," in *USENIX Security*, vol. 13, pp. 273–288, 2013.

[53] "Chrome Releases." `http://googlechromereleases.blogspot.com/`. [Online; accessed 12-May-2016].

[54] "Google Chrome release history." `https://en.wikipedia.org/wiki/Google\_Chrome\_release\_history`. [Online; accessed 12-May-2016].

[55] "Apache HTTP Server Project." `https://httpd.apache.org/`. [Online; accessed 12-May-2016].

[56] "Apache HTTP Server." `https://de.wikipedia.org/wiki/Apache\_HTTP\_Server\#Apache\_1`. [Online; accessed 12-May-2016].

[57] "Organization for Internet Safty, Guidelines for Security Vulnerability Reporting and Response." https://www.symantec.com/security/OIS_Guidelines\%20for\%20responsible\%20disclosure.pdf. [Online; accessed 13-May-2016].

[58] "CERT, Vulnerability Disclosure Policy." http://www.cert.org/vulnerability\-analysis/vul\-disclosure.cfm. [Online; accessed 13-May-2016].

[59] "Rebooting Responsible Disclosure: a focus on protecting end users." https://security.googleblog.com/2010/07/rebooting\-responsible\-disclosure\-focus.html. [Online; accessed 13-May-2016].

[60] M. Zhao, J. Grossklags, and P. Liu, "An empirical study of web vulnerability discovery ecosystems," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1105–1117, ACM, 2015.

[61] M. Hafiz and M. Fang, "Game of detections: how are security vulnerabilities discovered in the wild?," *Empirical Software Engineering*, pp. 1–40, 2015.

[62] "YEARFRAC function." https://support.office.com/en\-gb/article/YEARFRAC\-function\-3844141e\-c76d\-4143\-82b6\-208454ddc6a8. [Online; accessed 15-May-2016].

[63] "Google Chrome Bug Repository." https://code.google.com/p/chromium/issues/list. [Online; accessed 30-March-2016].

[64] "Chrome Reward Program Rules." https://www.google.com/about/appsecurity/chrome-rewards/index.html. [Online; accessed 30-March-2016].

[65] G. Forman, "An extensive empirical study of feature selection metrics for text classification," *The Journal of machine learning research*, vol. 3, pp. 1289–1305, 2003.

[66] L. Glanz, S. Schmidt, S. Wollny, and B. Hermann, "A vulnerability's lifetime: enhancing version information in cve databases," in *Proceedings of the 15th International Conference on Knowledge Technologies and Data-driven Business*, p. 28, ACM, 2015.

[67] W. Jansen, *Directions in security metrics research*. Diane Publishing, 2010.

[68] K. M. Goertzel, T. Winograd, H. L. McKinley, L. J. Oh, M. Colon, T. McGibbon, E. Fedchak, and R. Vienneau, "Software security assurance: a state-of-art report (sar)," tech. rep., DTIC Document, 2007.

[69] P. K. Manadhata and J. M. Wing, "An attack surface metric," *Software Engineering, IEEE Transactions on*, vol. 37, no. 3, pp. 371–386, 2011.

[70] V. Verendel, "Quantified security is a weak hypothesis: a critical survey of results and assumptions," in *Proceedings of the 2009 workshop on New security paradigms workshop*, pp. 37–50, ACM, 2009.

[71] O. H. Alhazmi, Y. K. Malaiya, and I. Ray, "Measuring, analyzing and predicting security vulnerabilities in software systems," *Computers & Security*, vol. 26, no. 3, pp. 219–228, 2007.

[72] P. Manadhata, J. Wing, M. Flynn, and M. McQueen, "Measuring the attack surfaces of two ftp daemons," in *Proceedings of the 2nd ACM workshop on Quality of protection*, pp. 3–10, ACM, 2006.

[73] J. Wing, "Measuring relative attack surfaces," *Computer Security in the 21st Century*, p. 109, 2005.

[74] P. Manadhata and J. M. Wing, "An attack surface metric," tech. rep., DTIC Document, 2005.

[75] R. B. Bloom and B. Foreword By-Behlendorf, *Apache Server 2.0: The Complete Reference*. Osborne/McGraw-Hill, 2002.

[76] "Web Server Survey." `http://www.netcraft.com/`. [Online; accessed 30-March-2016].

[77] "World Wide Web Technology Surveys." `http://w3techs.com/technologies/overview/web_server/all`. [Online; accessed 30-March-2016].

[78] J. Kim, Y. K. Malaiya, and I. Ray, "Vulnerability discovery in multi-version software systems," in *High Assurance Systems Engineering Symposium, 2007. HASE'07. 10th IEEE*, pp. 141–148, IEEE, 2007.

[79] "Apache Archive." `http://archive.apache.org/dist/httpd/`. [Online; accessed 30-March-2016].

[80] "SLOCCount." `http://www.dwheeler.com/sloccount/`. [Online; accessed 30-March-2016].

[81] "cflow ." `http://www.gnu.org/software/cflow/manual/cflow.html`. [Online; accessed 30-March-2016].

[82] L. Ponemon, "Cost of data breach study: Global analysis," *Poneomon Institute sponsored by Symantec*, 2013.

[83] L. Allodi and F. Massacci, "A preliminary analysis of vulnerability scores for attacks in wild: the ekits and sym datasets," in *Proceedings of the 2012 ACM Workshop on Building analysis datasets and gathering experience returns for security*, pp. 17–24, ACM, 2012.

[84] G. Stoneburner, A. Goguen, and A. Feringa, "Risk management guide for information technology systems (special publication 800-30). gaithersburg, md: National institute of standards and technology, 2002."

[85] S. Horwitz, T. Reps, and D. Binkley, "Interprocedural slicing using dependence graphs," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 12, no. 1, pp. 26–60, 1990.

[86] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The program dependence graph and its use in optimization," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 9, no. 3, pp. 319–349, 1987.

[87] D. J. Kuck, Y. Muraoka, and S.-C. Chen, "On the number of operations simultaneously executable in fortran-like programs and their resulting speedup," *Computers, IEEE Transactions on*, vol. 100, no. 12, pp. 1293–1310, 1972.

[88] D. Evans and D. Larochelle, "Improving security using extensible lightweight static analysis," *software, IEEE*, vol. 19, no. 1, pp. 42–51, 2002.

[89] S. Toolworks, "Inc:understand source code analysis & metrics," 2014.

[90] P. B. Galvin, G. Gagne, and A. Silberschatz, *Operating system concepts*. John Wiley & Sons, Inc., 2013.

[91] M. Bernaschi, E. Gabrielli, and L. V. Mancini, "Remus: a security-enhanced operating system," *ACM Transactions on Information and System Security (TISSEC)*, vol. 5, no. 1, pp. 36–61, 2002.

[92] "Red Hat Bugzilla Main Page.." `https://bugzilla.redhat.com/.` [Online; accessed 30-March-2016].

[93] "The Apache Software Foundation." `http://svn.apache.org/viewvc/.` [Online; accessed 30-March-2016].

[94] P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos, "Graph-based analysis and prediction for software evolution," in *Proceedings of the 34th International Conference on Software Engineering*, pp. 419–429, IEEE Press, 2012.

[95] S. Sparks, S. Embleton, R. Cunningham, and C. Zou, "Automated vulnerability analysis: Leveraging control flow for evolutionary input crafting," in *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pp. 477–486, IEEE, 2007.

[96] M. Howard, J. Pincus, and J. M. Wing, *Measuring relative attack surfaces*. Springer, 2005.

[97] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond heuristics: learning to classify vulnerabilities and predict exploits," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 105–114, ACM, 2010.

[98] L. Allodi and F. Massacci, "Comparing vulnerability severity and exploits using case-control studies," *ACM Transactions on Information and System Security (TISSEC)*, vol. 17, no. 1, p. 1, 2014.

[99] T. Avgerinos, S. K. Cha, A. Rebert, E. J. Schwartz, M. Woo, and D. Brumley, "Automatic exploit generation," *Communications of the ACM*, vol. 57, no. 2, pp. 74–84, 2014.

[100] D. Brenneman, "Improving software security by identifying and securing paths linking attack surfaces to attack targets," *white paper avail. at: http://http://www. mccabe. com*, 2012.

[101] Skape, "Improving software security analysis using exploitation properties. uninformed." http://www.uninformed.org/?o=about. [Online; accessed 30-March-2016].

[102] Y. Shin and L. Williams, "Is complexity really the enemy of software security?," in *Proceedings of the 4th ACM workshop on Quality of protection*, pp. 47–50, ACM, 2008.

[103] Y. Shin and L. Williams, "An empirical model to predict security vulnerabilities using code complexity metrics," in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pp. 315–317, ACM, 2008.

[104] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," *Journal of Systems Architecture*, vol. 57, no. 3, pp. 294–313, 2011.

[105] K. Nayak, D. Marino, P. Efstathopoulos, and T. Dumitraş, "Some vulnerabilities are different than others," in *Research in Attacks, Intrusions and Defenses*, pp. 426–446, Springer, 2014.

[106] M. W. Fagerland and L. Sandvik, "Performance of five two-sample location tests for skewed distributions with unequal variances," *Contemporary clinical trials*, vol. 30, no. 5, pp. 490–496, 2009.

[107] N. Fenton and J. Bieman, *Software metrics: a rigorous and practical approach*. CRC Press, 2014.

[108] T. J. McCabe, "A complexity measure," *Software Engineering, IEEE Transactions on*, vol. SE-2, no. 4, pp. 308–320, 1976.

[109] W. A. Harrison and K. I. Magel, "A complexity measure based on nesting level," *ACM Sigplan Notices*, vol. 16, no. 3, pp. 63–74, 1981.

[110] S. Henry and D. Kafura, "Software structure metrics based on information flow," *Software Engineering, IEEE Transactions on*, vol. 5, pp. 510–518, 1981.

[111] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *Proceedings of the 28th international conference on Software engineering*, pp. 452–461, ACM, 2006.

[112] M. A. Hall and L. A. Smith, "Practical feature subset selection for machine learning," in *Proceedings of the 21st Australasian Computer Science Conference (ACSC'98)*, pp. 181–191, Springer, 1998.

[113] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial intelligence*, vol. 97, no. 1, pp. 273–324, 1997.

[114] I. Jolliffe, *Principal component analysis*. Wiley Online Library, 2002.

[115] B. Schneier, *Beyond fear: Thinking sensibly about security in an uncertain world*. Springer Science & Business Media, 2006.

[116] E. Alata, V. Nicomette, M. Dacier, M. Herrb, *et al.*, "Lessons learned from the deployment of a high-interaction honeypot," *arXiv preprint arXiv:0704.0858*, 2007.

[117] P. Morrison, K. Herzig, B. Murphy, and L. Williams, "Challenges with applying vulnerability prediction models," in *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security*, p. 4, ACM, 2015.

[118] N. Schneidewind, "Standard for a software quality metrics methodology," *Soft. Eng. Standards Subcommittee of the IEEE*, 1990.

[119] "Linux Kernel Archive." `https://www.kernel.org/`. [Online; accessed 30-March-2016].

[120] "LocMetrics." `http://www.locmetrics.com/`. [Online; accessed 30-March-2016].

[121] "WEKA Toolkit." `http://www.cs.waikato.ac.nz/ml/weka`. [Online; accessed 30-March-2016].

[122] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[123] "Usage Statistics and Market Share of Web Servers for Websites.." `http://www.w3techs.com/technologies/overview/web_server/all`. [Online; accessed 30-March-2016].

[124] M. Gegick, L. Williams, J. Osborne, and M. Vouk, "Prioritizing software security fortification throughcode-level metrics," in *Proceedings of the 4th ACM workshop on Quality of protection*, pp. 31–38, ACM, 2008.

[125] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on*, pp. 9–9, IEEE, 2007.

[126] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen, "Predicting vulnerable software components via text mining," *Software Engineering, IEEE Transactions on*, vol. 40, no. 10, pp. 993–1006, 2014.

[127] "MSRC, Microsoft security response center security bulletin severity rating system." `https://technet.microsoft.com/en-us/security/ff943560.aspx`. [Online; accessed 30-March-2016].

[128] "X-Force, X-Force frequently asked questions." `http://www-935.ibm.com/services/us/iss/xforce/faqs.html`. [Online; accessed 30-March-2016].

[129] "Symantec Security Response, Threat severity assessment." `http://www.symantec.com/avcenter/threat.severity.html`. [Online; accessed 30-March-2016].

[130] C. Eiram, "Exploitability/Priority Index Rating Systems (Approaches, Value, and Limitations)." `https://www.riskbasedsecurity.com/reports/RBS-ExploitabilityRatings-2013.pdf`. [Online; accessed 30-March-2016].

[131] "Microsoft Security Bulletin." http://www.symantec.com/avcenter/threat.severity.html. [Online; accessed 30-March-2016].

[132] "Kaspersky Lab, Kaspersky Security Network Report: Windows usage & vulnerabilities 2014." https://securelist.com/files/2014/08/Kaspersky_Lab_KSN_report_windows_usage_eng.pdf. [Online; accessed 30-March-2016].

[133] "Secunia, Secunia Vulnerability Review, Key figures and facts from a global IT-Security perspective." http://secunia.com/?action=fetch\&filename=Secunia_Vulnerability_Review_2013.pdf. [Online; accessed 30-March-2016].

[134] "W3Counter." http://www.w3counter.com/. [Online; accessed 30-March-2016].

[135] L. A. Kuznar, A. Astorino-Courtois, and S. Canna, "From the mind to the feet: Assessing the perception-to-intent-to-action dynamic," tech. rep., DTIC Document, 2011.

[136] E. LeMay, M. D. Ford, K. Keefe, W. H. Sanders, and C. Muehrcke, "Model-based security metrics using adversary view security evaluation (advise)," in *Quantitative Evaluation of Systems (QEST), 2011 Eighth International Conference on*, pp. 191–200, IEEE, 2011.

[137] S. Vidalis and A. Jones, "Analyzing threat agents and their attributes.," in *Proceedings of the 4th European Conference on i-Warfare and Security (ECIW)*, pp. 369–380, 2005.

[138] "Dark Reading." http://www.darkreading.com/coordinated-disclosure-bug-bounties-helpspeed-%20patches/d/d-id/1139551. [Online; accessed 30-March-2016].

[139] S. S. Nagaraju, C. Craioveanu, E. Florio, and M. Miller, "Software vulnerability exploitation trends.," tech. rep., Microsoft Trustworthy Computing Security, 2013.

[140] "The Mozilla Security Bug Bounty Program." https://www.mozilla.org/en-US/security/bug-bounty/. [Online; accessed 30-March-2016].

[141] "Security Severity Ratings." https://wiki.mozilla.org. [Online; accessed 30-March-2016].

[142] "Severity Guidelines for Security Issues." https://www.chromium.org/developers/severity-guidelines. [Online; accessed 30-March-2016].

[143] M. Miller, T. Burrell, and M. Howard, "Mitigating software vulnerabilities," tech. rep., Technical report, Cambridge University, England, 2002.

[144] "Security Advisories for Firefox." https://www.mozilla.org/en-US/security/known-vulnerabilities/firefox/. [Online; accessed 30-March-2016].

[145] "R: A Language and Environment for Statistical Computing." https://www.r-project.org/. [Online; accessed 30-March-2016].

[146] "Using Exploitability Index." https://technet.microsoft.com/en-us/security/ff943560.aspx. [Online; accessed 30-March-2016].

[147] S. Rahimi and M. Zargham, "Vulnerability scrying method for software vulnerability discovery prediction without a vulnerability database," *Reliability, IEEE Transactions on*, vol. 62, no. 2, pp. 395–407, 2013.