ABSTRACT

Title of Dissertation:	Some Guidelines for Risk Assessment of Vulnerability Discovery Processes
	Yazdan Movahedi, Doctor of Philosophy, 2019
Dissertation directed by:	Associate Professor Michel Cukier, Department of Mechanical Engineering

Software vulnerabilities can be defined as software faults, which can be exploited as results of security attacks. Security researchers have used data from vulnerability databases to study trends of discovery of new vulnerabilities or propose models for fitting the discovery times and for predicting when new vulnerabilities may be discovered. Estimating the discovery times for new vulnerabilities is useful both for vendors as well as the end-users as it can help with resource allocation strategies over time.

Among the research conducted on vulnerability modeling, only a few studies have tried to provide a guideline about which model should be used in a given situation. In other words, assuming the vulnerability data for a software is given, the research questions are the following: Is there any feature in the vulnerability data that could be used for identifying the most appropriate models for that dataset? What models are more accurate for vulnerability discovery process modeling? Can the total number of

publicly-known exploited vulnerabilities be predicted using all vulnerabilities reported for a given software?

To answer these questions, we propose to characterize the vulnerability discovery process using several common software reliability/vulnerability discovery models, also known as Software Reliability Models (SRMs)/Vulnerability Discovery Models (VDMs). We plan to consider different aspects of vulnerability modeling including curve fitting and prediction.

Some existing SRMs/VDMs lack accuracy in the prediction phase. To remedy the situation, three strategies are considered: (1) Finding a new approach for analyzing vulnerability data using common models. In other words, we examine the effect of data manipulation techniques (i.e. clustering, grouping) on vulnerability data, and investigate whether it leads to more accurate predictions. (2) Developing a new model that has better curve filling and prediction capabilities than current models. (3) Developing a new method to predict the total number of publicly-known exploited vulnerabilities using all vulnerabilities reported for a given software.

The dissertation is intended to contribute to the science of software reliability analysis and presents some guidelines for vulnerability risk assessment that could be integrated as part of security tools, such as Security Information and Event Management (SIEM) systems.

SOME GUIDELINES FOR RISK ASSESSMENT OF VULNERABILITY DISCOVERY PROCESSES

by

Yazdan Movahedi

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2019

Advisory Committee:
Associate Professor Michel Cukier, Chair
Professor Rance Cleaveland, Dean's Representative
Professor Mohammad Modarres
Professor Jeffrey W. Herrmann
Assistant Professor Katrina Groth

Acknowledgements

Over the past four years, I have met many incredible people who have contributed to this significant personal and professional achievement. I would like to thank each and every one who has supported me, encouraged me and helped me throughout this path.

First, I would like to thank my committee members Dr. Mohammad Modarres, Dr. Rance Cleaveland, Dr. Jeffrey Herrmann, and Dr. Katrina Groth for their insightful and valuable feedback.

I am very thankful to my advisor, Dr. Michel Cukier, who gave me the opportunity to study at the University of Maryland. This dissertation would not have been possible without his guidance and support.

I also had the great opportunity to collaborate with Dr. Ilir Gashi and his research team from the City University in London, over the course of my graduate studies. I would like to express my gratitude to Dr. Gashi who never withholds his advice and support in all the time of research.

On a more personal note, moving to a foreign country was not always easy, but I was lucky to build many great friendships along the way: Mehdi, Sanaz, Ali, Daniel, Elaheh, Roohollah, Parastoo, Peyman, Miead, Fatimah, and Amin.

I would like to thank my family. This journey would not have been possible without the support of my family. You have always been there when I needed you. I admire how supportive you are; you taught me to never give up, and it is thanks to you that I have made it to where I am now.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Background and Motivation

Vulnerabilities are software faults, that have the potential to be exploited as results of security attacks [1]. The process of vulnerability risk assessment can be investigated from two different perspectives: (1) based upon the influence vulnerabilities have on software, or (2) based upon the risk which is associated with a single vulnerability [2].

At the software level, the risk of detecting new vulnerabilities can be assessed through determining the number of vulnerabilities that are going to be detected over time. Even though public vulnerability resources such as the common vulnerability exposures (CVE), national vulnerability database (NVD), and open source vulnerability database (OSVDB) exist, estimating the total number of vulnerabilities associated with a software is still difficult.

Software reliability models (SRMs) are well known and have been studied for over 40 years [3]. These models have been widely used for assessing the risk associated with vulnerabilities because vulnerabilities are software faults that can be exploited as the result of security attacks. Research has been conducted to create a link between the fault discovery process and the vulnerability discovery process for modeling purposes [4]. Thus, vulnerability discovery models (VDMs) and Software Reliability Models (SRMs) can be considered similar based on the fault detection processes [1].

Researchers have used data from various vulnerability databases to study trends of discovery of new vulnerabilities and used various models for predicting when new

vulnerabilities may be discovered [1], [3], [5]–[8]. Several studies have proposed new SRMs/VDMs or applied existing models to estimate security indicators such as total number of residual vulnerabilities in the system, time to next vulnerability (TTNV), vulnerability detection rate, etc. [1], [3]–[12].

Another approach for analyzing vulnerabilities is to find the risk associated with each vulnerability to help companies in making decisions with respect to the severity level of vulnerabilities and determining priorities for patching vulnerabilities. It is critical to rank the severity and exploitability of vulnerabilities since companies use such information to allocate their resources.

Overall, estimating the discovery times for new vulnerabilities is useful both for vendors and for the end-users as it can help with their resource allocation strategies over time.

1.2 Research Questions and Approaches

RQ1: Are there any features in the vulnerability data that could be used for identifying the most appropriate models for that dataset? There are few studies that have tried to provide guidance about which model should be used in a given situation. We address this problem by applying common VDMs on the vulnerabilities associated with eight software and compared their prediction accuracy.

RQ2: What models are more accurate for vulnerability discovery process modeling? The lack of prediction accuracy is another issue regarding reliable vulnerability risk assessment. Two strategies can be considered: (1) Finding a new approach for analyzing vulnerability data still using common VDMs (2) Introducing a

new approach with better prediction capabilities. In this research, we examine both strategies:

- We examine the effect of a data manipulation techniques (i.e. clustering, grouping) on vulnerability data, and investigate whether this approach leads to more accurate predictions compared to those without using clustering.
- We introduce a new approach using a neural network model with more accurate predictions than those from common VDMs.

RQ3: Can the total number of publicly-known exploited vulnerabilities be predicted using all vulnerabilities reported for a given software? Exploited vulnerabilities are vulnerabilities, which were exploited as the result of a security attack. They typically form a small portion of all the vulnerabilities reported for a given software (generally from 2-7% per software version). A huge proportion of the vulnerabilities may never exploited over the lifetime of a software. With such a small number of known exploited vulnerabilities compared to the total number of vulnerabilities, it is difficult to mathematically model and predict when a vulnerability with a known exploit will be reported. We study this issue by introducing an approach for predicting the total number of publicly-known exploited vulnerabilities using all publicly-known vulnerabilities reported for a given software.

1.3 Contributions

In this thesis, we present some guidelines to model the vulnerability discovery process of a given vulnerability dataset. We compare the model fitting and prediction capabilities of eight models: one right-skewed distribution model, one flexible-skewed distribution model, three symmetric distribution models, one Power-law model, one

exponential model, and one quadratic model. We use two different data processing approaches. In the first approach, all the vulnerabilities are considered together, while in the second approach they are initially clustered and then modeled. We calculate the accuracy for each model's fitting and prediction capabilities and analyze the average bias of the models (i.e., whether the models were overestimating or underestimating the number of vulnerabilities).

We then use the eight VDMs to compare their prediction capabilities with the ones of a neural network model.

We also study the link between publicly-known disclosure times of exploited vulnerabilities and all publicly-known vulnerabilities reported. Using this link, we mathematically model and predict when a vulnerability with a known exploit will be reported.

We apply the models on eight datasets: four datasets of well-known operating systems (i.e., Windows, Mac, IOS and Linux) and four datasets of well-known web browsers (i.e., Internet Explorer, Safari, Firefox and Chrome).

Our results showed that, given all the uncertainties associated with our datasets:

- ✓ Considering only VDMs, in terms of prediction accuracy, our proposed clustering approach led to more accurate results in 58% of the cases, while the commonly used approach (without clustering) resulted in more accurate results in 42% cases.
- ✓ Our presented a new modeling approach using neural networks provides higher curve fitting and prediction capabilities than current VDMs.

- ✓ Our proposed approach, which predicts the total number of publicly-known exploited vulnerabilities using all publicly-known vulnerabilities reported for a given software has higher prediction capabilities than current VDMs.

1.4 Dissertation Outline

The rest of the dissertation is organized as follows: Chapter 2 describes the related work. Chapter 3 describes the datasets and models that we used in our research. Chapter 4 applies the models without using clustering and compares them in terms of curve-fitting and prediction capabilities. Chapter 5 describes the clustering technique and a descriptive analysis of the vulnerability datasets after being clustered. It also presents the results of using the clustered datasets with the models for fitting and prediction respectively and provides a comparison between the capabilities with clustered and non-clustered data. Chapter 6 discusses the effect of another data manipulation technique (vulnerability grouping) on prediction capabilities of the VDMs. Chapter 7 presents a new modeling approach using neural networks and evaluates its prediction capability versus VDMs. Chapter 8 presents a new approach for modeling and predicting the total number of publicly-known exploited vulnerabilities using all publicly-known vulnerabilities reported. Chapter 9 discusses future work, summarizes the work completed, and concludes.

Chapter 2: Literature Review

2.1 Vulnerability Databases

Several publicly available vulnerability databases and security advisories exist such as the National Vulnerabilities Database (NVD), Common Vulnerabilities, Exposures (CVE), etc. They provide vulnerability features including Common Vulnerabilities and Enumeration (CVE) identifiers, severity, Common Vulnerability Scoring System (CVSS) scores, published date, patch date, discovery date, and vulnerability type. However, there is no ideal source for vulnerabilities since they usually overlap and complement each other. Then, as a solution, it is better to use a combination of datasets [5].

2.1.1 NVD database

The National Vulnerability Database (NVD) is a public database and is commonly used for research on vulnerability discovery modeling. The NVD, by providing official information about all pre-detected computer vulnerabilities, helps to successfully inform and warn the public about existing vulnerabilities. Since its introduction in 1997, information associated with more than 43,000 software vulnerabilities affecting more than 17,000 software applications has been published by NVD. This valuable information is a great help in understanding trends and detecting patterns in software vulnerabilities, so that security decision makers could better monitor the security of computer systems that are pestered by the ubiquitous software security flaws [13].

However, the NVD comes with some shortcomings such as chronological inconsistency (there is not a unique published date for a given vulnerability among the public repositories), incomplete inclusion (it does not include every detected vulnerability), lack of documentation and repetitive records of a single discovery event. These issues may have been derived from the fact that the NVD was not configured with vulnerability modeling needs in mind [5]. Each vulnerability entry in NVD consists of some fields associated with that vulnerability including Common Platform Enumeration (CPE), and Common Vulnerability Scoring System (CVSS). Based on [13], CPE is defined as "an open framework for communicating the characteristics and impacts of IT vulnerabilities, which provides us with information on a piece of software, including version, edition, language".

CVSS is a scoring system designed to provide a standardized mechanism for evaluating the risk associated with vulnerabilities. Communicating the base, temporal and environmental properties of a vulnerability helps organizations rate the risk associated with that vulnerability. Some components of the CVSS vector are as follows [13]:

- Access Complexity: the difficulty level of the attack required to exploit the vulnerability

- Authentication: indicates whether authenticate is required in order to exploit a vulnerability

- Confidentiality, Integrity and Availability: These features are three loss types of attacks. Confidentiality loss indicates the condition when information is leaked to people who are not supposed to know it. Integrity loss indicates the condition when

illegal modification of data is permitted. Availability loss refers to the situation when the compromised system is not capable of performing its predefined task or is crashed.

The final CVSS Score is calculated based upon the mentioned features, with the goal of indicating the severity associated with a vulnerability.

2.1.2 CVE database

The Common Vulnerabilities and Exposures (CVE) is another free to use source of vulnerabilities hosted by MITRE[1]. CVE is designed to allow vulnerability databases to be linked together, and to make the comparison of security tools and services more straightforward for users. The creation of this list dates to September 1999 [5]. According to the CVE's FAQ[2], "CVE is a dictionary that provides definitions for publicly disclosed cybersecurity vulnerabilities and exposures. The goal of CVE is to make it easier to share data across separate vulnerability capabilities (tools, databases, and services) with these definitions". In this list, every vulnerability is assigned an identification code known as Common Vulnerabilities and Enumeration Identifier (CVE ID). Thanks to these identifiers, the process of sharing data among network security databases and tools has become straightforward since they provide a baseline for evaluation of the security tools' coverage [14]. This source of data includes many features about vulnerabilities that can be leveraged in our analysis. These features are usually derived from CVE entries. Each CVE entry in this database includes[3]: a CVE ID (i.e., "CVE-1999-0067", "CVE-2014-10001"), a brief description of the security vulnerability and at least one pertinent reference (i.e., vulnerability reports and

[1] http://cve.mitre.org/
[2] http://cve.mitre.org/about/faqs.html
[3] https://cve.mitre.org/about/index.html

advisories). All these information are assigned to a vulnerability by a CVE Numbering Authority (CNA). CNAs consist of authorized organizations from around the world eligible to assign CVE IDs to vulnerabilities affecting products within their disclosure, for inclusion in first-time public announcements of new vulnerabilities.

2.1.3 SecurityFocus

SecurityFocus as a computer security portal is a home to the well-known Bugtraq mailing list. Based on SecurityFocus's FAQ[4], "BugTraq is a full disclosure moderated mailing list for the detailed discussion and announcement of computer security vulnerabilities: what they are, how to exploit them, and how to fix them". Each vulnerability entry in this database includes: a Bugtraq ID, a CVE ID, a published date, a brief description of the security vulnerability, and at least one public reference.

2.1.4 CXSecurity/WLB2

World Laboratory of Bugtraq is another collection of information on data communications safety. Based on CXSecurity's FAQ[5], every single user can interact with the database and report a vulnerability. However, each safety note in verified by CXSecurity. Each vulnerability entry in this database includes: a Bugtraq ID, a CVE ID, a published date, a brief description of the security vulnerability, and at least one public reference. Each entry in this database includes: a Bugtraq ID, a CVE ID, a published date, a brief description of the security vulnerability, and at least one public reference.

[4] https://www.securityfocus.com/archive/1/description
[5] https:// https://cxsecurity.com/wlb/about/

2.1.3 Exploit database (EDB)

EDB records exploits and vulnerable software [15]. According to the explanations provided in EDB's official website[6], "The Exploit Database is a CVE compliant archive of public exploits and corresponding vulnerable software, developed for use by penetration testers and vulnerability researchers." It reports vulnerabilities for which there is at least a proof-of-concept exploit. The process of collecting proof-of-concept exploit data is more convenient than collecting data on actual attacks. Based on [16], "a proof-of-concept exploit is merely a byproduct of the so-called 'responsible vulnerability disclosure' process, whereby a security researcher that finds a vulnerability discloses it to the vendor alongside a proof-of-concept exploitation code that proves the existence of the vulnerability itself". Most of EDB's data are derived from Metasploit, 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. In this repository, every vulnerability is assigned an identification code known as EDB Identifier (EDB ID), a CVE ID, an exploit date, and a description.

2.2 Vulnerability Risk Assessment and Modeling: Software Level

Even though vulnerability resources such as the common vulnerability exposures (CVE), national vulnerability database (NVD), and open source vulnerability database (OSVDB) are available, estimating the total number of vulnerabilities in a software is still difficult. Software reliability models (SRMs) are

[6] https://www.exploit-db.com/about-exploit-db

well known and has been studied for over 40 years [3]. Several studies have applied SRMs to estimate software security indicators like the total number of residual vulnerabilities, time to next vulnerability (TTNV), and vulnerability density [3], [6]–[8], [17], [18].

The earliest effort at modeling software reliability was a Markov birth-death model introduced by Hudson in 1967 [19]. A good overview of several SRMs that characterize the process of software defect-finding is provided in [3]. Recently, a few vulnerability discovery models (VDMs) have been proposed to estimate the number of total vulnerabilities in a given software/system. Since vulnerabilities are software faults which are exploited as a result of security attacks [1], VDMs and SRMs can be considered to be similar based on the fault detection processes [1]. Research has been conducted to create a link between the fault discovery process and the vulnerability discovery process for modeling purposes [4].

The earliest study on modeling the vulnerability discovery process was conducted in 2002, when Anderson [20] proposed the first VDM termed the Anderson Thermodynamic (AT) model. Since 2002, a few other VDMs have been proposed. Rescorla [6], [7] proposed a VDM to estimate the number of undiscovered vulnerabilities. In 2005, Alhazmi et al. [21] proposed the application of SRMs to vulnerability discover modeling. The same year, they also introduced a logistic VDM known as Alhazmi–Malaiya Logistic (AML) model, which assumes a symmetrical shape around the peak discovery rate value [8].

In another study [21], Alhazmi and Malaiya found that the AML model provides better goodness-of-fit results compared to Rescorla and Anderson models.

Moreover, an effort-based model was also introduced that uses system installations instead of calendar time as the independent factor. In other words, the authors argued that discovering a vulnerability associated with a software installed on a larger group of computers is more rewarding. However, the effort-based model requires knowing the number of users for a target product in market share, which is not always easy to obtain.

A Weibull distribution-based VDM was proposed by Kim in 2007 [22]. The author argued that the assumption made by the AML model that the vulnerability discovery rate is symmetric around the peak is not always consistent. He leveraged a Weibull distribution to model the asymmetric trend of the discovery rate as an alternative to the AML model. However, the Weibull model did not always provide a good fit. Li *et al*. [23] empirically showed that, in comparison to other reliability models, a Weibull model is better for estimating defect occurrence across a wide range of software systems.

Several studies applied existing models to different types of software packages, such as operating systems and web servers, to simulate the vulnerability discovery rate and predict the number of vulnerabilities that may potentially be present but not yet found [17], [18], [24]. Other studies tried to increase the accuracy of the vulnerability discovery modeling by examining the skewness of the vulnerability data [25].

Research on reliability and risk assessment with a continual vulnerability discovery process has recently started. Studies provided by Anderson [20], Rescorla [6], [7], Alhazmi and Malaiya [8], [18], [21], Kim [26], Ozment and Schechter [27],

Ozment [5], Chan et al. [28], Joh and Malaiya [25] proposed a number of VDMs capable of making different projections on vulnerabilities' disclosure trends.

2.3 Vulnerability Risk Assessment and Modeling: Vulnerability Level

Another approach for analyzing vulnerabilities is to find the risk associated with each vulnerability. Such an approach helps companies to make decisions with respect to the severity level of vulnerabilities. Ranking vulnerabilities is a hard task since it is often difficult to predict how attackers could exploit a vulnerability and use the exploit. Therefore, the risk that different vulnerabilities face is often unknown until they are exploited. In addition, comparing the severity of vulnerabilities is difficult when leveraging their descriptions. It is possible that a simple programing bug can lead to more harm than a major system flaw. Several studies analyze and model vulnerabilities based on their technical features such as exploitability and come up with better estimations. We can divide them into studies that focused on examining sources code of software, vulnerability life-cycle, CVSS metrics, and system calls.

2.3.1 Based on source code

In addition to vulnerabilities' publication dates, some studies used software source code for vulnerability assessment in the context of VDMs. Kim et al. [22] proposed a VDM based on shared source code measurements among multi-version software systems using the source code and vulnerability data of two major versions of Apache HTTP Web server and two major versions of Mysql DBMS. In 2006, Ozment and Schechter applied a reliability growth model to evaluate the security of the OpenBSD OS by examining its source code and the rate at which new code has been

introduced [27]. However, it has been shown that source code cannot be an adequately efficient measure in terms of prediction [5].

Younis [2] assessed vulnerability exploitability for individual vulnerabilities based on source code properties regardless of the availability or unavailability of a patch. In addition to the vulnerabilities publication dates, he took advantage of software source code for vulnerability analysis in the context of VDMs.

Nagappan and Ball [15] performed a pre-release defect prediction using relative code churn metrics on Windows Server 2003. Their multiple linear regression model using principal components analysis provided a high correlation between the estimated failures and the actual failures in software modules (r=0.889 for the Pearson correlation and r=0.929 for the Spearman rank correlation). The relationships between code complexity and vulnerabilities of the Mozilla JavaScript engine at the function level was studied by Shin and Williams [29]. The correlations between code complexity and vulnerabilities were weak (Spearman r=0.3 at best) but statistically significant [29]. Shin et al. [30] through an empirical study showed that by using complexity and code churn metrics, VDMs are capable of predicting vulnerable code locations with high number of calls during a security breach. However, it may generate many false positives. In 2013, Shin [31] showed that the performance of fault and vulnerability prediction is largely affected by the number of the reported faults and vulnerabilities in previous releases.

Recently, Nguyen et al. proposed an automated method that determines the code evidence for the presence of vulnerabilities in previous software versions to evaluate whether the target version is vulnerable or not [32].

2.3.2 Based on vulnerability lifecycle

Disclosure time, exploitation time, and patching time create the lifecycle of a vulnerability. In 2006, a vulnerability lifecycle model was presented by Arbaugh et al. [10] to measure the number of intrusions during the vulnerability lifecycle. Frei et al. [11], [12], linked the patching process to the lifecycle of a vulnerability. They extended Arbaugh et al.'s model using more than 80,000 vulnerabilities; they identified and measured three types of risk exposures as black, gray, and white. They also showed that exploits are often faster to occur than patches. This work has been extended by Shahzad et al. [33], who conducted a descriptive statistical analysis of a large software vulnerability dataset employing clustering on NVD and OSVDB datasets that included vendors and software.

A risk measure was defined by Joh and Malaiya [34] as a probability of adverse events and their impacts. Using Markovian stochastic models, they utilized the vulnerability lifecycle to measure the likelihood of vulnerability exploitability for an individual vulnerability and for the whole system. However, the transition rate between vulnerability lifecycle events has not been determined and the probability distribution of lifecycle events remains to be studied.

Zero-Day vulnerability and its lifespan have been defined by McQueen et al. [35]. Based upon their definition, the zero-day lifespan refers to the time between the vulnerability discovery date and the public disclosure date. They were able to identify the actual vulnerability discovery dates for 15 vulnerabilities. They also compared the CVSS base score to mean lifespan. Younis [2] considered time to vulnerability disclosure (TTVD) or lifespan starting from the vulnerability birth date and correlated

the TTVD with the CVSS base score. Bozorgi et al. [36] refused to do prediction of zero-day vulnerabilities since he believed that their reports occur with the vulnerability already exploited.

2.3.3. Based on CVSS metrics

Common Vulnerability Scoring System (CVSS) metrics are used to measure the severity of vulnerabilities [2]. Exploitability (the ease of exploiting a vulnerability) and impact (the effect of the exploitation) are indicators of the severity. Joh and Malaiya in [34] leveraged the impact related metrics from CVSS to determine the exploitability impact. They applied their metric to assess the risk of two systems that had known unpatched vulnerabilities using actual data.

Descriptive approaches and trends in scheduling of vulnerability patching and exploitation exist. However, most of them use exploit data from OSVDB that does not provide sufficient information about the actual exploitation of a vulnerability and usually the dates reported for exploits by this source are not accurate enough [37]. NVD timing data has also been reported to generate an unforeseeable amount of noise because of how the vulnerability disclosure process works [16], [37].

2.3.4. Based on system calls

System calls are entry points to privileged kernel operations [2]. A system call from a user function can violate the least privilege principle. The principle of least privilege indicates a security protocol, where each part of a system has only the privileges that are needed for its function. In this condition, even if attackers gain access to one part, they would have only limited access to the whole system [38]. An analysis

of UNIX system calls was presented by Bernaschi et al. [39]. They classified system calls according to their level of threat with respect to system penetration. To control system calls, they proposed the Reference Monitor for UNIX System (REMUS), a mechanism to detect an intrusion that may use these system calls. Younis [2] applied their idea utilizing system related attributes such as attack surface entry points, call function analysis, and the existence of dangerous system calls to measure the exploitability of a known vulnerability.

2.4 Methods of Analysis & Risk Assessment Strategies

Several other strategies exist to provide a better understanding of software risk. One is splitting vulnerabilities based on their specifications and studying their behavior in specific subsets like zero-day vulnerabilities. In addition, some studies focused on finding an optimized plan for patch releases with respect to the vulnerability discovery process to diminish the side effects of malicious attacks.

2.4.1 Cluster-based analysis

Clustering is a form of classification method that is very useful in understanding the complex nature of multivariable relationships. In others words, clustering is a method of searching data to detect similarities and dissimilarities in order to find a structure of natural groupings [40].

Clustering can be done for different purposes including splitting real-world exploited vulnerabilities from those which were exploited during software testing [41], and detecting exploited vulnerabilities versus non-exploited ones when there is not enough information about some vulnerabilities [42]. While clustering is categorized as

an explanatory method to simplify the interpretation of data, in most practical applications, to distinguish “good” groupings from “bad” groupings, the researcher should know enough about the context.

Generally, clustering algorithms are divided into hierarchical methods (connectivity-based clustering, used when the number of items is less than 100), non-hierarchical methods (centroid-based clustering, used for more the 100 items), distribution-based clustering (used when Clusters can be defined as items belonging most likely to the same distribution), and density-based clustering (used when clusters are defined as areas of higher density than the remainder of the data set) [40].

Most efforts require a measure of “closeness”, or “similarity” to provide a group structure from a complex data set. When data are clustered, the similarity should be indicated by a measure of distance. The most common distance measures are the Euclidian distance, the geometric distance in multidimensional space, and the Mahalanobis distance which is based on the covariance matrix of the variables [40].

Lee et al [43] investigated a distributed denial of service (DDoS) attack detection method using cluster analysis. Looking into the steps needed to develop a DDoS attack and extracting several traffic variables which best illustrate each phase of the DDoS attack, the authors performed cluster analysis to find precursors for proactive detection of the attack. Shahzad et al [33] conducted a descriptive statistical analysis of a large software vulnerability dataset employing clustering on type-based vulnerability data. Huang et al [44] classified NVD vulnerabilities employing several clustering algorithms to create a relatively objective classification criterion among the vulnerabilities.

2.4.2. Machine learning

Machine learning focuses on automatic recognition of complex patterns and making intelligent predictions or decisions based on data. The technique for learning a classifier (or a function) from training examples when each example is associated with a true label is called the supervised learning method, and it is composed of two main phases [45]. The first phase is the learning or training phase. In this phase, a machine learning algorithm is run on a fraction of training data that consists of pairs of input data (a vector of integers) and their associated output to learn a classifier. Testing is the second step where the pre-learned classifier is tested on the rest of the data to estimate the testing precision of the classifier.

Bozorgi et al. [36] measured vulnerability severity based on analyzing vulnerability exploitability. They discussed the weakness of exploitability measures like CVSS base score metric in providing sufficient information about the vulnerability severity. CVSS metrics are the de facto standard that is used to measure the severity of vulnerabilities [46]. However, CVSS exploitability measures have come under some criticism. They [36] believed that CVSS metrics are only built upon expert knowledge and static formula.

To remedy the situation, the authors proposed a machine learning model using supporting vector machines (SVMs) and a data mining technique that can predict the possibility of a vulnerability getting exploited. In their study, using the CVSS exploitability metric identified many vulnerabilities with a high severity score even though there were no known exploits for those vulnerabilities. This indicates that the

CVSS score does not differentiate between exploited and non-exploited vulnerabilities. This result was also confirmed by [16], [47].

Younis et al. [42] leveraged software properties such as the attack surface entry points, the source code structure, and the vulnerabilities location to determine the vulnerabilities' exploitability. Younis's approach is particularly important for newly released applications that do not have a large amount of historical vulnerabilities. Logistic Regression (LR), Naive Bayes (NB), Random Forests (RF), and Support Vector Machine (SVM) are the machine learning techniques employed in the study. The SVM, when the principal component analysis (PCA) was used, has performed best compared to the other classifiers.

Sabottke et al. [41] explored early detection of exploits using information available on Twitter. They proposed the design of a Twitter-based exploit detector, using supervised machine learning techniques. They leveraged the exploit-related discourse tweets on Twitter (the tweets that included 'CVE') and extracted information posted on public vulnerability resources to evaluate the chances for early detection of the vulnerabilities at risk of being exploited in the presence of benign and adversarial noise. In other words, they investigated techniques for minimizing false-positive detections—vulnerabilities that are not actually exploited—which is critical for prioritizing response actions.

2.4.3. Optimal patch planning

A security patch is a small program that fixes vulnerabilities. Patches usually get distributed to end-users to remove those vulnerabilities. Ideally, the best time to release a patch is the time when a vulnerability appears. However, the development and

distribution of patches involves considerable expenses for vendors. Additionally, a poorly designed patch may lead to introducing new issues. Thus, many of the vendors tend to release patches for their products in a pre-designed timeline [1].

Zheng et al. [48] presented a method for quantifying a security attribute called mean time to security failure (MTTSF) of a virtual machine-based (VM-based) intrusion tolerant system [49] based on queueing theory. They also presented a generalized scheme for tolerating intrusions in a VM-based intrusion tolerant system. In 2016, Luo et al. [50] discussed the patch release strategy from the perspective of vendors by cost criteria. The model assumed a non-homogeneous Poisson process (NHPP) as the number of vulnerabilities discovered, and formulated the expected total cost by considering the damage of exploiting vulnerabilities before and after a patch release. Some researchers have focused on proposing methods according to the cost of security breaches [51]; finding a cost function for vulnerabilities has remained a controversial topic.

2.5. Guidelines for Vulnerability Discovery Models

Security decision makers often use public data sources to help make better decisions regarding, for example, what security products to choose, check for security trends, and estimate when new vulnerabilities that affect their installations will be publicly reported. Several studies have applied software reliability models (SRMs) and vulnerability discovery models (VDMs) to estimate times between public reports of vulnerabilities [1], [3], [5]–[8].

Few studies have tried to provide a guideline about which model should be used in a given situation. Joh et al. [25] investigated the relationship between the

performance of five S-shaped VDMs (i.e., AML, Weibull, Gamma, Normal, and Beta) and the skewness in vulnerability datasets for eight software. Their results showed that Gamma-based VDM, which is a right-skewed VDM, always yields better results with positively skewed (right-skewed) datasets than other models in terms of goodness of fit results and prediction capabilities. For the other VDMs used, no significant correlation was observed. In addition, the authors showed that the AML model performs better than some right-skewed VDMs in terms of prediction when the vulnerability discovery datasets are asymmetrical.

Massacci et al. [24] proposed an empirical methodology that evaluates the performance of VDMs in terms of goodness of fit and predictability. They evaluated most existing VDMs (AT, Rescorla's models, AML, Weibull, Linear) on 30 major releases of four web browsers (i.e., IE, Firefox, Chrome, Safari). They also classified the age of a browser's version in three different periods: youth (within 6-12 months since release date), middle age (12-36 months since release date), and old age (beyond 30 months). Based upon their findings, for a young software, the linear model yielded the best results for estimating the vulnerabilities in the next 3-6 months. For middle-aged browsers the AML model was selected as the best model.

Regarding modeling exploited vulnerabilities, one aspect consists of the probabilistic examination of intrusions by [52], [53]. The lack of data is a significant barrier to modeling exploited vulnerabilities using current VDMs or the machine learning techniques, which require considerable amount of data for satisfactory training.

Chapter 3: Datasets and Models

3.1 Introduction

In this chapter, we will introduce the datasets used in this thesis. Then, we will present two groups of VDMs used for our analysis based upon the classification provided in [24]: S-shaped vulnerability discovery models (VDMs), and non S-shaped VDMs.

3.2 Vulnerability Dataset Creation

The data used in this research has been collected from six different vulnerability data sources including the National Vulnerability Database (NVD)[7] maintained by NIST, the Common Vulnerabilities and Exposures (CVE) database[8], the CVE Details data source[9], the Security database[10], the SecurityFocus data source[11], and the CXSecurity database[12]. All the datasets we used here, derived from those three databases, we previously introduced in Chapter 2.

We used the data generated by a security tool called "VepRisk"[13], which has a backend modules that mine, extract, and store data from public repositories of vulnerabilities. We then stored the data in our own database using MySQL, and identified each vulnerability by its Common Vulnerability Enumeration (CVE) identifier. We used the CVE identifier to compare the reporting date of each

[7] https://nvd.nist.gov
[8] https://cve.mitre.org
[9] https://cvedetails.com/
[10] https://www.security-database.com/
[11] http://www.securityfocus.com/
[12] https://cxsecurity.com/
[13] http://veprisk.city.ac.uk/main

vulnerability in NVD, with the dates in other public repositories on vulnerabilities. We then updated the reporting date on our database to the earliest date that a given vulnerability was known in any of these databases.

We used the NVD as the backbone of comparisons because it includes all the vulnerabilities that can be found in some of the other data sources. Even though some information might be missing in the NVD, it includes the fields that allows to search for the missing information in the other data sources. One example is the Common Platform Enumeration (CPE) identifier, which is only present in the NVD but can be used in combination with the CVE identifier to extract information from different sources. Another example is the vulnerability type that is only present in the CVE Details table.

Vulnerability databases might have high uncertainty regarding some variables associated with the reported vulnerabilities such as published dates. Even though we tried to overcome this issue by collecting vulnerability data from several sources and filtering the published dates based upon the earliest date a vulnerability reported, we should be aware of the uncertain nature of the reported published dates. Regarding that, all the conclusions we draw from this research and their validity are limited by our database uncertainties.

3.3 Vulnerability Dataset Overview

It is important to have a high number of vulnerabilities for each of our analyses. Thus, we decided to focus on two groups: operating systems (OSs) and web browsers. More specifically, we selected four OSs (Windows, Mac, IOS, and Linux) and four

web browsers (Internet Explorer, Safari, Firefox, and Chrome). The results presented in this thesis include the vulnerabilities until the end of 2018.

3.3.1 Operating systems

We focused on vulnerabilities reported for four well-known OSs: Windows (1995-2018), Mac (1997-2018), IOS (the OS associated with Cisco) (1992-2018), and Linux (1994-2018). We chose these OSs as they are most widely used, and had the highest number of vulnerabilities. The start dates indicate the first vulnerability occurrence for the specific OS. For each OS, we included all the vulnerabilities reported for any of its versions. For instance, all the vulnerabilities reported for mac_os, mac_os_server, mac_os_x, and mac_os_x_server were put together to create a vulnerability database for Mac. We did this to have a high number of vulnerabilities for each OS. The total number of distinct vulnerabilities (unique CVE-IDs) for these OSs is 12,852. Table I presents the number of vulnerabilities for the four OSs.

3.3.2 Web browsers

We focused on the vulnerabilities reported for four well-known web browsers: Internet Explorer (1997-2018), Safari (2003-2018), Firefox (2003-2018), and Chrome (2008-2018). These browsers were selected since they are widely used and had the highest number of vulnerabilities. The start dates indicate the first vulnerability occurrence for the specific web browser. Similar to what we did for OSs, we considered, for each browser, all the vulnerabilities reported for any of its versions. As an example, all the vulnerabilities reported for ie, ieexplorer, and ie_for_macintosh were combined under Internet Explorer. This allowed us to have a high number of

vulnerabilities for each browser. The total number of vulnerabilities for these browsers is 6,546. Table 1 presents the number of vulnerabilities for the four browsers.

Table 1: NUMBER OF VULNERABILITIES PER SOFTWARE

OS	**Windows**	**Mac**	**IOS**	**Linux**
# Vulnerabilities	3434	2908	698	5812
Web Browsers	**IE**	**Safari**	**Firefox**	**Chrome**
# Vulnerabilities	1862	994	1784	1906

3.4 Datasets Characterization

In this section, we characterize the datasets using well-known statistical indicators. Distributions are characterized by their first four moments and indicators like skewness and kurtosis highlight distribution properties. In this section, we will characterize the datasets based on their skewness since this indicator is widely used by the vulnerability modeling community [22], [25]. The skewness of a dataset/distribution specifies its degree of asymmetry around its expected value [25]. The skewness values are calculated via following equation [40]:

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

where n is the number of data points, x_i is the i^{th} data value, $\bar{x}$ represents the mean value and s is the standard deviation. For a given distribution with an absolute skewness value of greater than 1, between 0.5 and 1, and less than 0.5, the distribution is highly skewed, moderately skewed, and approximately symmetric, respectively [40]. The datasets with an absolute value greater than 0.5 and positive skewness values are called right-skewed datasets and referred to those datasets where more vulnerabilities are

reported later in the product lifecycle, and vice versa. Figure 1 shows an overall shape of the vulnerability discovery process with different skewness values.

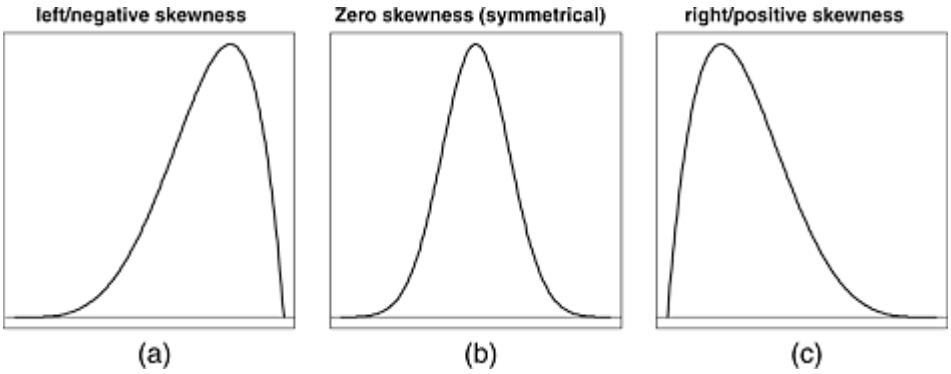

Figure 1. Overall view of a distribution with (a) negative skewness, (b) appoximately zero skewness, and (c) positive skewness[14]

Table 2 presents the skewness value of each software used in this research. All the datasets are right-skewed (each has a positive skewness value with an absolute value of skewness greater than 0.5). In other words, for a software with a right-skewed dataset, plots of the number of discovered vulnerabilities associated with the software grouped by time blocks lead to a significant number of vulnerabilities on the left side of the plot.

Table 2: SKEWNESS VALUES PER SOFTWARE

OS	**Windows**	**Mac**	**IOS**	**Linux**
# Skewness	2.381	2.459	3.687	3.589
Web Browsers	**IE**	**Safari**	**Firefox**	**Chrome**
# Skewness	2.705	3.381	11.00	1.973

3.5 S-shaped Vulnerability Discovery Models

S-shaped VDMs, which measure the total number of detected vulnerabilities, divide the process of vulnerability discovery into three phases as shown in Figure 2.

[14] H. Joh and Y. K. Malaiya, "Modeling Skewness in Vulnerability Discovery: Modeling Skewness in Vulnerability Discovery," Qual. Reliab. Eng. Int., vol. 30, no. 8, pp. 1445–1459, Dec. 2014.

Phase 1 represents the learning phase, which starts from the introduction of the software and continues until the beginning of the linear phase as a consequence of increasing popularity of the software [25]. During the learning phase, the vulnerability discovery intensity function is an increasing function. Phase 2 or the linear phase is the period when most of the vulnerabilities are detected. The intensity function associated with the vulnerability discovery process of this phase is constant. Phase 3 or the saturation phase is the period when most of the vulnerabilities have been discovered and only a few vulnerabilities remain undiscovered [24]. The vulnerability discovery intensity function for the saturation phase is decreasing. Note that the saturation phase might not be observable for all software. This phase will not appear as long as a significant number of vulnerabilities are still undetected. The S-shaped VDMs used in this research are not distributions. They are built based upon well-known distributions and their purpose is to count the total number of vulnerabilities [24]. Regarding this, for all the models we used in this study, we only applied their cumulative forms $\Omega(t)$ on vulnerability data. For each software, the variable we are predicting is the cumulative number of vulnerabilities reported in 30 days' time intervals. In other words, t is associated with 30 day intervals and Ω represents the cumulative number of vulnerabilities reported within each interval.

We use the model skewness to select the S-shaped VDMs in this research. We selected five S-shaped VDMs: one right-skewed model (Gamma-based VDM), one flexible-skewed model (Weibull-based VDM), and two symmetrical models (Alhazmi–Malaiya Logistic (AML) model and Normal distribution-based model). These VDMs were selected because they are the most well-known right-skewed,

flexible-skewed, and symmetrical distribution-based VDMs in modeling the vulnerability discovery process. We detail in the following sections these five models.

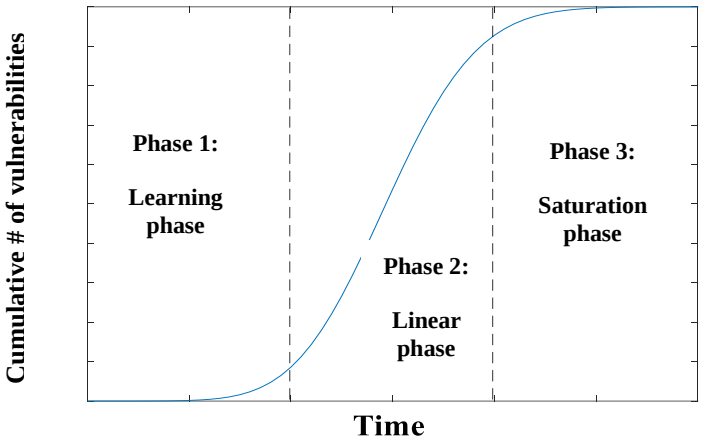

Figure 2. Three phases for S-shaped models

3.5.1. Gamma-based VDM

The Gamma-based VDM, derived from the Gamma distribution, belongs to the family of right-skewed distributions. It has a continuous intensity function with three parameters: α (shape parameter), β (scale parameter), and γ, which represents the total number of vulnerabilities that would finally be discovered. The vulnerability discovery rate/intensity function, ω, for the Gamma-based VDM as well as its cumulative model (Ω) presented in (2) and (3), respectively.

$$\omega(t) = \frac{\gamma}{\Gamma(\alpha)\beta^{\alpha}} t^{\alpha-1} e^{-\frac{t}{\beta}} \tag{2}$$

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

$$\Omega(t_0) = \int_{t=0}^{t_0} \frac{\gamma}{\Gamma(\alpha)\beta^{\alpha}} t^{\alpha-1} e^{-\frac{t}{\beta}}\, dt \tag{3}$$

This distribution is only defined for t >0. The shape and the scale parameters are always positive. It is expected that for the software with large values of t, right-skewed distributions provide better fits to vulnerability discovery data than other models [25] because of gradual reduction in the number of discovered vulnerabilities, which yields a tail on the right side of the relevant vulnerability discovery intensity function.

3.5.2 Weibull-based VDM

The Weibull-based VDM, derived from the Weibull distribution, belongs to the family of flexible-skewed distributions. This VDM was first introduced in 2007 [22]. The vulnerability discovery rate/intensity function, ω, for the Weibull-based VDM as well as its cumulative model (Ω) presented in (4) and (5), respectively.

$$\omega(t) = \frac{\alpha\gamma}{\beta}\left(\frac{t}{\beta}\right)^{\alpha-1} e^{-\left(\frac{t}{\beta}\right)^{\alpha}} \tag{4}$$

$$\Omega(t) = \gamma\left\{1 - e^{-\left(\frac{t}{\beta}\right)^{\alpha}}\right\} \tag{5}$$

Like the Gamma-based VDM, the Weibull-based VDM has a continuous intensity function with three parameters: α (shape parameter), β (scale parameter), and γ which represents the total number of vulnerabilities that would finally be discovered. This VDM can be symmetrical with zero skewness for α values around 3. For α <3, this VDM is always right-skewed, while for α >3, it is left-skewed. Like the Gamma-based VDM, this distribution is defined for t >0.

3.5.3 AML VDM

Alhazmi–Malaiya Logistic (AML) model belongs to the family of distributions with symmetrical intensity (rate) functions. This model was first introduced in 2005 [8] and is based upon the idea that as an operating system gains market share, the attention it receives increases. Then, after experiencing a peak, it starts decreasing when a newer version is released. Overall, the AML model assumes the cumulative number of vulnerabilities is influenced by two factors: the share of the installed base (increasing factor) and the number of remaining undiscovered vulnerabilities (declining factor). The AML model has three parameters including a constant C. Parameters A and B are empirical constants and directly estimated from the dataset. B stands for the total number of vulnerabilities that would finally be discovered. This model is defined for time values t from the negative infinity to the positive infinity, and the parameters must be positive. The vulnerability discovery rate/ intensity function (ω) for the AML VDM as well as its cumulative model (Ω) presented in (6) and (7), respectively.

$$\omega(t) = A\Omega(\mathrm{B} - \Omega) \tag{6}$$

$$\Omega(t) = \frac{B}{BCe^{-ABt} + 1} \tag{7}$$

3.5.4 Normal-based VDM

The Normal-based VDM belongs to the family of distributions with symmetrical intensity/probability density functions. This model presents a distribution with zero skewness that has three parameters: μ is a location parameter, σ is a scale parameter and γ is the total number of vulnerabilities that would eventually be

discovered. The vulnerability discovery rate/intensity function (ω) for the Normal-based VDM as well as its cumulative model (Ω) presented in (8) and (9), respectively.

$$\omega(t) = \frac{\gamma e^{-\frac{(t-\mu)}{s}}}{s\left(1 + e^{-\frac{(t-\mu)}{s}}\right)^2} \tag{8}$$

$$\Omega(t) = \frac{\gamma}{1 + e^{-\frac{(t-\mu)}{s}}} \tag{9}$$

The Normal-based VDM has lighter tails on both sides in comparison to the logistic distribution used for the AML model. For a dataset with fewer vulnerabilities discovered at the beginning and at the end of a discovery process, the Normal VDM might be a better fit than the AML model [25].

3.5.5 Younis Folded VDM

The normal distribution is symmetric around its mean and is defined for a random variable that takes values from -inf to +inf. In some cases, a distribution is needed that has no negative values. Folded distributions are kinds of asymmetrical models obtained by folding the negative values into the positive side of the distribution. The folded distribution has been found usable in industrial practices such as measurement of flatness and straightens.

In the Younis folded VDM [54] vulnerability discovery starts at time t = 0 which corresponds to the release time of the software. In this model, t represents the calendar time, τ is a location parameter, σ is a scale parameter, and γ represents the number of vulnerabilities that will be eventually discovered. The second term in its cumulative $\Omega(t)$ equation (13) represents the part of the distribution folded to the

positive side, which shows the discovery process for the Folded VDM. In the equation erf(.) is the error function. This distribution is defined for $t \geq 0$.

$$\omega(t) = \frac{\gamma}{\sqrt{2\pi}\sigma}\left\{\exp - \left(\frac{(t-\tau)^2}{2\sigma^2}\right) + \exp - \left(\frac{(t+\tau)^2}{2\sigma^2}\right)\right\} \tag{12}$$

$$\Omega(t) = \frac{\gamma}{2}\{\text{erf}\left(\frac{t-\tau}{\sqrt{2\sigma}}\right) + \text{erf}\left(\frac{t+\tau}{\sqrt{2\sigma}}\right)\} \tag{13}$$

Compared to AML, the 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 the vulnerability discovery plot is in linear phase even at the beginning.

3.6 Non-S-shaped Vulnerability Discovery Models

In addition to the S-shaped models described in Section 3.5, we also considered several VDMs that are not S-shaped and cannot be characterized solely by their skewness (note: we used the classification presented in [24]). These models are Power-law, Rescorla Quadratic (RQ), and Rescorla Exponential (RE).

3.6.1 Power-law Software Reliability Growth Models (SRGM-based)

Research has been conducted to find a link between the fault discovery process of a software and the discovery process of its vulnerabilities for modeling purposes [4]. When considering the fault detection process of a software, it is justifiable to conclude that software reliability growth models (SRGMs) and vulnerability discovery models (VDMs) are similar [1]. In such cases, the intensity/rate function can represent the detection rate of vulnerabilities.

When modeling the cumulative number of failures $\Omega(t)$ for software reliability evaluations, models derived from a nonhomogeneous Poisson process (NHPP) are often used. Allodi [55] showed that the vulnerability exploitation may follow a Power-law distribution. However, such models have several assumptions. The main one is that the number of detected vulnerabilities follows a nonhomogeneous Poisson process. In addition, if we consider a software as a repairable system, its intensity function $\omega(\mathrm{t}) = \mathrm{dE}[\Omega(\mathrm{t})]/\mathrm{dt}$, is often, for simplicity, assumed a monotonic function of t. Therefore, in NHPP-based software reliability growth models (SRGMs) or NHPP-based VDMs, the intensity function (the detection rate of software errors/the detection rate of vulnerabilities) is considered to be a monotonic function [56]. The equations associated with the Power-law model are presented in (10) and (11), respectively. This model is continuous over time and has two parameters: α (shape parameter), β (scale parameter).

$$\omega(t) = \frac{\alpha}{\beta}\left(\frac{t}{\beta}\right)^{\alpha-1} \tag{10}$$

$$\Omega(t) = (\beta^{-\alpha}).\, t^{\alpha} \tag{11}$$

3.6.2 Rescorla Exponential (RE)

Rescorla's models are simple exponential and quadratic models that are commonly used and first were introduced by Rescorla in vulnerability analysis area [7]. In equation (14) γ represents the number of vulnerabilities that will be eventually discovered, and λ represents the detection rate of software errors/the detection rate of vulnerabilities.

$$\Omega(t) = \gamma\left(1 - e^{-\lambda t}\right) \tag{14}$$

3.6.3 Rescorla Quadratic (RQ)

Equation (15) is a simple quadratic model introduced by Rescorla [7]. Parameters A and B are empirical constants and directly estimated from the dataset.

$$\Omega(t) = \frac{At^2}{2} + Bt \tag{15}$$

3.7 Summary

In this chapter, we introduced the datasets used in this thesis. We presented the models used for our analysis in two categories of S-shaped VDMs, and non S-shaped VDMs. In the next chapter, we will apply these models on the vulnerability datasets and compare their curve-fitting and prediction capabilities.

Chapter 4: Non Cluster-based Vulnerability Assessment

4.1 Introduction

In this chapter, we will apply the VDMs introduced in Chapter 3 on the vulnerability datasets associated with the operating systems and web browsers also introduced in Chapter 3. Then, we will compare the curve-fitting and prediction capability of these models and will investigate which models perform better in a given situation. Finally, we will present some guidelines to model vulnerability discovery data based upon common VDMs.

4.2 Motivation

Among the research conducted on vulnerability modeling, few studies have tried to provide a guideline about which model should be used in a given situation. In other words, assuming the vulnerability data is provided, the research question addresses the following. Is there any features in the vulnerability data that could be used for identifying the most appropriate models for that dataset? What models are more accurate for vulnerability discovery process modeling?

In this chapter, we make the following contributions:

-We compare the curve fitting and prediction capabilities of eight VDMs (i.e. one right-skewed distribution-based model, one flexible-skewed distribution-based model, three symmetric distribution-based models, one Power-law model, one exponential model, and one quadratic model) on two types of software (i.e. OSs and Web Browsers).

- We present some guidelines to model vulnerability discovery data based upon common VDMs.

- Based upon our findings from estimation results, we show that the Gamma-based VDM was the most accurate model for the datasets by being better in 62.5% of the cases.

- We also show that based upon our findings from prediction results, the Power-law model provides most accurate predictions for the datasets by performing better than other models in 62.5% of the cases.

4.3 Analysis

For each software, the variable we used in this research is the cumulative number of vulnerabilities reported in 30 days' time intervals. In other words, we splitted the study period associated with a given software into intervals of 30 days, and counted the total number of vulnerabilities detected in each interval. Then, we accumulated the total number of detected vulnerabilities for each interval. For curve fitting, the eight models were fitted to the eight datasets (for the four OSs and the four web browsers) using a regression method described in [24]. To avoid overfitting, 10-fold cross validation was also conducted using Python's sklearn library [57]. The analysis of the prediction capability is done for 2016, 2017, and 2018. During the training period (before 2016), all the available data was used to estimate model parameters. The estimated final values for each interval produced by the eight models were compared with the actual number of vulnerabilities to calculate the prediction accuracy.

4.3.1 Curve-fitting error indicators

We applied the Chi-square (χ^2) goodness of fit test [24] to see how well each model fits the datasets. The χ^2 statistic is calculated using the following equation:

$$\chi^2 = \sum_{i=1}^{N} \frac{(S_i - O_i)^2}{O_i} \tag{16}$$

where S_i and O_i are the simulated and real observed values at i^{th} interval, respectively. N is the number of observations. For the fit to be acceptable, the corresponding χ^2 critical value should be greater than the χ^2 statistic for the given alpha level and degrees of freedom. We selected an alpha level of 0.05. The null hypothesis indicates that the actual distribution is well described by the fitted model. Hence, if the p-value of the χ^2 test is below 0.05, then the fit will be considered unsatisfactory. A p-value closer to 1 indicates a better fit.

R^2 is another fitting statistic used in regression analysis [58]. A R^2 value close to 1 indicates a good fit. R^2 values are usually used in linear regression analysis [59] and might lead to some inaccuracy problem in the case of non-linear regression analysis. However, since this metric has been used in previous studies, we decided to also calculate it so that our results could be compared with these studies.

The root mean square error (RMSE) is often used in research to calculate fitting errors. However, Mentaschi et al. [60] showed that for some applications (e.g., high fluctuation of real data) the lower values of RMSE are not always a reliable indicator of the accuracy of simulations. Hence, a corrected estimator HH was proposed by Hanna and Heinold [61]:

$$HH = \sqrt{\frac{\sum_{i=1}^{N}(S_i - O_i)^2}{\sum_{i=1}^{N} S_i O_i}} \tag{17}$$

where S_i is the i^{th} simulated data, O_i is the i^{th} observation and N is the number of observations (the time blocks used for simulation). The closer to zero HH is, the more accurate the model.

4.3.2 Prediction error indicators

We calculated two normalized predictability measures, average error (AE) and average bias (AB) [25]. AE is a measure of how well a model predicts throughout the test phase, and AB indicates the general bias of the model, which assesses its tendency to overestimate or underestimate the number of discovered vulnerabilities. AE and AB are defined as:

$$AE = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{\Omega_t - \Omega}{\Omega}\right| \tag{18}$$

$$AB = \frac{1}{n}\sum_{t=1}^{n}\frac{\Omega_t - \Omega}{\Omega} \tag{19}$$

where n is a total number of intervals (one per recorded detection date) over the prediction period, and Ω is the actual number of total vulnerabilities, whereas Ω_t is the estimated number of total vulnerabilities at time t. AB can be positive (for overestimation) or negative (for underestimation), while AE is always positive.

4.4 Curve-Fitting Results

The eight models were fitted to the eight datasets (for the four OSs and the four web browsers) using a regression method described in [24]. To avoid overfitting, 10-fold cross validation was also conducted using Python's sklearn library [57]. In each case, the model that has the smallest value of HH is selected as the best fitting model and highlighted in green. If the HH values of two models were equal, we used the RMSE values to differentiate them. The model with a higher value of RMSE becomes the second best model, and highlighted in yellow. For models with equal HH and almost equal RMSE (difference<=0.001), both models were highlighted in green.

4.4.1 Operating systems

The data and the fitted curves for the OSs are shown in Figure 3. Table 3 contains the χ^2 goodness of fit test p-values, the values of R^2, RMSE and HH for the four OSs.

For the OSs, the Power-law model is only statistically sound with p-values greater than 0.05 for Linux. All the other VDMs are each significant in two cases. When comparing HH values for Windows, IOS, and Linux, we observe that Gamma-based and Weibull-based VDMs performed as well as the Power-law model. However, for Windows and IOS, they are not statistically significant based on the generated p-values.

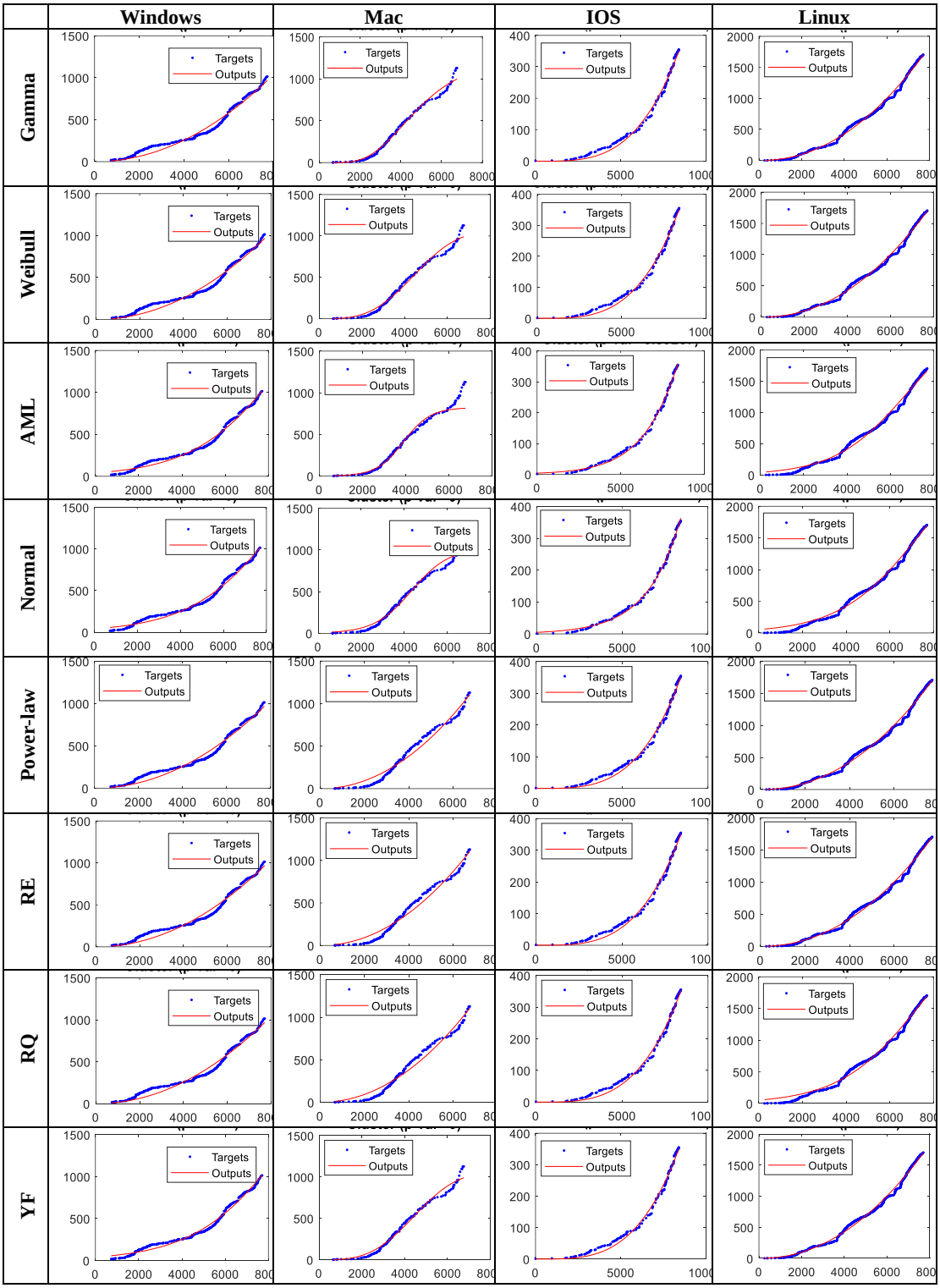

Figure 3. Fitted models for operating systems

For Windows, the YF VDMs provides the best fit since it has smallest HH and RMSE values as well as p-values greater than 0.05. For Mac, the Gamma and Weibull-based VDMs as well as Power-law model and YF VDM led to statistically sound fits. However, the Gamma-based VDM was selected as the best fit due to HH and RMSE values lower than the other models. For IOS, all the models except RQ are statistically sound. The models with smaller fitting errors (AML and Normal-based VDMs) were selected as the best fits. For Linux, like Windows and IOS, all the models except RE and YF are statistically sound. However, the Gamma and Weibull-based VDMs as well as the Power-law have smallest HH values and relatively equal RMSEs. Thus, they are the most accurate fits.

In addition, from Figure 3 we found that for the considered OSs, the vulnerability discovery intensity function is still increasing. These software appear to be in phase 2 (cf. Figure 2) and many vulnerabilities in these products are yet to be discovered.

Table 3: CURVE FITTING ACCURACY FOR OSS

	Windows				**Mac**			
	p-value	***R^2***	***RMSE***	***HH***	***p-value***	***R^2***	***RMSE***	***HH***
Gamma	0.686	0.998	38.549	0.033	0.193	0.993	52.139	0.061
Weibull	0.936	0.998	33.010	0.028	0.703	0.991	57.979	0.068
AML	0.956	0.999	30.643	0.026	0.001	0.988	67.885	0.080
Normal	0.956	0.999	30.704	0.026	0.001	0.987	68.033	0.080
Power-law	0.466	0.994	59.757	0.051	0.058	0.984	76.679	0.090
RE	0.026	0.983	102.871	0.088	0.000	0.968	108.229	0.128
RQ	0.888	0.995	58.298	0.050	0.002	0.988	67.651	0.080
YF	0.200	0.999	27.002	0.023	0.193	0.990	60.709	0.071
	IOS				**Linux**			
	p-value	***R^2***	***RMSE***	***HH***	***p-value***	***R^2***	***RMSE***	***HH***
Gamma	0.981	0.995	9.727	0.054	0.640	0.985	105.371	0.085
Weibull	0.981	0.995	9.379	0.052	0.640	0.985	105.264	0.085
AML	0.990	0.997	7.391	0.041	0.630	0.977	129.095	0.105
Normal	0.990	0.997	7.403	0.041	0.630	0.977	129.338	0.105
Power-law	0.642	0.995	9.357	0.052	0.640	0.985	105.234	0.085
RE	0.150	0.998	9.044	0.053	0.006	0.983	110.346	0.089
RQ	0.000	0.973	22.993	0.128	0.222	0.985	105.901	0.086
YF	0.370	0.998	6.334	0.035	0.006	0.982	114.012	0.092

4.4.2 Web browsers

The data and the fitted curves for the web browsers, are shown in Figure 4. Table 4 contains the χ^2 goodness of fit test p-values, the values of R^2, RMSE and HH for the four web browsers.

For IE, all the models are statistically sound and YF has the lowest HH value. Thus, it was selected as the best fit. For Safari, although all the models except RE are statistically sound, Gamma-based VDM provided a better fit than other models based on the HH values. For Firefox, Gamma-based VDMs is the best model since they have p-values greater than 0.5 and smaller fitting error (HH and RMSE) than the other models. For Chrome, Gamma-based VMD provided the best fit due to having the smallest HH and RMSE values.

Table 4: CURVE FITTING ACCURACY FOR WEB BROWSERS

	IE				Safari			
	p-value	***R^2***	***RMSE***	***HH***	***p-value***	***R^2***	***RMSE***	***HH***
Gamma	0.624	0.977	50.137	0.098	0.245	0.993	17.915	0.056
Weibull	0.624	0.978	50.051	0.098	0.739	0.993	18.945	0.059
AML	0.403	0.975	53.035	0.104	0.050	0.991	21.297	0.066
Normal	0.403	0.975	53.151	0.104	0.050	0.991	21.365	0.066
Power-law	0.624	0.978	50.041	0.098	0.378	0.984	27.731	0.086
RE	0.403	0.983	53.307	0.099	0.012	0.969	38.621	0.121
RQ	0.624	0.978	59.149	0.099	0.549	0.984	27.413	0.085
YF	0.285	0.983	44.100	0.086	0.986	0.992	19.629	0.061
	Firefox				**Chrome**			
	p-value	***R^2***	***RMSE***	***HH***	***p-value***	***R^2***	***RMSE***	***HH***
Gamma	0.378	0.998	18.683	0.028	0.368	0.998	20.094	0.036
Weibull	0.307	0.998	18.826	0.029	0.368	0.994	32.551	0.059
AML	0.250	0.993	34.519	0.053	0.690	0.995	29.213	0.053
Normal	0.250	0.993	34.630	0.053	0.690	0.995	29.415	0.053
Power-law	0.307	0.998	20.032	0.031	0.240	0.962	83.910	0.153
RE	0.115	0.992	36.779	0.056	0.000	0.937	107.974	0.199
RQ	0.193	0.996	24.914	0.038	0.000	0.973	70.106	0.127
YF	0.150	0.996	24.543	0.037	0.400	0.996	25.641	0.046

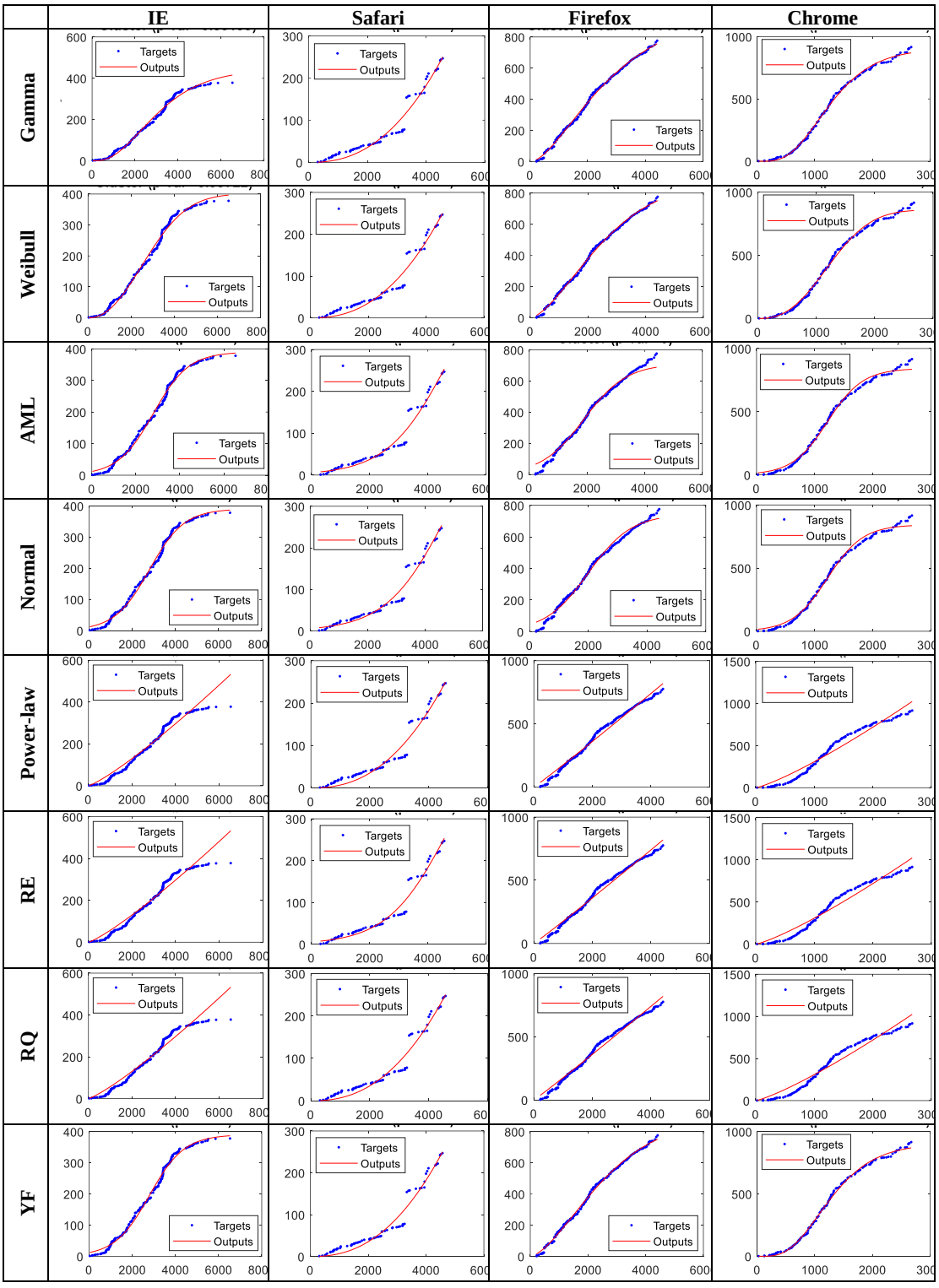

Figure 4. Fitted models for web browsers

In Figure 4, the vulnerability data associated with IE and Chrome show a saturation phase by the end of the learning phase, which means their discovery intensity function has a decreasing trend at the end (i.e., the rate of discovery of new vulnerabilities is predicted to decrease). Thus, these products appear to be in the saturation phase (cf. Figure 2). For Safari, and Firefox, the discovery intensity functions are increasing and constant, respectively.

4.3.4 Summary of Estimation Results

Overall, in terms of curve fitting, considering the OSs, Gamma and YF VDMs were the best models in 50% of the cases, while the Weibull-based VDM and Power-law model were the best models in 25% of the cases. Considering the web browsers, Gamma-based VDM provided the best fits in 75% of the cases, whereas the YF VDM was the best model in 25% of the cases. The other VDMs were better in none of the cases. However, considering the OSs and the web browsers (8 cases), Gamma, Weibull, AML, Normal, Power law, RE, RQ, and YF performed well in 5 (62.5%), 1(12.5%), 0(0%), 0(0%), 1(12.5%), 0(0%), 0(0%), and 3(37.5%) cases, respectively.

Therefore, based upon our findings from estimation results, **the Gamma-based VDM was the most accurate model for the datasets we analyzed**.

4.5 Prediction Results

The analysis of the prediction capability is initiated after two-thirds of the time period from the beginning of the vulnerability discovery process. During the training period, all the available data was used to estimate model parameters. The estimated final values for each 30-day interval produced by the eight models were compared with

the actual number of vulnerabilities to calculate the prediction accuracy. The model that has the smallest value of AE and p-value≥0.05 was selected as having the best prediction capability and is highlighted in green. Regarding p-values, we used * to show the models with p<0.05. If the AE values of two models were equal, we selected the best model based upon the AB and HH values. The model with a higher bias or HH was selected as the second best model and is highlighted in yellow.

4.5.1 Operating systems

The normalized error values $((\Omega_t - \Omega)/\Omega)$ for the OSs are shown in Figure 5. Table 5 presents the values of AE, AB, R^2 and HH for the four OSs in our study. Comparing prediction capabilities for Windows, we found that the Power-law model has the smallest AE, AB, R^2, and HH values.

Table 5: PREDICTION ACCURACY FOR OSS

	Windows				**Mac**			
	AE	***AB***	***R^2***	***HH***	***AE***	***AB***	***R^2***	***HH***
Gamma	0.063	-0.061	271.445	0.095	0.218	-0.218	595.482	0.263
Weibull	0.091	-0.091	367.988	0.131	0.233	-0.233	640.863	0.287
AML	0.138	-0.138	511.292	0.187	0.278*	-0.278	761.332	0.351
Normal	0.138	-0.138	511.292	0.187	0.278*	-0.278	761.328	0.351
Power-law	0.037	0.025	119.646	0.040	0.074	-0.074	198.921	0.080
RE	0.106*	0.106	314.463	0.100	0.024*	0.017	95.639	0.037
RQ	0.039	0.030	125.961	0.042	0.082*	-0.082	221.121	0.090
YF	0.114	-0.114	438.161	0.158	0.256	-0.256	703.498	0.320
	IOS				**Linux**			
	AE	***AB***	***R^2***	***HH***	***AE***	***AB***	***R^2***	***HH***
Gamma	0.018	0.000	15.316	0.025	0.268	-0.268	1382.822	0.354
Weibull	0.019	0.005	17.105	0.027	0.267	-0.267	1378.715	0.352
AML	0.076	0.076	57.333	0.088	0.272	-0.272	1423.009	0.366
Normal	0.076	0.076	57.332	0.088	0.272	-0.272	1423.002	0.366
Power-law	0.019	0.006	17.366	0.028	0.267	-0.267	1377.862	0.352
RE	0.131	0.131	98.947	0.148	0.190*	-0.190	987.650	0.239
RQ	0.154*	-0.154	98.230	0.172	0.278	-0.278	1431.530	0.369
YF	0.092	0.092	70.902	0.108	0.240*	-0.240	1248.693	0.313

For Mac, the Power-law model has the smallest values of AE, AB and HH. For IOS, the Gamma-based VDM has the smallest value of AE. For Linux, the Weibull-

based VDM and Power-law model have the best results. Note that in Table 5 negative values of AB indicate that the model may underestimate the total number of discovered vulnerabilities.

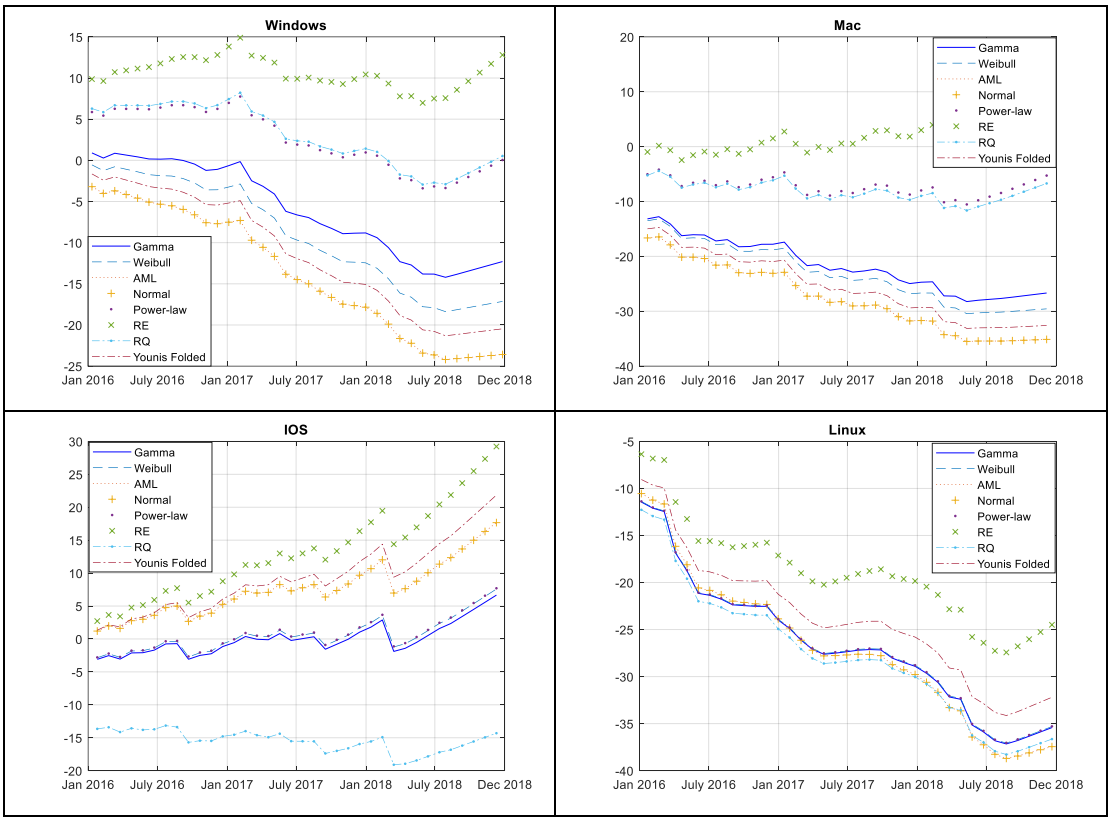

Figure 5. Normalized prediction error values for the models (OSs)

4.5.2 Web browsers

The normalized error values $((\Omega_t - \Omega)/\Omega)$ associated with the web browsers are shown in Figure 6. Table 6 presents the values of AE, AB, R^2 and HH for the four web browsers in our study. For IE, YF VDM had the smallest error values. For Safari, the Power-law model had the smallest prediction error values. For Firefox, Weibull-based VDM provided the best prediction capabilities. For Chrome, the power-law model had the smallest error values.

Table 6: PREDICTION ACCURACY FOR WEB BROWSERS

	IE				Safari			
	AE	***AB***	***R^2***	***HH***	***AE***	***AB***	***R^2***	***HH***
Gamma	0.234	-0.234	402.761	0.273	0.156	-0.156	159.467	0.201
Weibull	0.233	-0.233	400.858	0.272	0.187	-0.187	190.325	0.245
AML	0.157	-0.157	270.708	0.175	0.231	-0.231	228.863	0.304
Normal	0.157	-0.157	270.706	0.175	0.231	-0.231	228.863	0.304
Power-law	0.233	-0.233	400.719	0.271	0.030	0.026	32.144	0.037
RE	0.149	-0.149	255.874	0.164	0.133*	0.133	130.867	0.141
RQ	0.232	-0.232	398.766	0.270	0.041	0.040	41.458	0.047
YF	0.141	-0.141	242.287	0.155	0.211	-0.211	212.086	0.278
	Firefox				**Chrome**			
	AE	***AB***	***R^2***	***HH***	***AE***	***AB***	***R^2***	***HH***
Gamma	0.051	0.035	103.041	0.066	0.281	-0.281	527.771	0.367
Weibull	0.049	0.031	98.984	0.064	0.317	-0.317	590.544	0.422
AML	0.081	-0.081	162.703	0.112	0.307	-0.307	571.317	0.405
Normal	0.081	-0.081	162.703	0.112	0.307	-0.307	571.314	0.405
Power-law	0.069	0.067	140.551	0.089	0.167	0.167	355.845	0.191
RE	0.161	0.161	287.176	0.174	0.364*	0.364	776.614	0.383
RQ	0.096	0.096	180.913	0.113	0.077*	0.077	181.493	0.102
YF	0.051	-0.032	103.839	0.069	0.304	-0.304	567.935	0.402

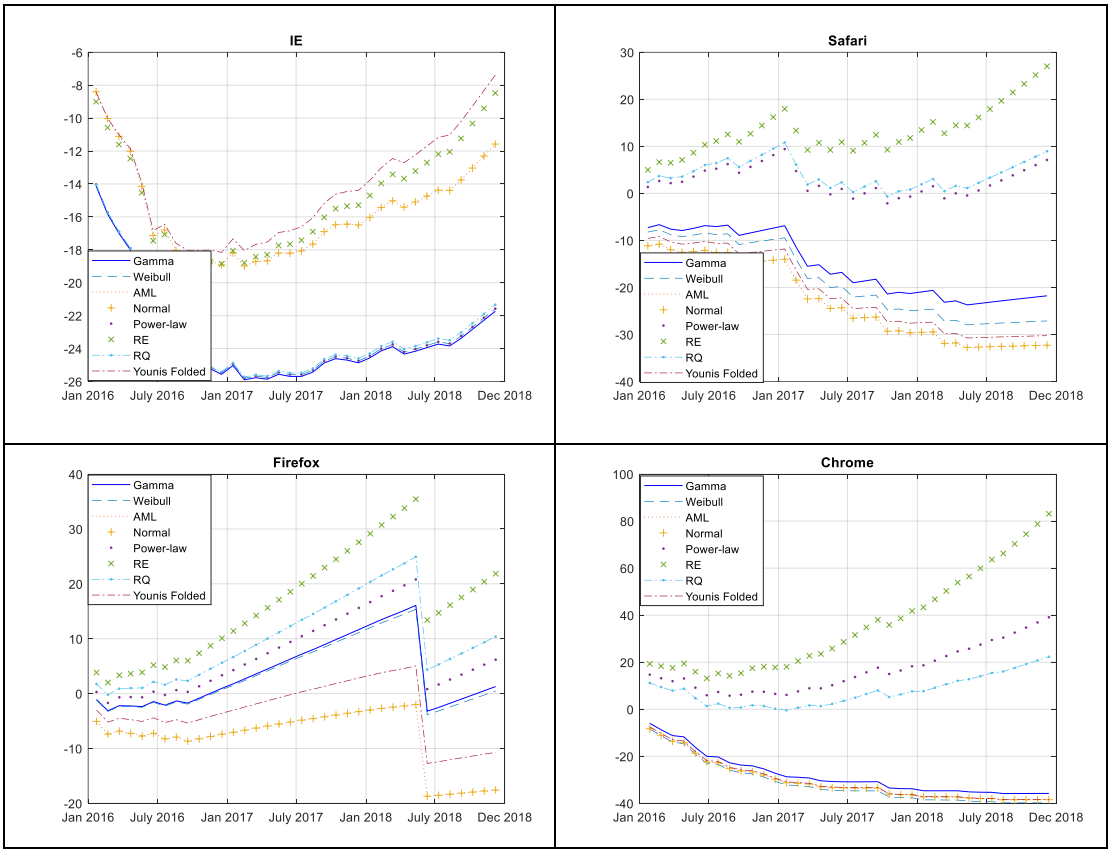

Figure 6. Normalized prediction error values for the models (web browsers)

4.5.4 Summary of prediction results

Overall, in terms of prediction, considering the OSs, the Power-law model performed better in 3 (75%) cases. Gamma and Weibull-based VDMs provided better prediction results in 1 (25%), and 1 (25%) cases, respectively. Normal-based, AML, RE, RQ, and YF VDMs were selected in none of the cases. Considering web browsers, the Power-law model led to better predictions in 2 (50%), while Weibull-based VDM along with YF VDM each were selected as the best predictor in 1 (25%) case. However, considering the OSs and the web browsers together (8 cases), Gamma, Weibull, AML, Normal, Power law, RE, RQ, and YF VDMs provided satisfactory prediction results in 1 (12.5%), 2 (25%), 0 (0%), 0 (0%), 5 (62.5%), 0 (0%), 0 (0%), and 1 (12.5%) cases, respectively.

Then, based upon our findings from prediction results, **the Power-law model yielded most accurate predictions for the datasets we analyzed**.

Please remember that all the conclusions we draw from this research and their validity are limited by our database uncertainties.

4.6 Discussion

Based on our results, we found that a model's ability to provide a good fit does not necessarily guarantee superior prediction capabilities. To the best of our knowledge, there are few studies that have tried to provide guidance about which model should be used in a given situation. Joh et al. [25] investigated the relationship between the performance of five S-shaped VDMs (i.e., AML, Weibull, Gamma, Normal, and Beta) and the skewness in vulnerability datasets for eight software.

Comparing our findings with those from Joh et al. [25], in terms of prediction, we didn't find any cases out of eight cases among asymmetrical datasets where the AML VDM performed better than right-skewed distribution models - Joh et al. [25] stated that AML model performs better than some right-skewed distribution models in terms of prediction when the vulnerability discovery datasets are asymmetrical.

The model tendency to overestimate or underestimate the results is another factor, which plays an important role in the procedure of model selection. We evaluated the bias values (AB) and explained that the final decision is up to the researcher to choose the best model based upon his/her priorities. However, from a security point of view, it is better to choose a model, which provides more conservative prediction results, if it has justifiable error values. In the current study, among the models that were selected as the best predictors, seven models provided overestimated results. Other selected models underestimated the number of vulnerabilities.

4.7 Limitations

There are several limitations to our work that prevent us from making more general conclusions. One limitation is with regard to the uncertainty of the databases we used. Vulnerability databases usually might have some uncertainty regarding variables associated with the reported vulnerabilities such as published dates.

Another limitation is associated with using SRMs (the Power-law model) as VDMs. Software reliability models usually assume that the time between failures represents total usage time of that product. What we are using is calendar time, which may not be a good proxy for usage. One important difference with security studies is the difficulty in estimating the "attacker effort" - the total amount of time that an

attacker spends in finding a vulnerability - which is something that is not needed in the context of reliability (we assume the users accidentally encounter faults that lead to failures, hence usage time is a good enough proxy for time between failures). A useful discussion of this is given in [30].

We have used all the vulnerabilities for all the versions of the products in our study. While number of studies utilize vulnerability data associated with separate version of software (e.g. Windows 7) on which to apply VDMs, there are papers that consider all versions of a software together [25], [62]. The first group expects that each version of a given software is an independent and all around characterized item, yet distinguishing the sources of reliance in vulnerability data is not a simple task.

4.8 Summary

In this chapter, we applied the models introduced in Chapter 3 on the vulnerability datasets associated with the operating systems and web browsers also discussed in Chapter 3. Then, we compared the curve-fitting and prediction capability of these models and investigated which models perform better in a given situation. Finally, we presented some guidelines for using eight common VDMs to model vulnerability discovery data based on a given dataset. In next chapter, we will test whether a clustering approach improves the accuracy of the curve-fitting/prediction results.

Chapter 5: Clustering

5.1 Introduction

In this chapter, we will test whether a clustering approach improves the accuracy of the curve-fitting/prediction results. We will focus on how clusters were created for the different vulnerability datasets associated with the operating systems and web browsers discussed in Chapter 3 and apply the five S-shaped VDMs and three non-S-shaped VDMs also introduced in Chapter 3 on them. Then, we will compare the curve-fitting and prediction capability of these models and will investigate which models perform better in a given situation as well as compare them with the results obtained without clustering (from Chapter 4).

5.2 Motivation

Several studies have applied SRMs/VDMs to estimate times between public reports of vulnerabilities [3], [6]–[8], [17], [18]. In all the studies we are aware of, curve fitting or/and prediction capabilities were estimated using all the vulnerabilities together. We postulate that such analysis may miss some trends that apply to separate categories of vulnerabilities, rather than all the vulnerabilities together.

Moreover, SRMs assume vulnerability detection to be an independent process. However, this process might not be independent due, for example, to the discovery of a new type of vulnerability that might prompt attackers to look for similar vulnerabilities [5]. This assumption may lead to sub-optimal predictions on the next reporting date of a vulnerability, or the total number of new vulnerabilities reported in

the next time interval. One way to mitigate these issues is to split vulnerabilities into separate clusters and ensure that the clusters are independent.

In this chapter, we make the following contributions:

- We present an approach that uses existing clustering techniques to group vulnerabilities into distinct clusters, leveraging the textual information reported in these vulnerabilities as a basis for constructing the clusters.

- Our approach uses existing VDMs to make predictions on the number of new vulnerabilities that will be discovered in a given time period for each cluster for a given OS/web browser;

- Our approach also superposes the VDMs used for each cluster together into a single model for predicting the number of vulnerabilities that will be discovered in a given time period for a given OS/web browser

- We show that, based upon our findings from prediction results, comparing modeling strategies, the results with clustering were more accurate by performing better that without clustering approach in 58% of the cases.

5.3 Data Processing

For each software, we included all the vulnerabilities reported for any of its versions. For instance, all the vulnerabilities reported for mac_os, mac_os_server, mac_os_x, and mac_os_x_server were put together to create a vulnerability database for Mac.

To prepare the data for the clustering phase, we used text information within vulnerabilities reports to label the vulnerabilities. The keywords for labelling (e.g., denial, injection, buffer, execute) were extracted from these reports. Tables 43 and 50

in Appendix A show the total number of vulnerabilities as well as the number of labelled and non-labelled (vulnerabilities without any associated text information in the database) vulnerabilities for the datasets (OSs and web browsers). For the labelled vulnerabilities, we indicate the number and proportion of vulnerabilities associated with a specific keyword. Note that vulnerabilities can be labelled with more than one keyword.

For cluster analysis, we need to ensure that the features (keywords) are not correlated. Therefore, we checked the Pearson correlation coefficient for every two keywords per dataset. When we found statistically significant correlation, we merged the correlated keywords with a title which included both terms. For instance, due to the high correlation of .99 (p-value<0.001, $H_0: \rho = 0$) between "Execute" and "Code" for all the datasets, these terms were treated as "Execute Code". The same applied for the keywords "SQL" and "Injection". No other significant correlation was observed. Figure 7 shows the diagram of our clustering approach.

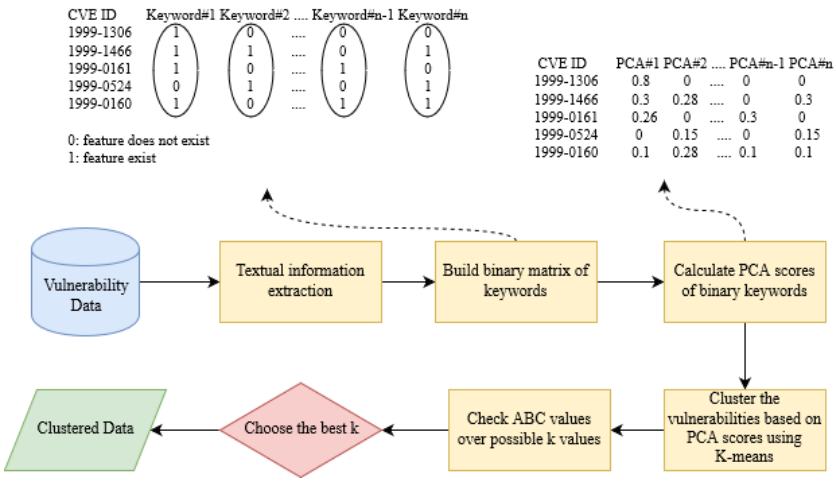

Figure 7. Diagram of the presented clustering approach

5.4 Clustering Method

We used the HPCLUS (High Performance Clustering) procedure in SAS 9.4 with the k-means and k-modes algorithms for clustering nominal input variables. This procedure uses the least square method in k-means to compute cluster centroids. Each iteration reduces the criterion (e.g., the least squared criterion for Euclidean distance) until convergence is achieved or the maximum iteration number is reached [63]. Additionally, we set our method to cluster the data based upon the associated principal component analysis (PCA) scores derived from the linear combinations of binary attributes for each dataset. PCA reduces the number of features that might be correlated to independent linear combinations of them [40].

To estimate the best number of clusters the aligned box criterion (ABC) method was used. Among other existing methods, the cubic clustering criterion (CCC) is a common metric which is usually used in clustering applications to find the most suitable number of clusters [64]. In addition, Tibshirani et al. [65] introduced a gap statistics method, which leverages Monte Carlo simulation for finding the best number of clusters in a database. However, it has been found that the ABC method improves the CCC and gap statistics methods by leveraging a high-performance machine-learning based analysis structure [63]. Within-cluster dispersion is used as an error measure (also called a ‘Gap’) by the ABC method [65]. In order to find the best number of clusters, we applied the ABC method and compared the calculated Gap values over a range of possible k values. The best number of clusters occurs at the maximum peak value in Gap (k) [63].

After clustering, the most frequent keywords were selected to name the clusters with respect to the keywords' weights information provided in Appendix A. We assumed that the keywords which covered at least 60% of vulnerabilities in each cluster can be good representatives of relative clusters. If none of the keywords reached the weight threshold of 0.6 in a cluster associated with a given software, the keyword with greatest weight was selected as the cluster's label.

5.4.1 Operating systems

We obtained 6, 6, 7, 7 clusters for Windows, Mac, IOS, and Linux, respectively. The list of keywords associated with each of the six / seven clusters for Windows, Mac, IOS, and Linux provided in Appendix A. Since none of the keywords reaches the weight threshold of 0.6 in the fifth cluster associated with Mac, the keyword with the greatest weight (Execute Code) was selected as the cluster's label. Table 49 shows the cluster summaries for the OSs. All the OSs have one cluster with a similar name. There are also similarly named clusters within some OSs. However, after having analyzed their linear correlation, we did not find any significant relationship based on the Pearson correlation test.

5.4.2 Web browsers

Following the same clustering approach we explained for the OSs, we obtained the following number of clusters: Internet Explorer (5), Safari (3), Firefox (5), and Chrome (5). More details about the clusters and frequency of the keywords associated with are provided in Appendix A. The cluster summaries for the browsers are shown in Table 56.

5.5 Analysis

In repairable systems with only one type of failure, the intensity function $\omega(t)=dE[\Omega(t)]/dt$ is often assumed to be a monotonic function of t. Similarly, for most SRMs and VDMs, the intensity function (the detection rate of software errors/vulnerabilities) is considered to be a monotonic function [56].

Let us expand the discussion for a software when there exists more than one type of error. When any type of error independently causes the software normal function to be compromised, then the superposition model represents the software failures. Let us assume that we are dealing with vulnerabilities classified into independent clusters. Considering a given model (NHPP Power-law or a distribution-based VDM), let $\Omega_j(t)$ denote the mean cumulative number of vulnerabilities from the j^{th} cluster in (0 t], with intensity function $\omega_j(t|\alpha_j,\beta_j)$ where the function form of $\omega_j(t|\alpha_j,\beta_j)$ is given and the values of the parameters α_j, $\beta_j$are unknown. It is assumed that the number of vulnerabilities from any j^{th} cluster $\Omega_j(t)$, $j=1,2,\dots J$ is independent. A process $\Omega(t)=\sum_{j=1}^{J}\Omega_j(t)$, which counts the total number of vulnerabilities in the interval (0 t] for the superposition model, is also a same-type model (NHPP Power-law/distribution-based model) with an intensity function $\omega(t|\alpha,\beta)=\omega_1(t|\alpha_1,\beta_1)+\cdots+\omega_J(t|\alpha_J,\beta_J)$, where $\alpha=\{\alpha_1,\dots,\alpha_J\}$, $\beta=\{\beta_1,\dots,\beta_J\}$. Since the superposition model keeps its type (all intensity functions are of the same type), the associated superposition model can be applicable [56]. For instance, considering the NHPP Power-law model, the equations become:

$$\omega_j(t|\alpha_j,\beta_j) = \frac{\alpha_j}{\beta_j}\left(\frac{t}{\beta_j}\right)^{\alpha_j-1} = \frac{\alpha_j t^{\alpha_j-1}}{{\beta_j}^{\alpha_j}} \quad (20)$$

$$\Omega(\mathrm{t}) = \int_0^t \sum_{j=1}^{J} \omega_j(t|\alpha_j,\beta_j)\, dt\ , \quad \alpha_j > 0, \quad \beta_j > 0 \quad (21)$$

In this chapter, we investigate the assessment results when relaxing the monotonicity assumption of the intensity function that is prevalent in SRMs and VDMs. Selecting a given VDM/SRM, we considered two approaches for each model. The first approach uses non-clustered data (including only the labeled vulnerabilities) (cf. Chapter 4). The second approach is the superposition of the same model fitted to the clustered data (only the labelled vulnerabilities can be used to create the clusters), which relaxes the monotonicity assumption of the intensity function.

For each software, the analysis was done in two steps. First, we used the training data (note: the variable we used is the total number of vulnerabilities detected on 30 days' time interval) to find the model parameters from the process of fitting models to the data (clustered and non-clustered). In other words, similar to what we described in Chapter 4, for the vulnerability data in each cluster, we divided the time axis into 30-day intervals (t=0 is associated with the vulnerability with the earliest published date), and counted the cumulative frequency of vulnerabilities detected in each interval. Non-homogeneity of the clusters was also validated by looking at Laplace-trend test results provided by MiniTab 16 to see whether there were meaningful trends in clusters. Second, we used the estimated parameters and the models, and simulated corresponding mean cumulative function (MCF) (one MCF for clustered data, and one

for non-clustered data) over the study period. For non-clustered data, we used the results from Chapter 4.

When comparing the models with clustering versus those without-clustering, for each software, the best approach is highlighted in green. In addition, for prediction results, the negative AB values are colored in red (i.e., the associated model underestimated the results). All the accuracy metrics, in terms of curve fitting/prediction, are the same than in Chapter 4. For the models that used the clustered data, the first and second best models are highlighted in yellow. Overall, green is used to indicate the best modeling approach while yellow indicates the best models among the models when applying clustering.

5.6 Curve-Fitting Results

In this section we provide the results regarding curve-fitting capabilities of the models (comparing the results between clustered data and non-clustered data, and comparing the results generated from clustering).

5.6.1 Operating systems

Tables 7-10 contain the Chi-square goodness of fit test for the clustering-based MCF and the MCF without clustering, the values of R^2, RMSE and HH for the vulnerabilities of the four operating systems in our study. For Windows and IOS, the MCF without clustering yielded more accurate results in all the cases with p-values>0.05 because of smaller RMSE and HH values. For Mac, only in one case, the MCF with clustering performed better than the MCF without clustering. For Linux, the MCF without clustering yielded more accurate results in two cases. Besides, among the

models uses non-clustered data and clustered data, in five cases neither of the MCFs were statistically sound. We cannot compare these cases and call them as "invalid" cases.

In addition, among the models with clustering, for Windows, the Gamma and Weibull-based VDMs provide the best fits since they have smaller HH and RMSE values. For Mac and IOS, the Gamma-based VDM leads to HH and RMSE values lower than other VDMs. For Linux, the AML and Normal-based VDMs led to the smallest and relatively equal HH values and provide the most accurate fits.

Table 7: CURVE FITTING ACCURACY FOR WINDOWS

Estimation Windows	With clustering				Without clustering			
	p-value	*R-sq*	*RMSE*	*HH*	*p-value*	*R-sq*	*RMSE*	*HH*
Gamma	0.233	0.983	63.550	0.134	0.686	0.998	38.549	0.033
Weibull	0.233	0.981	63.996	0.135	0.936	0.998	33.010	0.028
AML	0.074	0.982	64.949	0.137	0.956	0.999	30.643	0.026
Normal	0.074	0.982	64.845	0.137	0.956	0.999	30.704	0.026
Power-law	0.098	0.990	101.389	0.202	0.466	0.994	59.757	0.051
RE	0.000	0.980	101.425	0.220	0.026	0.983	102.871	0.088
RQ	0.000	0.980	102.100	0.278	0.888	0.995	58.298	0.050
YF	0.074	0.982	64.845	0.137	0.200	0.999	27.002	0.023

Table 8: CURVE FITTING ACCURACY FOR MAC

Estimation Mac	With clustering				Without clustering			
	p-value	*R-sq*	*RMSE*	*HH*	*p-value*	*R-sq*	*RMSE*	*HH*
Gamma	0.058	0.985	118.583	0.236	0.193	0.993	52.139	0.061
Weibull	0.058	0.984	121.376	0.242	0.703	0.991	57.979	0.068
AML	0.000	0.986	121.620	0.244	0.001	0.988	67.885	0.080
Normal	0.000	0.986	120.570	0.241	0.001	0.987	68.033	0.080
Power-law	0.072	0.987	131.265	0.247	0.058	0.984	76.679	0.090
RE	0.059	0.986	129.168	0.245	0.000	0.968	108.229	0.128
RQ	0.000	0.890	150.230	0.360	0.002	0.988	67.651	0.080
YF	0.114	0.986	121.621	0.244	0.193	0.990	60.709	0.071

Table 9: CURVE FITTING ACCURACY FOR IOS

Estimation IOS	With clustering				Without clustering			
	p-value	*R-sq*	*RMSE*	*HH*	*p-value*	*R-sq*	*RMSE*	*HH*
Gamma	0.170	0.992	399.504	1.130	0.981	0.995	9.727	0.054
Weibull	0.175	0.993	404.167	1.138	0.981	0.995	9.379	0.052
AML	0.076	0.990	450.814	1.224	0.990	0.997	7.391	0.041
Normal	0.076	0.991	458.197	1.237	0.990	0.997	7.403	0.041
Power-law	0.055	0.990	405.189	1.140	0.642	0.995	9.357	0.052
RE	0.000	0.987	410.026	1.149	0.150	0.998	9.044	0.053
RQ	0.000	0.987	450.159	1.219	0.000	0.973	22.993	0.128
YF	0.098	0.991	408.237	1.140	0.370	0.998	6.334	0.035

Table 10: CURVE FITTING ACCURACY FOR LINUX

Estimation Linux	With clustering				Without clustering			
	p-value	*R-sq*	*RMSE*	*HH*	*p-value*	*R-sq*	*RMSE*	*HH*
Gamma	0.371	0.994	166.381	0.167	0.640	0.985	105.371	0.085
Weibull	0.371	0.991	129.92	0.134	0.640	0.985	105.264	0.085
AML	0.560	0.990	104.775	0.110	0.630	0.977	129.095	0.105
Normal	0.560	0.994	105.670	0.110	0.630	0.977	129.338	0.105
Power-law	0.392	0.990	128.021	0.148	0.640	0.985	105.234	0.085
RE	0.075	0.992	145.390	0.155	0.006	0.983	110.346	0.089
RQ	0.171	0.992	147.008	0.156.	0.222	0.985	105.901	0.086
YF	0.151	0.994	128.620	0.149	0.006	0.982	114.012	0.092

5.6.2 Web browsers

Tables 11-14 contain the Chi-square goodness of fit test values for the clustering-based MCF and the MCF without clustering, the values of R^2, RMSE and HH for the vulnerabilities of the four web browsers in our study. For all the cases with p-values>0.05, the MCF without clustering led to more accurate results than those from clustering-based MCFs, because of having smaller RMSE and HH values as well as having statistically sound p-values. Besides, among the models uses non-clustered data and clustered data, in three cases neither of the MCFs were statistically sound.

In addition, among the models with clustering, for IE, the Power-law model led to the smallest values of HH and RMSE. For Safari, the AML and Normal-based VDMs provided better fits than other models. For Firefox, Gamma and Weibull-based VDMs are the best models since they have smaller fitting errors (HH and RMSE) than other models. For Chrome, Gamma-based VMD provided the best fit due to smaller HH and RMSE values.

Table 11: CURVE FITTING ACCURACY FOR IE

Estimation IE	With clustering				Without clustering			
	p-value	*R-sq*	*RMSE*	*HH*	*p-value*	*R-sq*	*RMSE*	*HH*
Gamma	0.169	0.997	65.088	0.342	0.624	0.977	50.137	0.098
Weibull	0.469	0.997	64.970	0.341	0.624	0.978	50.051	0.098
AML	0.012	0.997	69.876	0.372	0.403	0.975	53.035	0.104
Normal	0.012	0.997	69.882	0.372	0.403	0.975	53.151	0.104
Power-law	0.162	0.973	56.496	0.285	0.624	0.978	50.041	0.098
RE	0.067	0.997	69.872	0.368	0.403	0.983	53.307	0.099
RQ	0.000	0.990	62.805	0.285	0.624	0.978	59.149	0.099
YF	0.529	0.997	65.079	0.341	0.285	0.983	44.100	0.086

Table 12: CURVE FITTING ACCURACY FOR SAFARI

Estimation Safari	With clustering				Without clustering			
	p-value	*R-sq*	*RMSE*	*HH*	*p-value*	*R-sq*	*RMSE*	*HH*
Gamma	0.057	0.884	125.658	0.878	0.245	0.993	17.915	0.056
Weibull	0.057	0.880	125.476	0.878	0.739	0.993	18.945	0.059
AML	0.080	0.883	98.623	0.763	0.050	0.991	21.297	0.066
Normal	0.080	0.810	96.776	0.769	0.050	0.991	21.365	0.066
Power-law	0.080	0.92	129.947	0.887	0.378	0.984	27.731	0.086
RE	0.052	0.91	125.250	0.884	0.012	0.969	38.621	0.121
RQ	0.039	0.95	130.402	0.890	0.549	0.984	27.413	0.085
YF	0.000	0.880	120.714	0.860	0.986	0.992	19.629	0.061

Table 13: CURVE FITTING ACCURACY FOR FIREFOX

Estimation Firefox	With clustering				Without clustering			
	p-value	*R-sq*	*RMSE*	*HH*	*p-value*	*R-sq*	*RMSE*	*HH*
Gamma	0.294	0.998	22.944	0.056	0.378	0.998	18.683	0.028
Weibull	0.294	0.998	23.704	0.058	0.307	0.998	18.826	0.029
AML	0.115	0.996	32.155	0.080	0.250	0.993	34.519	0.053
Normal	0.115	0.996	32.185	0.080	0.250	0.993	34.630	0.053
Power-law	0.053	0.990	31.791	0.078	0.307	0.998	20.032	0.031
RE	0.054	0.998	31.850	0.079	0.115	0.992	36.779	0.056
RQ	0.000	0.996	45.291	0.155	0.193	0.996	24.914	0.038
YF	0.113	0.990	32.001	0.080	0.150	0.996	24.543	0.037

Table 14: CURVE FITTING ACCURACY FOR CHROME

Estimation Chrome	With clustering				Without clustering			
	p-value	*R-sq*	*RMSE*	*HH*	*p-value*	*R-sq*	*RMSE*	*HH*
Gamma	0.137	0.998	57.890	0.123	0.368	0.998	20.094	0.036
Weibull	0.137	0.998	60.450	0.128	0.368	0.994	32.551	0.059
AML	0.385	0.997	62.917	0.134	0.690	0.995	29.213	0.053
Normal	0.385	0.997	62.913	0.134	0.690	0.995	29.415	0.053
Power-law	0.106	0.978	82.021	0.174	0.240	0.962	83.910	0.153
RE	0.000	0.998	115.135	0.291	0.000	0.937	107.974	0.199
RQ	0.000	0.996	81.852	0.170	0.000	0.973	70.106	0.127
YF	0.280	0.978	60.455	0.128	0.400	0.996	25.641	0.046

5.6.3 Summary of Curve-Fitting Results

Overall, in terms of curve-fitting, considering the OSs, the models that were using non-clustered data performed better in 24 cases out of 27 valid cases (the valid cases are those with at least one statistically sound MCF). Considering web browsers, again the non-clustering approach led to more accurate results in all the valid cases (29 cases). Considering the OSs and the web browsers together (8 cases), out of 56 valid cases (8 datasets * eight models per dataset – invalid models) we analyzed, in terms of estimation, the approach without-clustering led to more accurate results in 53 (94.6%) cases.

Comparing with-clustering results together, in terms of curve fitting, out of eight datasets, the Gamma-based VDMs was best in five (62.5%) cases. The Weibull, AML, and Normal VDMs each were equally best in two (25%) cases. The Power-law model was most accurate in one (12.5%) case.

Therefore, based upon our findings from estimation results, **comparing modeling strategies, the results from the non-clustered approach were more accurate than those from the clustered approach.**

However, when comparing clustering results together, **the Gamma-based VDM was the most accurate model given all the datasets we had**.

5.7 Prediction Results

In this section, we provide the results regarding prediction capabilities of the models (comparing the obtained predictions with clustered data and non-clustered data, and comparing the results generated from the clustering approach). For each case, the model that has the smallest value of AE and p-value≥0.05 was selected as having the

best prediction capability and is highlighted in green. Regarding p-values, we used * to show the models with $p<0.05$.

5.7.1 Operating systems

Tables 15-18 present the values of AE, AB, R^2 and HH for the four operating systems in our study. Eight models per software were analyzed. There are five cases where neither of the MCFs were statistically sound (five invalid cases). For Windows, in all the valid VDMs but RQ, the MCFs with clustering led to more accurate results compared to the MCFs without clustering. For Mac, in four cases out of five valid cases, the MCF with clustering led to more accurate results. For IOS, the MCFs without clustering led to more accurate results in all the valid cases. For Linux, for all the models except Gamma-based VDM, the MCFs with clustering resulted in the most accurate prediction results.

In addition, comparing with-clustering, for Windows and Mac, the AML VDM and the YF VDM have the smallest values of AB, AE and HH, respectively. For IOS, the Weibull-based VDM and the Power-law model have the smallest AE values. However, the HH values show that the Power-law model should be selected as the first best model. For Linux, Weibull-based VDM has the best results. Note that negative values of AB indicate that the model may underestimate the total number of discovered vulnerabilities.

Table 15: PREDICTION ACCURACY FOR WINDOWS

Prediction Windows	With clustering				Without clustering			
	AE	*AB*	*R-sq*	*HH*	*AE*	*AB*	*R-sq*	*HH*
Gamma	0.037	-0.037	0.96	0.060	0.063	-0.061	0.94	0.095
Weibull	0.035	-0.045	0.97	0.049	0.091	-0.091	0.94	0.131
AML	0.015	-0.015	0.99	0.018	0.138	-0.138	0.88	0.187
Normal	0.021	-0.021	0.99	0.035	0.138	-0.138	0.88	0.187
Power-law	0.035	-0.031	0.84	0.046	0.037	0.025	0.90	0.040
RE	0.021*	-0.019	0.86	0.035	0.106*	0.106	0.86	0.100
RQ	0.033*	-0.040	0.98	0.040	0.039	0.030	0.95	0.042
YF	0.036	-0.032	0.96	0.050	0.114	-0.114	0.94	0.158

Table 16: PREDICTION ACCURACY FOR MAC

Prediction Mac	With clustering				Without clustering			
	AE	*AB*	*R-sq*	*HH*	*AE*	*AB*	*R-sq*	*HH*
Gamma	0.080	-0.041	0.97	0.175	0.218	-0.218	0.99	0.263
Weibull	0.061	-0.027	0.98	0.265	0.233	-0.233	0.99	0.287
AML	0.032*	0.0194	0.99	0.234	0.278*	-0.278	0.97	0.351
Normal	0.153*	-0.130	0.98	0.309	0.278*	-0.278	0.97	0.351
Power-law	1.059	1.059	0.58	0.786	0.074	-0.074	0.94	0.080
RE	0.119	0.890	0.86	0.279	0.024*	0.017	0.97	0.037
RQ	0.270*	-0.202	0.89	0.361	0.082*	-0.082	0.86	0.090
YF	0.035	-0.035	0.98	0.236	0.256	-0.256	0.99	0.320

Table 17: PREDICTION ACCURACY FOR IOS

Prediction IOS	With clustering				Without clustering			
	AE	*AB*	*R-sq*	*HH*	*AE*	*AB*	*R-sq*	*HH*
Gamma	0.260	-0.181	0.95	0.367	0.018	0.000	0.97	0.025
Weibull	0.259	-0.179	0.95	0.379	0.019	0.005	0.98	0.027
AML	0.502	-0.501	0.87	0.894	0.076	0.076	0.98	0.088
Normal	0.598	-0.598	0.74	1.142	0.076	0.076	0.95	0.088
Power-law	0.259	-0.179	0.90	0.365	0.019	0.006	0.87	0.028
RE	0.492*	-0.492	0.87	0.853	0.131	0.131	0.86	0.148
RQ	0.492*	-0.318	0.84	1.142	0.154*	-0.154	0.86	0.172
YF	0.263	-0.155	0.95	0.373	0.092	0.092	0.99	0.108

Table 18: PREDICTION ACCURACY FOR LINUX

Prediction Linux	With clustering				Without clustering			
	AE	*AB*	*R-sq*	*HH*	*AE*	*AB*	*R-sq*	*HH*
Gamma	0.274	-0.274	0.95	0.398	0.268	-0.268	0.94	0.354
Weibull	0.032	0.013	0.99	0.032	0.267	-0.267	0.74	0.352
AML	0.142	0.091	0.98	0.132	0.272	-0.272	0.78	0.366
Normal	0.091	0.063	0.99	0.082	0.272	-0.272	0.78	0.366
Power-law	0.037	0.037	0.83	0.037	0.267	-0.267	0.70	0.352
RE	0.174	-0.174	0.85	0.398	0.190*	-0.190	0.95	0.239
RQ	0.274	-0.274	0.90	0.398	0.278	-0.278	0.74	0.369
YF	0.083	0.083	0.98	0.074	0.240*	-0.240	0.85	0.313

5.7.2 Web browsers

Tables 19-22 present the values of AE, AB, R^2 and HH for the four web browsers in our study. Again, eight models per software were analyzed. There are two cases where neither of the MCFs were statistically sound (two invalid cases). For IE, for all the VDMs but not for the Power-law model, the MCFs without clustering led to more accurate results compared to the MCFs without clustering. For Safari, in all the VDMs but the Power-law model and RE VDM, the MCFs without clustering led to most accurate results. For Firefox, for all the VDMs but not for the RQ VDM, the MCFs with clustering led to most accurate results. For Chrome, all the MCFs with clustering led to the more accurate results compared to the MCFs without clustering.

In addition, comparing with-clustering, for IE, Safari, and Firefox, the Power-law model had the smallest error values. For Chrome, the Gamma-based VDM had the smallest error values.

Table 19: PREDICTION ACCURACY FOR IE

Prediction IE	With clustering				Without clustering			
	AE	***AB***	***R-sq***	***HH***	***AE***	***AB***	***R-sq***	***HH***
Gamma	0.240	-0.230	0.99	0.302	0.234	-0.234	0.97	0.273
Weibull	0.280	-0.280	0.99	0.329	0.233	-0.233	0.98	0.272
AML	0.294*	-0.294	0.99	0.347	0.157	-0.157	0.99	0.175
Normal	0.295*	-0.295	0.99	0.347	0.157	-0.157	0.99	0.175
Power-law	0.108	0.074	0.89	0.184	0.233	-0.233	0.89	0.271
RE	0.276	-0.276	0.97	0.326	0.149	-0.149	0.92	0.164
RQ	0.308*	-0.308	0.97	0.372	0.232	-0.232	0.88	0.270
YF	0.240	-0.230	0.99	0.302	0.141	-0.141	0.97	0.155

Table 20: PREDICTION ACCURACY FOR SAFARI

Prediction Safari	With clustering				Without clustering			
	AE	***AB***	***R-sq***	***HH***	***AE***	***AB***	***R-sq***	***HH***
Gamma	0.609	-0.609	0.79	1.126	0.156	-0.156	0.94	0.201
Weibull	0.611	-0.611	0.79	1.134	0.187	-0.187	0.94	0.245
AML	0.656	-0.656	0.67	1.278	0.231	-0.231	0.88	0.304
Normal	0.656	-0.656	0.68	1.280	0.231	-0.231	0.88	0.304
Power-law	0.023	-0.018	0.90	0.026	0.030	0.026	0.90	0.037
RE	0.640	-0.640	0.70	1.248	0.133*	0.133	0.88	0.141
RQ	0.640	-0.566	0.85	1.251	0.041	0.040	0.99	0.047
YF	0.616*	-0.612	0.83	1.140	0.211	-0.211	0.86	0.278

Table 21: PREDICTION ACCURACY FOR FIREFOX

Prediction Firefox	With clustering				Without clustering			
	AE	*AB*	*R-sq*	*HH*	*AE*	*AB*	*R-sq*	*HH*
Gamma	0.017	-0.017	0.99	0.026	0.051	0.035	0.99	0.066
Weibull	0.020	-0.020	0.99	0.029	0.049	0.031	0.99	0.064
AML	0.024	-0.024	0.98	0.023	0.081	-0.081	0.99	0.112
Normal	0.034	-0.034	0.95	0.032	0.081	-0.081	0.98	0.112
Power-law	0.015	0.014	0.99	0.015	0.069	0.067	0.98	0.089
RE	0.034	-0.034	0.96	0.030	0.161	0.161	0.89	0.174
RQ	0.075*	-0.054	0.86	0.032	0.096	0.096	0.99	0.113
YF	0.021	-0.021	0.99	0.022	0.051	-0.032	0.99	0.069

Table 22: PREDICTION ACCURACY FOR CHROME

Prediction Chrome	With clustering				Without clustering			
	AE	*AB*	*R-sq*	*HH*	*AE*	*AB*	*R-sq*	*HH*
Gamma	0.173	-0.173	0.99	0.167	0.281	-0.281	0.99	0.367
Weibull	0.225	-0.225	0.98	0.215	0.317	-0.317	0.99	0.422
AML	0.232	-0.232	0.99	0.222	0.307	-0.307	0.99	0.405
Normal	0.232	-0.232	0.99	0.222	0.307	-0.307	0.99	0.405
Power-law	0.324	0.324	0.94	0.345	0.167	0.167	0.94	0.191
RE	0.298*	-0.298	0.86	0.305	0.364*	0.364	0.89	0.383
RQ	0.334*	0.324	0.94	0.345	0.077*	0.077	0.99	0.102
YF	0.225	-0.225	0.98	0.213	0.304	-0.304	0.95	0.402

5.7.3 Summary of Prediction Results

In terms of prediction accuracy, considering the OSs, out of 27 valid cases (4 OS and 8 models excluding invalid cases), the MCFs that used clustered data led to more accurate results in 17 (63%) cases than those which used non-clustered data. Considering web browsers, the MCFs with-clustering has most accurate results in 16 (53.3%) cases, out of the 30 valid cases we analyzed. However, considering the OSs and web browsers together, out of 57 valid cases analyzed (8 software and 8 models excluding 7 invalid cases), in terms of prediction accuracy, the MCFs with clustering approach led to more accurate results in 33 (58%) cases.

Overall, in terms of prediction accuracy, out of 57 valid cases (eight datasets multiplied by eight models per dataset minus seven invalid cases), the MCF with

clustering led to more accurate results in 33 (58%) cases, while the MCF without clustering resulted in more accurate results in 24 (42%) cases.

Comparing with-clustering results together, in terms of prediction, out of eight datasets, the Power-law model and the Weibull-based VDM was the most accurate in four (50%) and two (25%) cases, respectively. The Gamma-based, AML and YF VDMs each were equally best in one (12.5%) cases. The other VDMs was best in neither of the cases.

Therefore, based upon our findings from prediction results, **comparing modeling strategies, the results with clustering were more accurate for eight datasets we analyzed.**

Comparing with-clustering results together, in terms of prediction accuracy, the **Power-law model was the most accurate model**.

Please remember that all the conclusions we draw from this research and their validity are limited by our database uncertainties.

5.8 Discussion

In this chapter, we explored the applicability of clustering to vulnerability data, and investigated whether this approach can lead to more accurate predictions with common SRMs/VDMs. Our results show that the cluster-based approach provided better prediction results than without clustering for our datasets. We have summarized our findings in the guidelines presented in Table 23:

Table 23: MODELING GUIDELINE

Approach	Without Clustering		With- Clustering	
	fitting	***prediction***	***fitting***	***prediction***
Model	Gamma	Power-law	Gamma	Power-law

5.9 Limitations

The main limitations of this chapter are the following:

- One limitation is with regard to the uncertainty of the databases we used. Vulnerability databases usually might have some uncertainty regarding variables associated with the reported vulnerabilities such as published dates.
- We have only applied the approach to 8 software. We don't know yet how well this works for other vulnerability datasets associated with a software, though we plan to extend this work in the future. Since we integrated all the vulnerabilities associated with multiple versions of a software and it might have triggered some sources of dependency, we plan to rebuild the datasets we used. In other words, for a given software, we will integrate the versions in which their source codes have reached a threshold of similarity. We will define a similarity metric.
- We have applied the approach to all the vulnerabilities of a software, rather than subdivided by version type. This was mainly because the sample size of vulnerabilities get much smaller when considering individual versions. Other researchers have looked at individual versions separately [24]. However, we

believe that different versions of a software cannot be assumed completely independent since different versions have large overlaps in their code base.

5.10 Summary

In this chapter, we used a common clustering technique to group the vulnerabilities into distinct clusters, using the textual information reported in these vulnerabilities. We have applied our approach on the vulnerabilities datasets introduced in Chapter 3. We also investigated whether this approach could result in more accurate results, in terms of estimation/prediction, compared to the case where all the vulnerabilities estimated together (results from Chapter 3). In next chapter, we will investigate different vulnerability grouping approaches.

Chapter 6: A Comparison of Vulnerabilities' Grouping Strategies

6.1 Introduction

In this chapter, we present some guidelines to model vulnerability discovery data based on two commonly employed vulnerabilities' grouping strategies. In the first strategy, for each software, we analyze all vulnerabilities reported for any of its versions. In the second strategy, for each software, we select only versions for which there are most vulnerabilities shared by two subsequent versions of the product. We used the eight models introduced in Chapter 3 (eight common VDMs) for the discovery process of vulnerabilities in the eight well-known software (four operating systems and four web browsers) also introduced in Chapter 3. The accuracy of these models was investigated based on their fitting and prediction capabilities.

6.2 Motivation

Many studies choose all vulnerabilities when assessing product families. This may be difficult to justify for products with long lifespans: some of the older vulnerabilities have long been fixed, and the code-based of these products is likely to have evolved significantly.

Other studies [24], [32] asses specific versions of products and only consider vulnerabilities that have been reported for those specific versions. This may be too restrictive as a subsequent version of the same product family is likely to share a large

proportion of the code base with its predecessor. Hence it is also likely to share a large proportion of the vulnerabilities.

Our research question investigates whether we can improve the results by filtering a given vulnerability dataset associated with a product with only versions for which there are most vulnerabilities shared by two subsequent versions of the product. In other words, the question is "should we only consider the vulnerabilities associated with a single version of a software for our modeling or all vulnerabilities reported for any of its versions?". To answer these questions, we need to consider each scenario separately and compare their results.

In this chapter, we make the following contributions:

- We use two strategies for grouping vulnerabilities (vulnerabilities merged for all versions and groups built based on a number of common vulnerabilities across versions).
- We apply eight common VDMs on the mentioned groups and compare their curve-fitting and prediction capabilities to derive a guideline.

6.3 Grouping Strategy

As mentioned is Chapter 3, we will analyze the reported vulnerabilities associated with four well-known OSs: Windows (1995-2017), Mac (1997-2017), IOS (the OS associated with Cisco) (1992-2017), and Linux (1994-2017), as well as four well-known web browsers including Internet Explorer (1997-2017), Safari (2003-2017), Firefox (2003-2017), and Chrome (2008-2017). These software have been selected because they are the most widely used and have the most vulnerabilities among the databases.

For each software, we considered two grouping strategies. In the first strategy (St. 1), for each software, we analyze all vulnerabilities reported for any of its versions. Thus, for each software, all the vulnerabilities reported for any of its versions were included. For instance, all the vulnerabilities reported for mac_os, mac_os_server, mac_os_x, and mac_os_x_server were put together to create a vulnerability database for Mac.

In the second strategy (St. 2), for each software, we group versions based on the percentage of common vulnerabilities (i.e., we group the consecutive versions with more than 70% common reported vulnerabilities). We assume the percentage of common vulnerabilities for two or more consecutive versions of a software be a good measure regarding the similarity of their source code. We selected 70% as a threshold for versions of high level of source code similarity that should be analyzed together. Table 24 shows the percentage of common vulnerability within some versions of Firefox. The threshold is met for versions 0.x, 1.x, 2.x and their associated vulnerabilities can be grouped together. Even though the threshold is also met for versions 0.x and 3.x, we do not consider them as a separate group since they are not consecutive versions. For the software versions that don't satisfy this condition we only consider the product with the most vulnerabilities reported (e.g., Linux_Kernel).

Table 24: PERCENTAGE OF COMMON VULNERABILITIES WITHIN FIREFOX VERSIONS

Versions (% of common vuls.)	Firefox 0.x	Firefox 1.x	Firefox 2.x	Firefox 3.x
Firefox 0.x	1	0.94	0.79	0.71
Firefox 1.x	*	1	0.70	0.52
Firefox 2.x	*	*	1	0.63
Firefox 3.x	*	*	*	1

Table 25 presents the total number of vulnerabilities for each software (All versions together (St. 1) and versions grouped based on the similarity threshold (St. 2)) as well as their skewness values. All the datasets associated with the eight software we analyzed are right skewed (each has a positive skewness value with an absolute value of skewness greater than 0.5). Safari was not considered in St. 2 since we didn't have enough vulnerabilities for modeling by grouping the versions with more than 70% common vulnerabilities.

Table 25: NUMBER OF VULNERABILITIES PER SOFTWARE

OS (St.1)	**Windows**	**Mac**	**IOS**	**Linux**
# Vulnerabilities	3100	2705	650	4745
Skewness	2.10	2.19	3.12	3.50
Web Browser (St.1)	**IE**	**Safari**	**Firefox**	**Chrome**
# Vulnerabilities	1775	943	1477	1837
Skewness	2.65	2.98	10.00	1.63
OS (St.2)	**Windows Vista & 7**	**Mac OS_X**	**IOS 11.x & 12.x**	**Linux Kernel**
# Vulnerabilities	1054	1907	317	1993
Skewness	1.52	0.63	0.51	2.79
Web Browser (St.2)	**IE (5.x, 6.x)**	**IE (9.x, 10.x)**	**IE (10.x, 11.x)**	**Firefox (0.x, 1.x, 2.x)**
# Vulnerabilities	667	720	677	882
Skewness	1.07	2.34	2.50	8.05
Web Browser (St.2)	**Chrome (0.x, 1.x, 2.x)**	**Chrome (3.x, 4.x)**		
# Vulnerabilities	878	959		
Skewness	0.86	1.35		

6.4 Analysis

Like what we described in Chapter 4, for the vulnerability data in each group, we divided the time axis into 30-day intervals (t=0 is associated with the vulnerability with the earliest published date), and counted the cumulative frequency of

vulnerabilities detected in each interval. The regression and the analysis methods we used described in Section 4.3.

6.4.1 Curve-fitting error indicators

We used the eight models for the discovery process of vulnerabilities introduced in Chapter 3 in eight well-known software (four operating systems and four web browsers). The models were fitted to the 18 datasets (8 datasets from St. 1 and 10 datasets from St. 2) using a non-linear regression method described in [24]. Most of the error indicators considered in this chapter are the same than in previous chapters (χ^2, HH). However, we added a few more indicators for better judgment between the models.

The Akaike Information Criteria (AIC) is frequently used to make a fair comparison between models. It also accounts for overfitting detection by penalizing the models with an issue of overfitting whereas metrics like χ^2 don't have such capability. AIC is formally defined as:

$$AIC = (-2 \times log\ likelihood) + 2\mathrm{M} \tag{22}$$

where M is the number of free parameters of the examined model. Alhazmi & Malaiya in [18], [21] reported the AIC values. To compare two models, we consider their difference:

$$\Delta\mathrm{i} = \mathrm{AIC}_i - \mathrm{AIC}_{min} \tag{23}$$

where AIC_i is the AIC of the i-th model, and AIC_{min}n is the lowest AIC one obtains among the set of models examined (i.e., the preferred model). The rule of thumb,

outlined in [66], is: if $\Delta i<2$, then there is substantial support for the i-th model (or the evidence against it is worth only a bare mention), and the proposition that it is a proper description is highly probable; if $2<\Delta i<4$, then there is strong support for the i-th model; if $4<\Delta i<7$, then there is considerably less support for the i-th model; models with $\Delta i>10$ have essentially no support. In this chapter, models with $\Delta i<4$ were selected as the best models.

6.4.2 Prediction error indicators

In addition to AE and AB, we also report $\%\Delta AE_i$, which represents the percentage of difference between the AE of the i-the model and the model with minimum AE

$$\%\Delta AE_i = (AE_i - AE_{min}) * 100 \quad (24)$$

where AE_i is the AE of the i-th model, and AE_{min} is the lowest AE obtained among the set of models examined (i.e., the best model).

6.5 Curve-Fitting Results

6.5.1 Operating systems

Tables 26-27 contain the χ^2 goodness of fit test p-values, HH, the AIC values, and Δi for the OSs (St. 1 and St. 2) and the web browsers (St. 1 and St. 2), respectively. In each case, the model that has the smallest values of HH and AIC is selected as the best fitting model and highlighted in green. The other models with $\Delta i<4$ are also selected as the best models.

Table 26: CURVE FITTING ACCURACY FOR OSS (ST.1)

	Windows				Mac			
	p-val	***HH***	***AIC***	***Δi***	***p-val***	***HH***	***AIC***	***Δi***
Gamma	1.00	0.03	2914.58	0.00	1.00	0.07	6080.39	5.78
Weibull	1.00	0.03	2916.34	1.75	1.00	0.07	6079.86	5.26
AML	1.00	0.05	3083.02	168.43	0.99	0.08	6252.64	178.04
Normal	1.00	0.05	3082.02	167.43	0.99	0.08	6251.64	177.04
Power-law	1.00	0.05	3075.58	161.00	1.00	0.07	6077.50	2.90
RE	1.00	0.08	3381.79	467.21	0.99	0.08	6165.09	90.49
RQ	1.00	0.05	3078.25	163.66	1.00	0.07	6074.60	0.00
YF	1.00	0.04	2967.94	53.36	0.98	0.08	6166.42	91.81
	IOS				**Linux**			
	p-val	***HH***	***AIC***	***Δi***	***p-val***	***HH***	***AIC***	***Δi***
Gamma	0.25	0.03	2220.06	102.65	0.46	0.119	20506.23	1036.55
Weibull	0.43	0.03	2216.41	99.00	0.78	0.118	20493.64	1023.95
AML	1.00	0.03	2118.41	1.00	0.47	0.108	20240.71	771.02
Normal	1.00	0.03	2117.41	0.00	0.98	0.083	19469.69	0.00
Power-law	0.48	0.03	2213.77	96.36	0.99	0.118	20490.83	1021.14
RE	0.00	0.04	2288.65	171.24	0.99	0.088	19652.27	182.59
RQ	0.00	0.07	2633.48	516.06	0.00	0.134	20856.60	1386.91
YF	1.00	0.03	2152.12	34.71	0.99	0.088	19652.72	183.04

Table 27: CURVE FITTING ACCURACY FOR OSS (ST.2)

	Windows (Vista & 7)				Mac_OS_X			
	p-val	***HH***	***AIC***	***Δi***	***p-val***	***HH***	***AIC***	***Δi***
Gamma	0.99	0.07	1830.80	39.84	0.23	0.12	3236.55	77.19
Weibull	0.99	0.07	1829.74	38.78	0.45	0.12	3235.16	75.81
AML	0.00	0.10	1932.50	141.53	0.90	0.11	3188.84	29.49
Normal	0.00	0.10	1931.50	140.53	0.98	0.11	3178.92	19.57
Power-law	0.98	0.07	1827.65	36.69	0.16	0.12	3233.08	73.72
RE	0.99	0.07	1790.96	0.00	1.00	0.10	3159.35	0.00
RQ	0.98	0.07	1810.02	19.06	0.99	0.12	3238.09	78.74
YF	1.00	0.07	1794.40	3.44	0.99	0.10	3164.66	5.31
	IOS (11.x & 12.x)				**Linux_Kernel**			
	p-val	***HH***	***AIC***	***Δi***	***p-val***	***HH***	***AIC***	***Δi***
Gamma	0.27	0.05	852.90	99.33	0.99	0.05	7325.63	73.01
Weibull	0.72	0.04	827.59	74.01	0.99	0.05	7320.49	67.88
AML	1.00	0.03	754.58	1.00	0.98	0.06	7389.79	137.17
Normal	1.00	0.03	753.58	0.00	0.98	0.06	7388.79	136.17
Power-law	1.00	0.06	923.13	169.55	0.76	0.05	7317.41	64.79
RE	0.00	0.09	997.64	244.06	1.00	0.05	7252.62	0.00
RQ	0.14	0.07	953.12	199.54	0.00	0.06	7568.41	315.79
YF	1.00	0.03	762.78	9.21	1.00	0.05	7275.13	22.52

For OSs (St. 1 and St. 2), in all cases, most of the models are statistically sound with p-values greater than 0.05. RE and RQ VDMs each resulted in unsound p-values in two cases. However, we cannot accept all the models with sound p-values and have to consider their associated Δi values. Comparing grouping strategies, in terms of curve fitting, the AML and Normal-based VDMs were the best models for IOS in both strategies. However, for other OSs, the results associated with St. 1 and St. 2 differ.

6.5.2 Web browsers

Tables 28-29 contain the χ^2 goodness of fit test p-values, HH, the AIC values, and Δi for the web browsers (St. 1 and St. 2) and the web browsers (St. 1 and St. 2), respectively.

For web browsers (St. 1), most of the models still have statistically sound p-values. However, this is not true for web browsers (St. 2). Comparing the grouping strategies, there is no common best model within Chrome (St. 1) and Chrome (3.x, 4.x). Comparing grouping strategies, in terms of curve fitting, for Chrome and one of its subversions (Chrome 3.x &4.x), the Gamma-based VDM was the best common model using both strategies.

Table 28: CURVE FITTING ACCURACY FOR WEB BROWSERS (ST.1)

	IE				Safari			
	p-val	***HH***	***AIC***	***Δi***	***p-val***	***HH***	***AIC***	***Δi***
Gamma	1.00	0.12	4906.01	293.35	1.00	0.06	1598.98	0.00
Weibull	1.00	0.12	4903.48	290.82	1.00	0.07	1610.80	11.82
AML	1.00	0.09	4666.23	53.57	0.99	0.09	1716.48	117.49
Normal	1.00	0.08	4612.66	0.00	0.99	0.09	1715.48	116.49
Power-law	0.99	0.12	4901.34	288.68	1.00	0.07	1629.14	30.16
RE	1.00	0.09	4639.14	26.48	0.99	0.10	1750.82	151.84
RQ	1.00	0.13	4952.64	339.98	1.00	0.07	1632.85	33.87
YF	0.99	0.09	4619.17	6.51	1.00	0.08	1667.18	68.20
	Firefox				**Chrome**			
	p-val	***HH***	***AIC***	***Δi***	***p-val***	***HH***	***AIC***	***Δi***
Gamma	1.00	0.03	2332.83	3.07	0.47	0.07	2773.10	0.00
Weibull	1.00	0.03	2332.35	2.60	0.43	0.08	2822.16	49.05
AML	0.00	0.05	2606.51	276.76	0.23	0.09	2892.74	119.64
Normal	0.00	0.05	2605.51	275.76	0.20	0.09	2891.74	118.64
Power-law	1.00	0.03	2329.75	0.00	0.24	0.10	2905.96	132.86
RE	0.05	0.04	2496.37	166.61	0.00	0.12	3025.95	252.85
RQ	0.46	0.03	2370.79	41.04	0.34	0.09	2848.45	75.35
YF	1.00	0.03	2334.13	4.38	0.35	0.08	2843.79	70.69

Table 29: CURVE FITTING ACCURACY FOR WEB BROWSERS (ST.2)

	IE (5.x &6.x)				IE (9.x &10.x)			
	p-val	*HH*	*AIC*	*Δi*	*p-val*	*HH*	*AIC*	*Δi*
Gamma	0.42	0.04	2403.89	0.00	0.00	0.03	714.25	81.52
Weibull	0.82	0.04	2405.72	1.83	0.00	0.03	664.79	32.06
AML	0.00	0.06	2581.77	177.88	0.28	0.02	633.73	1.00
Normal	0.00	0.06	2580.77	176.88	0.26	0.02	632.73	0.00
Power-law	0.00	0.06	2648.43	244.54	0.00	0.11	913.15	280.42
RE	0.00	0.08	2752.95	349.06	0.00	0.13	946.00	313.27
RQ	0.00	0.07	2734.52	330.63	0.00	0.12	933.75	301.02
YF	0.00	0.05	2491.38	87.49	0.00	0.03	714.25	81.52
	IE (10.x &11.x)				**Firefox (0.x , 1.x, 2.x)**			
	p-val	*HH*	*AIC*	*Δi*	*p-val*	*HH*	*AIC*	*Δi*
Gamma	0.95	0.02	543.12	0.00	0.00	0.08	1710.73	7.76
Weibull	0.65	0.03	551.15	8.04	0.00	0.08	1710.73	7.76
AML	0.85	0.04	598.93	55.81	0.00	0.11	1802.55	99.58
Normal	0.63	0.04	597.93	54.81	0.00	0.11	1801.55	98.58
Power-law	0.00	0.12	769.67	226.56	0.00	0.08	1707.73	4.76
RE	0.00	0.14	796.28	253.16	0.00	0.08	1707.57	4.60
RQ	0.00	0.13	786.27	243.16	0.05	0.08	1706.58	3.60
YF	0.76	0.03	574.28	31.17	0.11	0.08	1702.97	0.00
	Chrome (0.x , 1.x, 2.x)				**Chrome (3.x , 4.x)**			
	p-val	*HH*	*AIC*	*Δi*	*p-val*	*HH*	*AIC*	*Δi*
Gamma	0.00	0.04	1026.71	146.40	0.07	0.09	1273.70	0.00
Weibull	0.32	0.02	908.83	28.52	0.05	0.09	1275.41	1.71
AML	0.68	0.03	936.37	56.07	0.00	0.13	1357.99	84.29
Normal	0.68	0.03	935.37	55.07	0.00	0.13	1356.99	83.29
Power-law	0.00	0.22	1444.48	564.17	0.00	0.10	1292.40	18.70
RE	0.00	0.24	1464.31	584.00	0.00	0.11	1322.00	48.30
RQ	0.00	0.24	1462.09	581.78	0.04	0.10	1279.97	6.27
YF	1.00	0.02	880.31	0.00	0.04	0.10	1282.88	9.18

6.5.3 Summary of curve-fitting results

Overall, in terms of curve fitting, considering the OSs (St. 1), the Normal VDM was selected in two cases out of four. The AML, Weibull, and Gamma-based VDM, the Power-law, and RQ models were equally best model in one case out of four, while the RE model was the best model in three cases out of four. When considering the OSs (St. 2), the AML and Normal-based VDMs, and the YF model each are being selected as the best model in one case out of four (St. 2). However, for other web browsers, the results associated with St. 1 and St. 2 differ.

Considering the web browsers (St. 1), the Gamma-based VDM provided the best fit in three out of four cases, whereas the Weibull and Normal-based VDMs along

with the Power-law model were each the best model in one of the cases. However, considering the web browsers (St. 2), the Gamma-based VDM was selected as the best model in three out of six cases. The Weibull-based VDM was found the best model in two cases out of six.

6.6 Prediction Results

6.6.1 Operating systems

Tables 30-31 present the values of AE, AB, and $\%\Delta AE_i$ for the cases we analyzed per strategy, respectively. AB can be positive (for overestimation) or negative (for underestimation), while AE is always positive. We used * to show the models with $p<0.05$ and didn't calculate $\%\Delta AE_i$ for those models (note: we used "NA" as the $\%\Delta AE_i$ value of the models with $p<0.05$). In each case, the model that has the smallest value of AE and $p>0.05$ was selected as having the best prediction capability and is highlighted in green. In addition, the model/models with $\%\Delta AE_i < 2$ were also selected as the best prediction models, which, show similar prediction capability compared to the best model (the model/models with $\%\Delta AE_i = 0$).

Comparing the grouping strategies in terms of prediction capabilities, only for Mac (St. 1) and Mac_OS_X (St. 2) and Linux (St. 1) and Linux_Kernel (St. 2), was there one common best model. In other words, the Gamma-based VDM and the Power-law model were the best models for IOS, Linux in both strategies, respectively. However, for other OSs, the results associated with St. 1 and St. 2 differ.

Table 30: PREDICTION ACCURACY FOR WEB BROWSERS (ST.1)

	Windows				Mac			
	AE	***AB***	***HH***	***%ΔAEi***	***AE***	***AB***	***HH***	***%ΔAEi***
Gamma	0.13	0.12	0.167	7.59	0.15	-0.14	0.280	0.00
Weibull	0.13	0.12	0.169	7.72	0.25	-0.25	0.435	10.40
AML	0.05	-0.05	0.103	0.00	0.27	-0.27	0.459	12.57
Normal	0.05	-0.05	0.103	0.00	0.27	-0.27	0.459	12.57
Power-law	0.13	0.12	0.171	7.85	0.36	0.36	0.322	21.40
RE	0.45	0.45	0.531	39.98	0.87	0.87	0.737	72.64
RQ	0.07	0.05	0.094	2.07	0.20	0.20	0.181	5.29
YF	0.08	0.08	0.100	3.01	0.24	-0.24	0.420	9.64
	IOS				**Linux**			
	AE	***AB***	***HH***	***%ΔAEi***	***AE***	***AB***	***HH***	***%ΔAEi***
Gamma	0.17	-0.17	0.228	4.13	0.31	-0.31	0.594	18.33
Weibull	0.17	-0.17	0.225	3.91	0.38	-0.38	0.733	25.17
AML	0.25	-0.25	0.390	12.69	0.41	-0.41	0.785	28.00
Normal	0.25	-0.25	0.390	12.69	0.41	-0.41	0.785	28.00
Power-law	0.16	-0.16	0.224	3.87	0.14	-0.08	0.245	1.77
RE	0.20*	0.20	0.226	NA	0.13	0.09	0.107	0.00
RQ	0.27*	-0.27	0.388	NA	0.14*	-0.04	0.206	NA
YF	0.13	-0.12	0.179	0.00	0.40	-0.40	0.771	27.10

Table 31: PREDICTION ACCURACY FOR OSS (ST.2)

	Windows (Vista & 7)				Mac_OS_X			
	AE	***AB***	***HH***	***%ΔAEi***	***AE***	***AB***	***HH***	***%ΔAEi***
Gamma	0.14	-0.12	0.252	5.83	0.24	-0.24	0.443	0.40
Weibull	0.17	-0.16	0.313	8.86	0.31	-0.31	0.549	7.56
AML	0.24	-0.24	0.421	15.91	0.32	-0.32	0.559	8.54
Normal	0.24	-0.24	0.421	15.91	0.32	-0.32	0.559	8.54
Power-law	0.08	0.05	0.080	0.00	0.23	0.23	0.197	0.00
RE	0.15	0.15	0.131	6.50	0.55	0.55	0.440	31.42
RQ	0.11	0.10	0.097	2.50	0.23	0.23	0.198	0.04
YF	0.23	-0.23	0.404	14.37	0.31	-0.31	0.552	7.82
	IOS (11.x & 12.x)				**Linux_Kernel**			
	AE	***AB***	***HH***	***%ΔAEi***	***AE***	***AB***	***HH***	***%ΔAEi***
Gamma	0.08	0.04	0.099	0.41	0.31	-0.31	0.552	26.96
Weibull	0.08	0.04	0.099	0.45	0.38	-0.38	0.705	33.67
AML	0.07	-0.07	0.084	0.00	0.40	-0.40	0.730	35.28
Normal	0.07	-0.07	0.084	0.00	0.40	-0.40	0.730	35.28
Power-law	0.08	0.04	0.100	0.46	0.04	0.02	0.053	0.00
RE	0.29*	0.29	0.359	NA	0.33	0.33	0.369	28.72
RQ	0.10	0.07	0.129	2.38	0.06*	-0.06	0.118	NA
YF	0.13	0.12	0.172	5.91	0.37	-0.37	0.685	32.90

6.6.2 Web browsers

Tables 32-33 present the values of AE, AB, and $\%\Delta AE_i$ for the four web browsers (St. 1 and St. 2).

Table 32: PREDICTION ACCURACY FOR WEB BROWSERS (ST.1)

	IE				Safari			
	AE	***AB***	***HH***	***%ΔAEi***	***AE***	***AB***	***HH***	***%ΔAEi***
Gamma	0.20	-0.16	0.364	7.90	0.16	0.16	0.140	5.42
Weibull	0.22	-0.19	0.423	10.17	0.10	0.01	0.131	0.00
AML	0.33	-0.33	0.673	21.31	0.15	-0.14	0.271	4.94
Normal	0.33	-0.33	0.673	21.31	0.15	-0.14	0.271	4.94
Power-law	0.17	-0.09	0.266	4.92	0.43	0.43	0.388	32.61
RE	0.12	0.06	0.104	0.00	1.02	1.02	0.870	91.95
RQ	0.15	-0.05	0.221	3.78	0.29	0.29	0.266	18.74
YF	0.29	-0.29	0.610	17.77	0.12	-0.05	0.187	1.34
	Firefox				**Chrome**			
	AE	***AB***	***HH***	***%ΔAEi***	***AE***	***AB***	***HH***	***%ΔAEi***
Gamma	0.05	-0.05	0.049	2.21	0.21	-0.21	0.360	0.00
Weibull	0.05	-0.04	0.049	2.17	0.31	-0.31	0.508	10.13
AML	0.18*	-0.18	0.236	NA	0.27	-0.27	0.446	6.22
Normal	0.18*	-0.18	0.236	NA	0.27	-0.27	0.446	6.22
Power-law	0.05	-0.04	0.048	2.13	1.00	1.00	0.833	79.27
RE	0.06	0.06	0.086	3.94	2.14*	2.14	1.537	NA
RQ	0.02	0.00	0.031	0.00	0.40	0.40	0.374	19.41
YF	0.07	-0.07	0.079	4.13	0.26	-0.26	0.435	5.12

Table 33: PREDICTION ACCURACY FOR WEB BROWSERS (ST.2)

	IE (5.x &6.x)				IE (9.x &10.x)			
	AE	***AB***	***HH***	***%ΔAEi***	***AE***	***AB***	***HH***	***%ΔAEi***
Gamma	0.06	0.06	0.065	2.07	0.23*	0.23	0.268	NA
Weibull	0.04	0.03	0.043	0.00	0.17*	0.17	0.192	NA
AML	0.10*	-0.10	0.138	NA	0.04	-0.04	0.054	1.80
Normal	0.10*	-0.10	0.138	NA	0.04	-0.04	0.054	1.80
Power-law	0.17*	0.17	0.169	NA	0.33*	0.33	0.365	NA
RE	0.27*	0.27	0.260	NA	0.68*	0.68	0.680	NA
RQ	0.23*	0.23	0.218	NA	0.33*	0.33	0.363	NA
YF	0.07*	-0.07	0.110	NA	0.03	0.03	0.029	0.00
	IE (10.x &11.x)				**Firefox (0.x , 1.x, 2.x)**			
	AE	***AB***	***HH***	***%ΔAEi***	***AE***	***AB***	***HH***	***%ΔAEi***
Gamma	0.06	0.06	0.067	0.85	0.14*	-0.14	0.157	NA
Weibull	0.05	-0.05	0.066	0.00	0.14*	-0.14	0.157	NA
AML	0.12	-0.12	0.149	6.88	0.24*	-0.24	0.280	NA
Normal	0.12	-0.12	0.149	6.88	0.24*	-0.24	0.280	NA
Power-law	0.43*	0.43	0.442	NA	0.14*	-0.13	0.148	NA
RE	0.86*	0.86	0.811	NA	0.11*	-0.10	0.119	NA
RQ	0.41*	0.41	0.424	NA	0.14	-0.14	0.160	2.21
YF	0.09	-0.09	0.114	3.79	0.12	-0.11	0.134	0.00
	Chrome (0.x , 1.x, 2.x)				**Chrome (3.x , 4.x)**			
	AE	***AB***	***HH***	***%ΔAEi***	***AE***	***AB***	***HH***	***%ΔAEi***
Gamma	0.12*	0.12	0.175	NA	0.25	-0.25	0.305	0.00
Weibull	0.04	0.04	0.063	2.10	0.27	-0.27	0.329	2.51
AML	0.02	-0.02	0.026	0.17	0.27*	-0.27	0.334	NA
Normal	0.02	-0.02	0.026	0.17	0.27*	-0.27	0.334	NA
Power-law	0.52*	0.52	0.779	NA	0.04*	-0.02	0.052	NA
RE	1.19*	1.19	1.778	NA	0.04*	0.04	0.074	NA
RQ	0.55*	0.55	0.816	NA	0.12*	-0.12	0.135	NA
YF	0.02	0.01	0.028	0.00	0.27*	-0.27	0.338	NA

6.6.3 Summary of prediction results

Overall, in terms of prediction, considering the OSs (St. 1), all the models except the Weibull-based VDM were selected as the best model in one case out of four. Considering the OSs (St. 2), the Power-law model was the best one in all four cases. The Gamma-based VDM was the most accurate in two cases; the other models except the RE model each were selected as the best model in one case.

Considering the web browsers (St. 1), the Gamma and Weibull-based VDMs and the RE, YF, and RQ models were each better than other models in one case out of four. However, considering the web browsers (St. 2), the YF model was most accurate in three cases out of six. Gamma, Weibull, AML, and Normal-based VDMs are the best models in two cases.

Please remember that all the conclusions we draw from this research and their validity are limited by our database uncertainties.

6.7 Discussion

In this section, we take a closer look at our findings and then offer some guidelines to model vulnerability discovery data with respect to the shape of the discovery intensity functions.

Tables 34-35 show the summary of the selected models per dataset for curve-fitting and prediction, respectively. For curve-fitting, the models p-value<0.05 are highlighted in red. Based upon the results, in terms of curve fitting, out of eight datasets grouped by Strategy #1 (St. 1), the Gamma-based, Normal-based, Weibull-based, Power-law, AML and RQ VDMs provided the best (or equally best) fit in four, three,

two, two, one, one, cases, respectively. The other models were most accurate in neither of the cases.

Table 34: SUMMARY OF SELECTED MODELS PER DATASET (CURVE-FITTING)

Curve-Fitting	OS								Web Browsers									
	Windows		Mac		IOS		Linux		IE				Safari	Firefox		Chrome		
	St.1	St.2	St.1	St.2	St.1	St.2	St.1	St.2	St.1	5&6	9&10	10&11	St.1	St.1	0&1&2	St.1	0&1&2	3&4
Gamma	✓	×	×	×	×	×	×	×	×	✓	×	✓	✓	✓	×	✓	×	✓
Weibull	✓	×	×	×	×	×	×	×	×	✓	×	×	×	✓	×	×	×	✓
AML	×	×	×	×	✓	✓	×	×	×	×	×	×	×	×	×	×	×	×
Normal	×	×	×	×	✓	✓	✓	×	✓	×	×	×	×	×	×	×	×	×
Power-law	×	×	✓	×	×	×	×	×	×	×	×	×	×	✓	×	×	×	×
RE	×	✓	×	✓	×	×	×	✓	×	×	×	×	×	×	×	×	×	×
RQ	×	×	✓	×	×	×	×	×	×	×	×	×	×	×	✓	×	×	×
YF	×	✓	×	×	×	×	×	×	×	×	✓	×	×	×	✓	×	✓	×

Table 35: SUMMARY OF SELECTED MODELS PER DATAS ET (PREDICTION)

Prediction	OS								Web Browsers									
	Windows		Mac		IOS		Linux		IE				Safari	Firefox		Chrome		
	St.1	St.2	St.1	St.2	St.1	St.2	St.1	St.2	St.1	5&6	9&10	10&11	St.1	St.1	0&1&2	St.1	0&1&2	3&4
Gamma	×	×	✓	✓	×	✓	×	×	×	×	×	✓	×	×	×	✓	×	✓
Weibull	×	×	×	×	×	✓	×	×	×	✓	×	✓	✓	×	×	×	×	×
AML	✓	×	×	×	×	✓	×	×	×	×	✓	×	×	×	×	×	✓	×
Normal	✓	×	×	×	×	✓	×	×	×	×	✓	×	×	×	×	×	✓	×
Power-law	×	✓	×	✓	×	✓	✓	✓	×	×	×	×	×	×	×	×	×	×
RE	×	×	×	×	×	×	✓	×	✓	×	×	×	×	×	×	×	×	×
RQ	×	×	×	✓	×	×	×	×	×	×	×	×	×	✓	×	×	×	×
YF	×	×	×	×	✓	×	×	×	×	×	✓	×	✓	×	✓	×	✓	×

In terms of curve fitting, out of ten datasets in St.2, the YF, Gamma-based, and RE VDMs were most accurate in four, three, and three cases, respectively. The AML,

Normal-based, and RQ VDMs each were most accurate in one case. The other models were most accurate in neither of the cases.

In terms of prediction accuracy, out of eight datasets in St.1, the Gamma-based, RE, and YF VDMs, each provided the best (or equally best) fit in two cases. The other models each were most accurate in one case.

In terms of prediction accuracy, out of ten datasets in St.2, the Gamma-based, Power-law, Weibull-based, AML, Normal-based, and YF VDMs were most accurate in four, four, three, three, and three cases, respectively. The RQ model was most accurate in one case.

Based upon our findings we found similar guideline per grouping strategy, in terms of cure-fitting (the Gamma-based VDM). However, we found different guidelines, in terms of prediction accuracy. In some cases, we found that the selected model/models for a software from St. 1 is/are similar to those selected for its subversions from St. 2. However, it just occurred in few cases. For IOS (curve-fitting), AML and Normal-based VDMs were the best models in both strategies. For Chrome (curve-fitting) and one of its subversions (Chrome 3.x & 4.x), the Gamma-based VDM was the best common model considering both strategies. For Mac (prediction), the Gamma-based VDM was the most accurate model in both strategies. For Linux (prediction), the Power-law model was the best model in both strategies.

6.8 Limitations

There are several limitations to our work that prevent us from making more general conclusions. Part of the limitations we are facing with are common with previous chapters like the uncertainties associated with vulnerability databases and the

way we collect the vulnerability data reported for any version of a software from different sources and filtering them based upon the earliest date that a given vulnerability was known in any of these sources (cf. Section 4.7).

In addition, we have only applied the approach to 8 software and their subsets. We don't know yet how well this works for other software and vulnerability datasets associated with them (cf. Section 5.9).

The next limitation is that we assumed the percentage of common vulnerabilities for two or more consecutive versions of a software would be a good measure regarding the similarity of their source code (i.e., we grouped the consecutive versions with more than 70% common reported vulnerabilities). There other factors which could be considered together with our assumption to improve its validity. For example, looking at the source code of those versions and finding the number of lines with similar code. However, collecting such features is only feasible for open-source software and can't be applied on private software.

6.9 Summary

In this chapter, we developed some guidelines for analyzing vulnerabilities (i.e., those datasets where more vulnerabilities are reported earlier in the product lifecycle) using two vulnerability grouping strategies. We compared the curve fitting and prediction capabilities of eight different models: one NHPP Power-law model, two right-skewed distribution models, one flexible-skewed distribution model, three symmetric distribution models, and two non S-shaped VDMs. These models were applied on eighteen datasets that originated from four OSs and four web browsers. The datasets were built using two strategies for grouping vulnerabilities (vulnerabilities

merged for all versions and groups built based on a number of common vulnerabilities across versions). We found that a model's ability to provide a good fit does not necessarily guarantee superior prediction capabilities from the same model.

Chapter 7: Vulnerability Prediction Capability: A Comparison between Vulnerability Discovery Models and Neural Network Models

7.1 Introduction

In this chapter, we introduce an approach for predicting the total number of software vulnerabilities and compare the results with those found from aforementioned VDMs. Our approach uses a neural network model (NNM) to model the nonlinearities associated with vulnerability disclosure. Eight common VDMs were used to compare their prediction capability with NNM.

7.2 Motivation

Vulnerability discovery models (VDMs) were developed to predict future software vulnerabilities based on their historical behavior. Although VDMs are often accurate in terms of curve fitting, they might not perform well in prediction phase [25]. Indeed, VDMs are often not powerful enough to take the nonlinear nature of vulnerability disclosure times into consideration. In recent years, some software vulnerability disclosure process models were developed using traditional time series models like Auto Regressive Moving Average (ARIMA) [67]. However, vulnerability disclosure data contain a lot of nonlinearity and thus traditional time series models might not be appropriate [68]. Pokhrel et al. [69] compared the modeling capability of linear and nonlinear time series for three OSs (i.e. Windows 7, Mac OS X, and Linux

Kernel). They developed models based on ARIMA, Artificial Neural Network (ANN), and Support Vector Machine (SVM) settings.

In this chapter, we make the following contributions:

- We introduce a nonlinear modeling approach based on neural networks to model the nonlinearities associated with vulnerability disclosure times and predict the total number of software vulnerabilities in 30-day time intervals.

- We compare the prediction capability of the neural network model (NNM) with eight commonly used VDMs. We applied the models to vulnerability data associated with our aforementioned database consist of four well known operating systems (OSs) (Windows**,** Mac, IOS (the OS associated with Cisco), and Linux), as well as four well-known web browsers (Internet Explorer, Safari, Firefox, and Chrome).

- We show that the NNM outperforms the VDMs in all the cases in terms of prediction accuracy, and provides smaller values of absolute average bias in seven cases.

7.3 Data Processing

We will analyze the reported vulnerabilities associated with four well-known OSs: Windows (1995-2017), Mac (1997-2017), IOS (the OS associated with Cisco) (1992-2017), and Linux (1994-2017), as well as four well-known web browsers: Internet Explorer (1997-2017), Safari (2003-2017), Firefox (2003-2017), and Chrome (2008-2017). These software have been selected because they are the most widely used and have the most vulnerabilities in the database. Figures 8 and 9 show the detection frequency of all vulnerabilities associated with each software over time intervals of 30

days for the studied OSs and web browsers, respectively. We also plotted the 180-days moving average (MOVAVG) for each software to gain a better understanding of vulnerability detection trend. As is shown, the maximum value of MOVAVG for all cases occurred after 2015.

As mentioned in Chapter 3, for each software, we analyze all vulnerabilities reported for any of its versions. Thus, for each software, all the vulnerabilities reported for any of its versions were included. In addition, regarding our analysis, we divided the vulnerability dataset associated with each software into two groups: training and testing. The training data set consists of all the vulnerabilities reported before 2015. The testing data set consists of vulnerabilities reported in years 2015, 2016, and 2017. Table 36 represents the total number of vulnerabilities per software, as well as the number of vulnerabilities in the training and testing phases.

Table 36: NUMBER OF VULNERABILITIES PER SOFTWARE

OS	**Windows**	**Mac**	**IOS**	**Linux**
Total	3100	2705	650	4745
Train	2237	1605	438	2609
Test	863	1100	212	2136
Web Browser	**IE**	**Safari**	**Firefox**	**Chrome**
Total	1775	943	1477	1837
Train	1059	701	1150	1229
Test	716	242	327	608

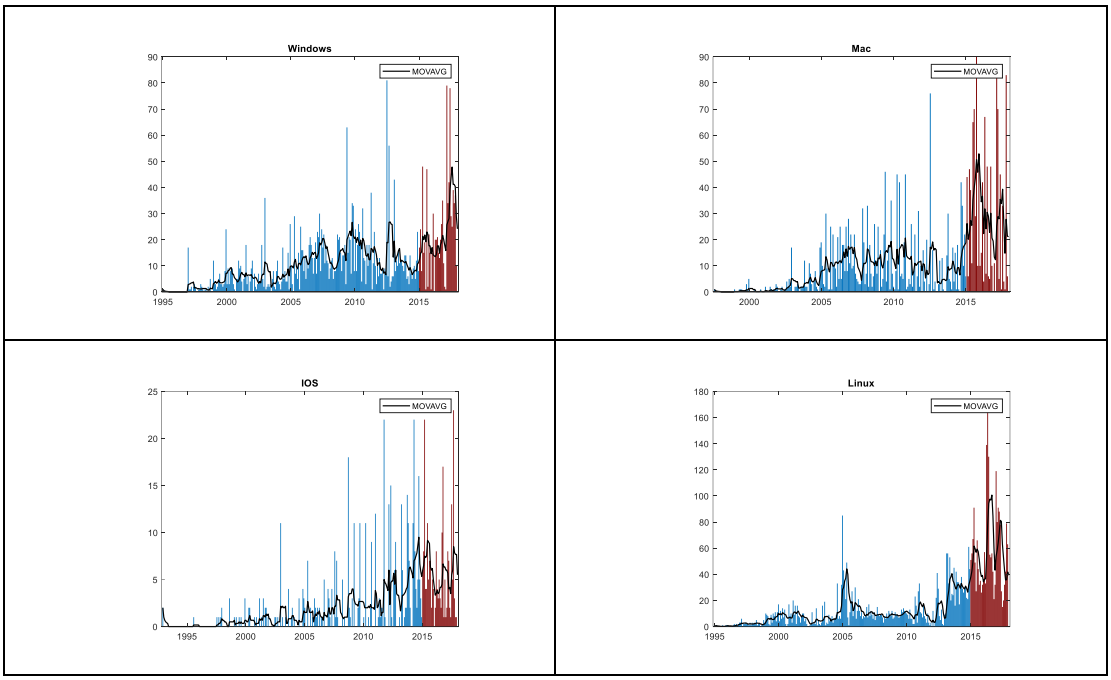

Figure 8. Histogram of the number of detected vulnerabilities per 30 days together with its 180-days moving average for the studied OSs. The X-axis represents time (Year). The Y-axis shows the frequency of discovered vulnerabilities over 30 days time intervals. The blue and red colors show data associated with the training and test datasets.

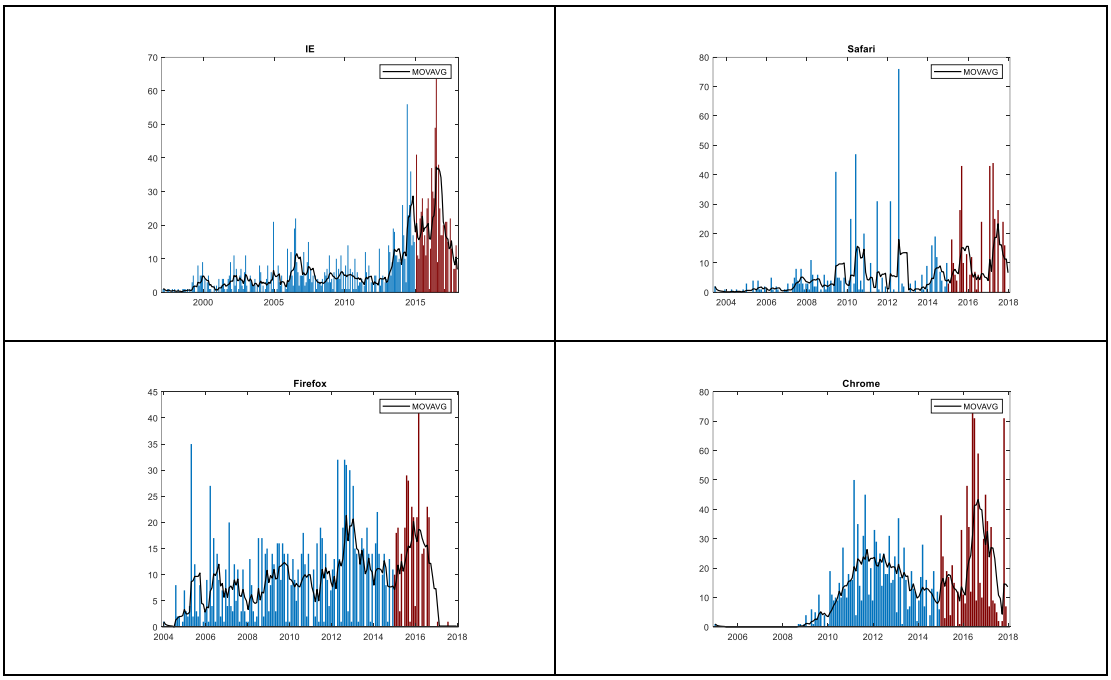

Figure 9. Histogram of the number of detected vulnerabilities per 30 days together with its 180-days moving average for the studied Web browsers. The X-axis represents time (Year). The Y-axis shows the frequency of discovered vulnerabilities over 30 days time intervals. The blue and red colors show data associated with the training and test datasets.

7.4 Neural Network Model (NNM)

Neural network models (NNMs) consist of a set of algorithms for modeling and recognizing patterns. NNMs have been widely used for predicting data with sequential time series data such as monthly electricity demand of a city or stock price [68], [70], [71]. Unlike VDMs, NNMs are capable of integrating the nonlinearity that exist in noisy time series data. In addition, NNMs are not built upon assumptions regarding the form of the basic model since they are completely data driven models. In other words, NNMs are flexible nonlinear data driven models with powerful prediction power. Data driven models are useful for the cases where there is not a theoretical guidance to explain the data generation process. It has been empirically shown that NNMs are capable of predicting both linear and nonlinear time series of different forms [72].

To predict the number of discovered vulnerabilities getting over time for a given software, we use a feedforward NNM, which is the most widely used neural network [68]. Feedforward NNMs accept a fixed number of inputs at a time and generate one output. We assume that the number of future vulnerabilities depend on the number of vulnerabilities disclosed over the past periods (lags).

We use a single hidden-layer NNM for one step-ahead prediction. According to [73], a single hidden layer NNM is capable of approximating any non-linear function with arbitrary precision. Figure 10 shows the structure of the NNM used in our study. Our feedforward NNM consists of three layers called input, hidden, and output. Each layer is a collection of neurons (nodes) where the connections are governed by the corresponding weights. Data have been fed through the input layer, and then they pass

through the one or more hidden layers, and the final outcome is provided by the output layer.

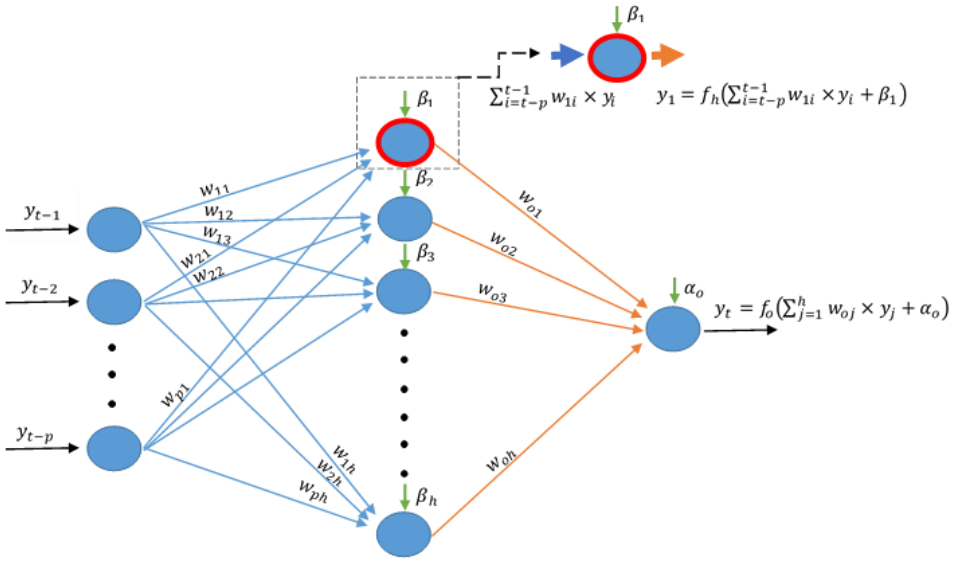

Figure 10. The NNM Architecture Used for Our Study

To predict the present value, several past observations are used. In other words, the inputs are a p-element subset of the set $\{y_{t-p}, \ldots, y_{t-2}, y_{t-1}\}$; and y_t is the output or the total number of vulnerabilities reported in period t. Equations 25 and 26 show the formulas associated with the input and output values of the hidden layer, respectively. For the output layer, the input and output values are represented by equations 27 and 28, respectively.

$$I_j = \sum_{i=t-p}^{t-1} w_{ji} \times y_i + \beta_j \quad (j = 1, \ldots, h), \tag{25}$$

$$y_j = f_h(I_j) \qquad (j = 1, \ldots, h), \tag{26}$$

$$I_o = \sum_{j=1}^{h} w_{oj} \times y_j + \alpha_o \quad (o = 1), \tag{27}$$

$$y_t = f_o(I_o) \quad (o = 1), \tag{28}$$

where I denotes the input; y denotes the output; p and h are the number of input and hidden layer nodes, respectively; w_{ji} represents the connection weights of the input and hidden layers; and w_{oj} denotes the connection weights of the hidden and output layers. The bias values of the hidden and output layers are respectively shown by β_j and α_o, and are always between -1 and 1. f_h and f_o are the non-linear activation functions associated with the hidden and the output layers, respectively. As the hidden layer activation function, we used a hyperbolic tangent function since it is the function that most widely used [68].

The initial step in designing a NNM is to determine the optimal number of input nodes (lags) and hidden layer nodes. Based on the literature, there is no systematic approach [68]; the most common way of identifying the appropriate number of the nodes (input and hidden) is via trial and error based upon finding the minimum mean square error (MSE) of the test data [74]. We followed this approach and identified the number of hidden nodes experimentally for the time series associated with each software. We evaluated up to 50 hidden nodes for each time series and chose the number of hidden nodes that minimized the MSE. Regarding the optimal number of inputs (lags), we used an optimization algorithm and chose the best combination of lags that led to the lowest MSE. We started with statistically significant lags derived from the process of evaluating the partial autocorrelation function (PACF) associated with each time series.

In time series analysis, PACF gives the linear partial correlation of a time series with its own lagged values and evaluated [75]. However, we cannot only rely on the lags we found from the PACF since, in such case, the selection of inputs would be merely based on the identification of a linear model, while the goal for using NNM is to capture non-linear correlations, as well. A very good review of existing input selection methods for NNMs is provided in [76].

The NNM developed was programmed using Matlab R2018a. For each software, we began our analysis by dividing the vulnerability dataset into two groups; training and testing. The training dataset consists of all the vulnerabilities reported before 2015. The testing data set consists of vulnerabilities reported in years 2015, 2016, and 2017. NNM training is a complex nonlinear optimization problem. Thus, there is the possibility to get trapped in local minima of the error surface. To avoid getting poor results, the training process should be repeated several times with different random starting weights and biases [72].

We set the maximum training number equal to 500 epochs. Epoch stands for the total number of times a given dataset is utilized for training and shows the number of times the weights in a network were updated [77]. Since model optimization in deep learning algorithms is done using the gradient decent method [78], it makes sense to pass the learning dataset through the network multiple times accordingly to update the weights and achieve a more accurate model, in terms of prediction [77]. We used the Levenberg-Marquardt (LM) method as our learning function. The activation function of the hidden and output layers are the *tansig* and *purelin* functions, respectively. To avoid overfitting/over training, for each software, we employed a cross validation

method by dividing our training dataset into three subgroups of training data (70%), validation data (15%), and test data (15%); and checked the validation performance of the trained network via metrics provided by Matlab Neural Network toolbox such as gradient decent (*gradient threshold*=1.00e-4) and maximum number of validation checks (*max_fail*=100). These metrics served as stop conditions of the training phase. Whenever the parameters of the network under training met any of these thresholds, the training process was stopped. This process avoids the algorithm to be over-trained and produce overfitted results.

7.5 Analysis

We used the eight VDMs introduced in Chapter 3 and one NNM for the discovery process of vulnerabilities in four OSs and four web browsers. The VDMs were fitted to the datasets using a non-linear regression method described in [24].

The analysis of the prediction capability started by dividing the data into two groups of training and testing data. Both the VDMs and the NNM use a dataset that includes all vulnerabilities reported for all versions of a given software. The training period starts from the time when the first vulnerability associated with a given software was discovered and continues until 12/31/2014. We calculated the predictions for the years 2015, 2016, and 2017. As it is shown in Figure 8 and Figure 9, the blue and the red show the data associated with training and testing datasets, respectively. We split the vulnerability data into intervals of 30 days as is common in the vulnerability analysis literature [18], [24], [25].

For the VDMs, during the training period, the training data was used to estimate model parameters. The estimated final values for each 30-day interval produced by the

eight models were compared with the actual number of vulnerabilities to calculate the prediction accuracy. For the NNM, for each software, we used the training data to train the NNM. Using the trained NNM, we predicted the expected values for the next steps. The prediction accuracy is based on the comparison between the obtained estimation and the actual number of vulnerabilities.

For the training part, for VDMs, we applied the Chi-square (χ^2) goodness of fit test to assess how well each model fits the training datasets. For the training part, for the NNM, out of the models trained with different number of lags, the optimal analytical model was selected based on the MSE value. Finally, for each software, the best selected analytical model was used to make the prediction for the testing data set (the vulnerabilities reported in 2015, 2016, and 2017). In this chapter, regarding the NNMs, we just reported the results associated with the best NNM.

For the prediction part, we calculated the two normalized predictability measures, AE, AB. These indicator and their associated equations were introduced in Chapters 4 and 6. In addition, for the VDMs, we report ΔAE_i, which represents the percentage of difference between the AE of the i-the model and the model with minimum AE.

7.6 Results

Tables 37- 38 present the values of AE, AB, ΔAE_i, and p-value (we used * to show the models with $p<0.05$) for the cases we analyzed per model (VDMs and NNM), respectively. AB can be positive (for overestimation) or negative (for underestimation), while AE is always positive. In each case, we first found the best VDMs by comparing their prediction accuracy and then compared the accuracy of those models with the

NNM. In other words, for the VDMs, the model that has the smallest value of AE was selected as having the best prediction capability and is highlighted in yellow. In addition, the VDMs with $\Delta AE_i < 2$ were also selected as the best VDMs, which, show similar prediction capability compared to the best model (the model/models with $\Delta AE_i = 0$). In addition, the normalized error values $((\Omega_t - \Omega)/\Omega)$ associated with the OSs and web browsers are plotted in Figure 11 and Figure 12, respectively. As it is shown, the models with less fluctuations lead to higher accuracy.

Table 37: PREDICTION ACCURACY FOR OSS (VDMS & NNM)

	Windows				**Mac**			
	AE	***AB***	***%ΔAEi***	***p-value***	***AE***	***AB***	***%ΔAEi***	***p-value***
Gamma	0.028	0.002	0.000	0.805	0.247	-0.247	20.537	0.817
Weibull	0.036	-0.031	0.776	0.870	0.281	-0.281	24.006	0.437
AML	0.087	-0.087	5.844	0.016*	0.287	-0.287	24.601	0.809
Normal	0.087	-0.087	5.844	0.016*	0.287	-0.287	24.601	0.809
Power-law	0.078	0.078	4.982	0.435	0.062	-0.045	2.066	0.030
RE	0.172	0.172	14.365	0.027*	0.041	0.041	0.000	0.402
RQ	0.075	0.075	4.718	0.579	0.066	-0.049	2.452	0.001*
YF	0.059	-0.057	3.067	0.178	0.279	-0.279	23.751	0.751
NNM	0.015	-0.011	NA	NA	0.018	0.010	NA	NA
	IOS				**Linux**			
	AE	***AB***	***%ΔAEi***	***p-value***	***AE***	***AB***	***%ΔAEi***	***p-value***
Gamma	0.076	-0.076	4.778	0.901	0.277	-0.277	9.302	0.828
Weibull	0.071	-0.071	4.195	0.902	0.272	-0.272	8.791	0.611
AML	0.045	-0.045	1.646	0.890	0.336	-0.336	15.146	0.000*
Normal	0.045	-0.045	1.646	0.890	0.336	-0.336	15.146	0.000*
Power-law	0.070	-0.070	4.173	0.002*	0.246	-0.246	6.204	0.342
RE	0.059	0.045	3.027	0.001*	0.184	-0.184	0.000	0.705
RQ	0.199	-0.199	16.988	0.000*	0.238	-0.238	5.341	0.342
YF	0.029	-0.018	0.000	0.930	0.307	-0.307	12.229	0.569
NNM	0.021	-0.014	NA	NA	0.040	-0.034	NA	NA

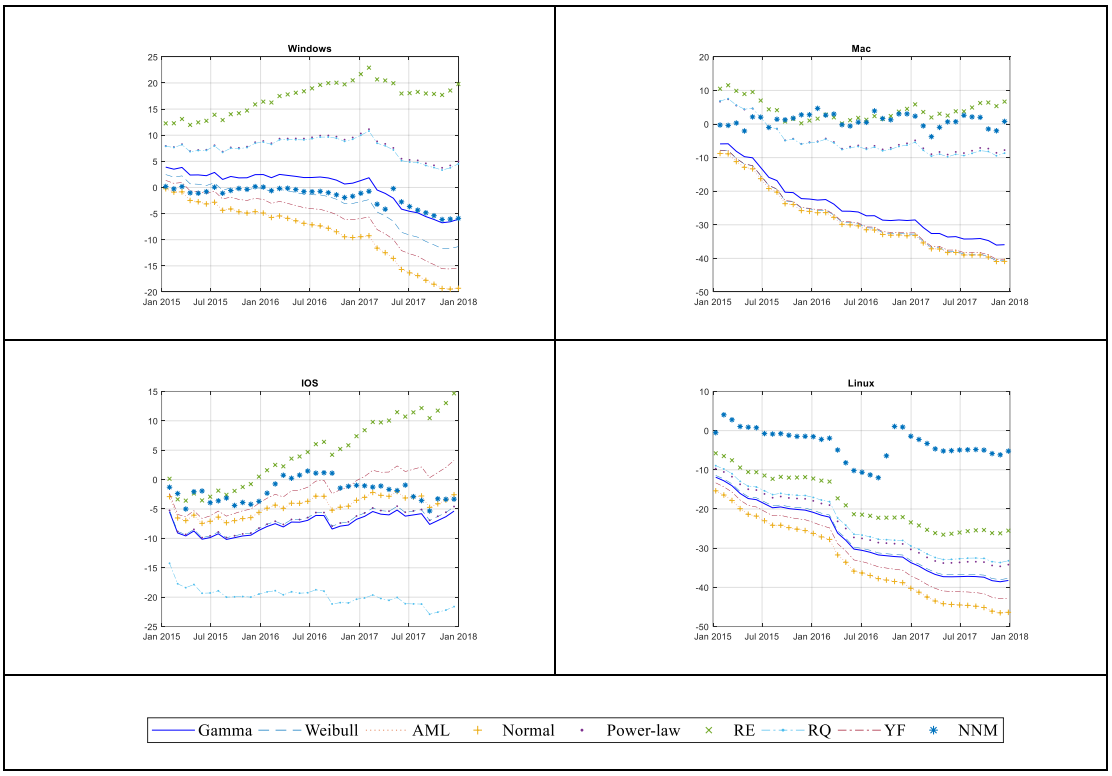

Figure 11. Prediction errors for OSs. The X-axis indicates time (Year). The Y-axis represents normalized prediction error values $((\Omega_t - \Omega)/\Omega)$.

Table 38: PREDICTION ACCURACY FOR WEB BROWSERS (VDMS & NNM)

	IE				Safari			
	AE	***AB***	***%ΔAEi***	***p-value***	***AE***	***AB***	***%ΔAEi***	***p-value***
Gamma	0.264	-0.264	5.887	0.891	0.165	-0.165	9.391	0.049
Weibull	0.264	-0.264	5.841	0.891	0.216	-0.216	14.474	0.330
AML	0.319	-0.319	11.357	0.009*	0.225	-0.225	15.371	0.330
Normal	0.319	-0.319	11.357	0.009*	0.225	-0.225	15.371	0.330
Power-law	0.264	-0.264	5.820	0.891	0.072	0.072	0.000	0.409
RE	0.206	-0.206	0.000	0.309	0.190	0.190	11.801	0.017*
RQ	0.249	-0.249	4.324	0.590	0.081	0.081	0.976	0.693
YF	0.268	-0.268	6.287	0.309	0.215	-0.215	14.336	0.978
NNM	0.028	0.006	NA	NA	0.042	-0.037	NA	NA
	Firefox				**Chrome**			
	AE	***AB***	***%ΔAEi***	***p-value***	***AE***	***AB***	***%ΔAEi***	***p-value***
Gamma	0.036	0.032	0.123	0.263	0.248	-0.248	0.000	0.890
Weibull	0.036	0.032	0.099	0.263	0.292	-0.292	4.379	0.897
AML	0.088	-0.088	5.304	0.008*	0.270	-0.270	2.174	0.805
Normal	0.088	-0.088	5.304	0.008*	0.270	-0.270	2.174	0.805
Power-law	0.056	0.056	2.089	0.263	0.336	0.336	8.802	0.000*
RE	0.160	0.160	12.452	0.091	0.623	0.623	37.481	0.000*
RQ	0.084	0.084	4.927	0.159	0.120	0.120	NA	0.000*
YF	0.035	-0.034	0.000	0.121	0.269	-0.269	2.146	0.950
NNM	0.030	-0.016	NA	NA	0.043	-0.009	NA	NA

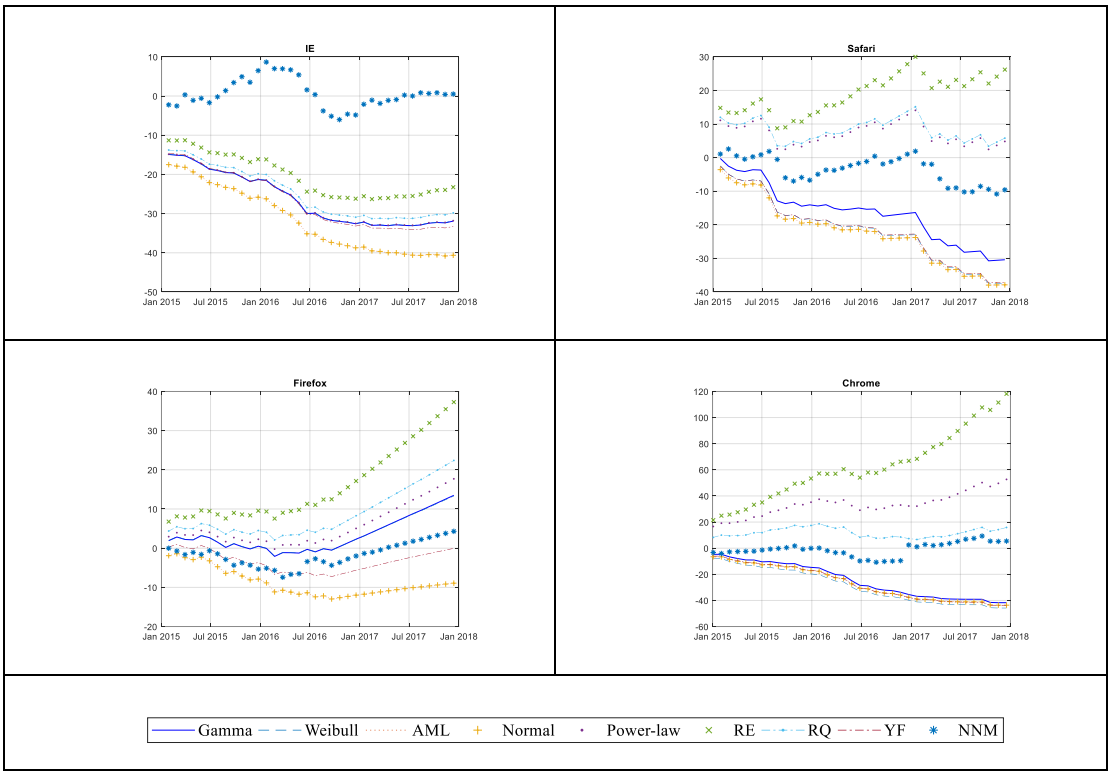

Figure 12. Prediction errors for web browsers. The X-axis indicates time (Year). The Y-axis represents normalized prediction error values $((\Omega_t - \Omega)/\Omega)$.

Based on the results provided by Tables 37-38, in terms of prediction accuracy (AE), the NNM led to most accurate results in all of the eight software we analyzed. To be more precise, for Windows, the NNM's average error (AE) is 1.3%, and 2.1% smaller than the AEs associated with the best VDMs, which were Gamma and Weibull-based VDMs. For Mac, this difference is 2.3%. For IOS, the NNM outperforms the best VDMs (YF, AML and Normal) by having 0.8%, 2.4%, and 2.4% smaller average errors, respectively. Linux and IE are two of the cases where the NNM provides far better predictions than those from VDMs by being 14.4%, and 17.8% more accurate. The average error of the NNM for Safari is 3% and 3.9% smaller than those from the best VDMs (Power-law and RQ).

For Firefox, the NNM improved the predictions by 0.5%, 0.6%, and 0.6% compared to YF, Gamma, and Weibull-based VDMs, respectively. For Chrome, the VDM with smallest is not statistically sound from the training part. So, we opt for the next VDM with p-value>0.05 and smallest AE, which is Gamma. In this case, the NNM accuracy improvement is 20.5%. Overall, the highest differences in prediction accuracy between the NNM and the VDMs were found in Chrome (20.5%), IE (17.8%), Linux (14.4%), and Safari (3.9%), respectively.

In terms of magnitude of error, out of eight software we analyzed, the NNM outperformed the VDMs in seven cases by having smaller |AB| values. Only for Windows, the absolute value of bias provided by one of the selected VDMs was 0.9% smaller than the one resulted from the NNM. For Mac, IOS, Linux, IE, Safari, Firefox, and Chrome, the bias magnitudes provided by the NNM were smaller than those from the best VDMs (in each case, we considered the best VDM, which had smallest |AB|) by 3.1%, 0.4%, 15%, 20%, 3.5%, 1.6%, and 23.9%, respectively.

Overall, in terms of accuracy, out of the eight cases we analyzed, the NNM outperformed VDMs in all the cases. Besides, in terms of magnitude of bias, the NNM led to smallest bias values in seven cases.

Please remember that all the conclusions we draw from this research and their validity are limited by our database uncertainties.

7.7 Discussion

In terms of prediction accuracy (AE), considering the OSs and web browsers, the NNM led to more accurate results than the best selected VDMs in all the cases. The Gamma-based VDM was selected as the best model in three cases out of four. The

Weibull and YF VDMs were each best compared with other models in one case out of four.

In terms of overall magnitude of bias (i.e., absolute value of AB), out of the eight cases we analyzed, the NNM provided smaller absolute values of bias in seven cases compared to the best VDMs. Only for Windows, the absolute value of bias provided by the Gamma-based VDM (0.002) was smaller than the one resulted from NN (-0.011).

We believe that the final decision, in equal accuracy conditions, in terms of bias, is up to the researcher to choose the best model based upon his/her priorities. However, from a security point of view, it is better to choose a model, which provides more conservative prediction results. In the current study, among the models that were selected as the best predictors, only two NNMs (Mac and IE) provided overestimated results. Other selected NNMs underestimated the number of vulnerabilities. It can also be easily inferred from Figure 11 and Figure 12, where for Mac and IE most of the prediction points associated with the NNMs are located over the X=0 axis.

We believe that the NNM's better performance compared to VDMs comes from the capability of the NNM in predicting the nonlinearity nature of the vulnerability disclosure time series. In addition, most VDMs consider the vulnerability discovery process as a pure S-shaped curve or a function with a monotonic intensity function with constant total number of vulnerabilities. However, the number of vulnerabilities associated with a given software may change as newer versions are released. Additionally, VDMs and traditional time-series functions only use one set of parameters for estimation. On the other hand, NNMs due to having multilayer

perceptron structure, having multiple neurons per layer, and using different set of parameters per neuron provide a more complicated structure for prediction. Of course, the specific validation method we used to avoid being trapped by overfitting in the learning phase is another advantage of using NNMs.

7.8 Limitations

There are a few limitations to our work that prevents us from expanding our conclusions in a more generalized manner. Some of them are common with previous chapters of this dissertation such as uncertainties associated with vulnerability data public repositories (cf. Section 4.7).

Another common limitation is with respect to utilizing announced published date of vulnerabilities as their discovery date. Vulnerabilities normally get identified before by pernicious users than the time they are officially reported. To ensure that this gauge is as close as conceivable to the real date the vulnerability is publicly known to the world, we searched for various vulnerability repositories and selected the earliest date announced for a vulnerability (cf. Section 4.7).

Another limitation is as to the manner in which we combined all vulnerabilities announced for all versions of a given software to have sufficient data for training the models (cf. Section 5.9). While number of studies utilize vulnerability data associated with separate version of software (e.g. Windows 7) on which to apply VDMs, there are papers that consider all versions of a software together [25], [62]. The first group expects that each version of a given software is an independent and all around characterized item, yet distinguishing the sources of reliance in vulnerability data is not a simple task.

The next limitation is that we only used one machine learning algorithm (Neural Network) to show the better performance of such algorithms over the commonly used VDMs. There are several other algorithms like SVMs, which could also be used. However, since our research question was not associated with comparing the prediction performance of other machine learning algorithms, we only chose one of them.

7.9 Summary

In this chapter, we compared the capabilities of eight common vulnerability discovery models (VDMs) with a nonlinear neural network model (NNM) in terms of predicting the total number of future vulnerabilities over a prediction period of three years. We applied the mentioned models to vulnerability data associated with four well known OSs and four well-known web browsers. The models were assessed in terms of prediction accuracy and prediction bias. The results showed that the NNM outperformed the VDMs in all the cases in terms of prediction accuracy. In terms of overall magnitude of bias, out of the eight cases we analyzed, the NNM provided the smallest absolute values of bias in seven cases compared to the best VDMs. This chapter shows that neural networks are promising for accurate predictions of the total number of software vulnerabilities.

Chapter 8: Predicting Exploited Vulnerabilities

8.1 Introduction

Exploited vulnerabilities typically form 2-7% of all the vulnerabilities reported for a given software. With a smaller number of known exploited vulnerabilities compared with the total number of vulnerabilities, it is more difficult to mathematically model and predict when a vulnerability with a known exploit will be reported. In this chapter, we introduce an approach for predicting the total number of publicly-known exploited vulnerabilities using all publicly-known vulnerabilities reported for a given software. Eight commonly used VDMs and one NNM were utilized to evaluate the prediction capability of our approach. We compared their predictions results with the scenario when only exploited vulnerabilities were used for prediction.

8.2 Motivation

For some vulnerabilities, exploits are never published. This might be because the patches for these vulnerabilities are made available very quickly by the vendors, and hence it is not profitable for the crackers to develop exploits for them; the vulnerabilities have a lower criticality from the security viewpoint; or it might be that that exploits for these vulnerabilities are only known to the vendors, to security agencies or are exchanged in, for example, darkweb forums. Whatever the explanation may be, vulnerabilities with publicly-known exploits usually form only 2-7% of all the vulnerabilities reported for an specific version of given software [47], [79]. In addition, as opposed to vulnerability databases such as NVD, which are actively maintained,

security repositories reporting exploited vulnerabilities like Exploit Database, also known as “ExploitDB”, are less common. A comparison between NVD and ExploitDB finds that only 22% of NVD distinct vulnerabilities have exploits reported and listed in ExploitDB. On the other hand, vulnerabilities with known exploits are more dangerous to end users, even if patches may be available, since not all users regularly patch their systems. For this reason, it is important for both vendors and users to be able to predict the time to the next vulnerability with a known exploit. However, with a smaller number of known exploited vulnerabilities compared with the total number of vulnerabilities, it is difficult to model and predict when a vulnerability with a known exploit will be reported. Specifically, the data scarcity makes it difficult to use data driven models, which are helpful where there is no theoretical guidance to explain the data generation process, for such data [80]. Therefore, we postulate that it is a worthwhile research activity to explore whether there is a link between disclosure times of all vulnerabilities reported for a given software and discovery times of its exploited vulnerabilities. Finding such link would allow to use a larger dataset of all vulnerabilities for predicting the time when vulnerabilities with exploits will be reported.

To the best of our knowledge, there is no research focusing on modeling exploited vulnerabilities. The only efforts in this area is probabilistic examination of intrusions by [52], [53]. Lack of data is a big barrier on the way to modeling exploited vulnerabilities using current VDMs or the machine learning techniques, which require considerable amount of data for satisfactory training.

In this chapter, we introduce an approach for predicting the total number of publicly-known exploited vulnerabilities using all vulnerabilities reported for a given software. Eight commonly used VDMs as well as one NNM were used to evaluate the prediction capability of our approach. We applied the models to vulnerability data associated with four well-known OSs (Windows, Mac, IOS (the OS associated with Cisco), and Linux), as well as four well-known web browsers (Internet Explorer, Safari, Firefox, and Chrome).

Our work makes the following contributions:

- We introduce an approach for predicting total number of publicly-known exploited vulnerabilities using all vulnerabilities reported for a given software;
- We compare the prediction capability of two scenarios S1 and S2, S1 when all the vulnerabilities are considered, S2 when only exploited vulnerabilities are, utilizing eight VDMs and one NNM on eight well-known software;
- We show that, out of eight software we analyzed, scenario S1 outperforms scenario S2 in seven cases in terms of prediction accuracy. Only in one case, the prediction of S1 was worse than S2 by 1.5%. In other words, for most of the cases analyzed, we show that using all the vulnerability data available for a system allows to better predict when vulnerabilities that will have publicly-known exploits for them will be reported

8.3 Data Processing

We will analyze the reported vulnerabilities associated with four well-known OSs: Windows (1995-2017), Mac (1997-2017), IOS (the OS associated with Cisco)

(1992-2017), and Linux (1994-2017), as well as four well-known web browsers: Internet Explorer (1997-2017), Safari (2003-2017), Firefox (2003-2017), and Chrome (2008-2017). These software have been selected because they are the most widely used and have the most vulnerabilities in the database.

For each software, all the vulnerabilities reported for any of its versions were included. For instance, all the vulnerabilities reported for mac_os, mac_os_server, mac_os_x, and mac_os_x_server were put together to create a vulnerability database for Mac.

Two scenarios were considered. In the first scenario (S1), we analyze all vulnerabilities reported for a software for any of its versions. In the second scenario (S2), for each software, we only consider the exploited vulnerabilities.

Table 39 presents the total number of vulnerabilities for each software (All vulnerabilities together ("S1") and only exploited vulnerabilities ("S2")). The percentages of exploited/unexploited vulnerabilities per software are presented in Figure 13. Windows and IE had the most percentages of exploited vulnerabilities with 24.13% and 22.65%, respectively.

Table 39: NUMBER OF VULNERABILITIES PER SOFTWARE (ALL VS. EXPLOITED)

OS	**Windows**	**Mac**	**IOS**	**Linux**
# All Vulnerabilities	3100	2705	650	4745
# Exploited Vulnerabilities	748	282	27	481
Web Browser	**IE**	**Safari**	**Firefox**	**Chrome**
# All Vulnerabilities	1775	943	1477	1837
# Exploited Vulnerabilities	402	108	100	78

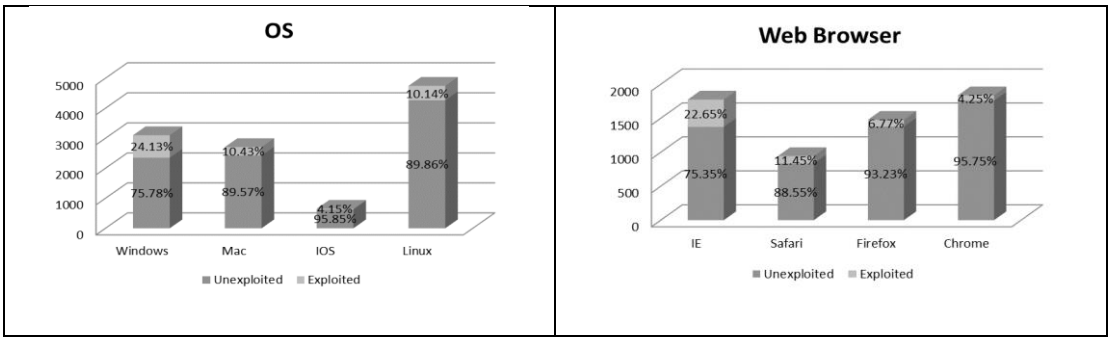

Figure 13. Percentage of exploited vulnerabilities per software

8.4 Analytical steps of scenario S1

8.4.1 For the VDMs

In this section, we explain the approach we developed to predict the number of publicly-reported exploited vulnerabilities associated with a given software using all vulnerabilities reported for the software. Regarding VDMs, we need to find a relationship between the discovery pattern of all vulnerabilities (S1) and those vulnerabilities that were exploited (S2). We focused on the ratio of the time to next vulnerability (TTNV) for exploited vulnerabilities over the TTNV associated with all vulnerabilities. We use this ratio as a multiplier in the equations associated the VDMs in the training phase to approximate the VDMs' equations for exploited vulnerabilities. We use a resampling method and a filtering method to take care of the noisy nature of vulnerability data [81], [82]. For each software, we resample/split the vulnerability data (all vulnerabilities & exploited vulnerabilities) into intervals of 120, 150, 180, 210, 240, 270, 300, 330, 360 days to remove the effect of daily fluctuations. For each interval (i-th interval), we calculate the mean TTNV of the observations at each time step (MTTNV) and calculate the ratio of MTTNVs, $\text{Ratio}_{interval\ i}(t) = \text{MTTNV}_{Exploited}(t)/$

$\text{MTTNV}_{All}(t)$. Figures 14 and 15 show the box plot of ratios associated with each interval per software. The median of the ratios for each software, $Median\ (\text{Ratio}_{interval\ i}(t))$ is almost constant over different intervals. The median values of the ratios per software are presented in Table 40. The VDM for exploited vulnerabilities is calculated as follows:

$$\Omega(t)_{Exploited} = \Omega(t)_{All}/Median\left(\text{Ratio}_{interval\ i}(t)\right) \quad (5)$$

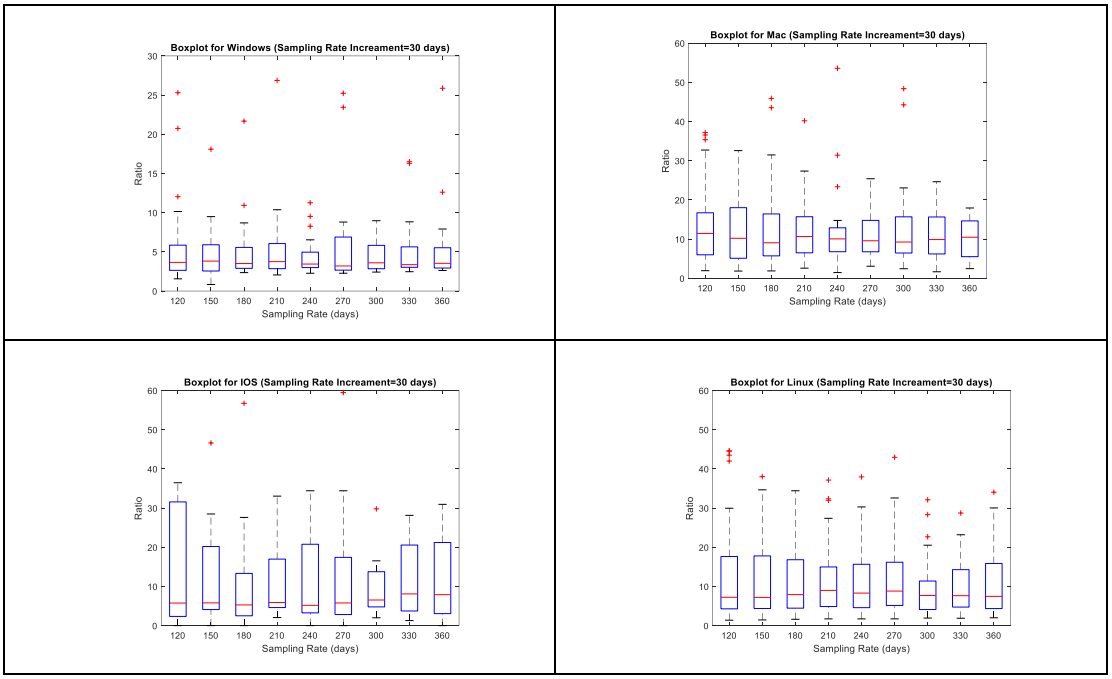

Figure 14. Box chart for TTNV coefficient ratio per OS (S2/S1)

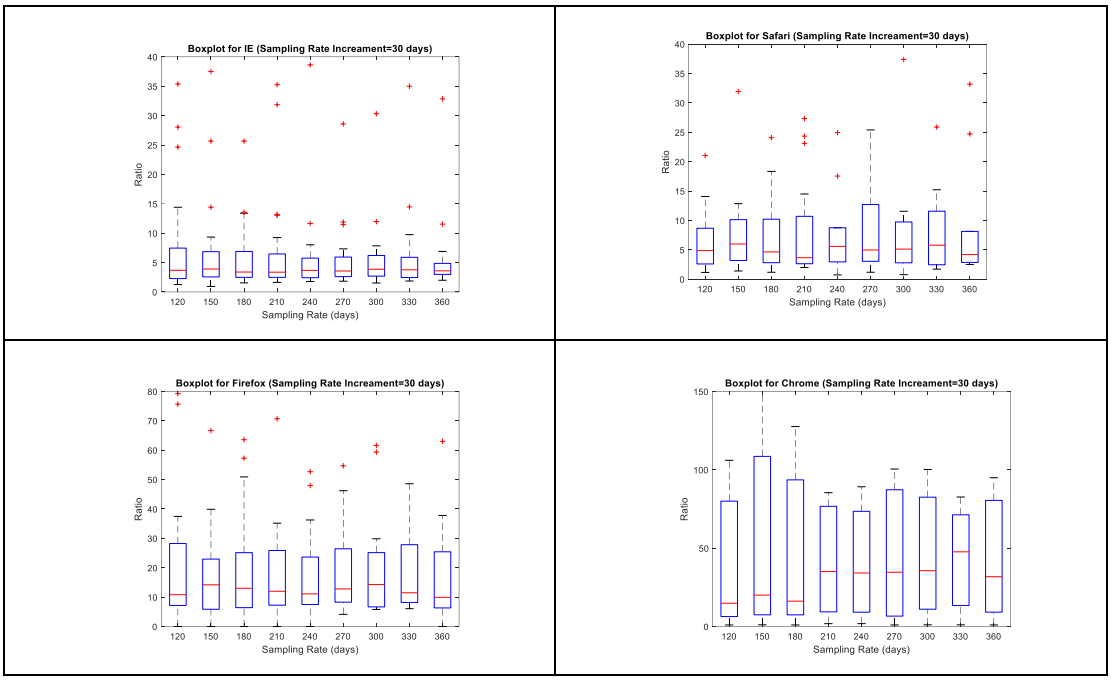

Figure 15. Box chart for TTNV coefficient ratio per web browser (S2/S1)

Table 40: TABLE OF TTVN MEAN RATIOS PER SOFTWARE

OS	**Windows**	**Mac**	**IOS**	**Linux**
TTNV Ratio (Median)	3.526	9.393	5.696	5.306
Web Browser	**IE**	**Safari**	**Firefox**	**Chrome**
TTNV Ratio (Median)	3.360	5.792	10.917	50.127

8.4.2 For the NNM

Regarding the NNM, since we want to link two time series, we feed one time series (all vulnerabilities) into the NNM as input and select the output (y_t) from the second time series (exploited vulnerabilities). In other words, the vector of inputs $\{y_{t-p}, \ldots, y_{t-2},\ y_{t-1}\}$ belongs to S1 and the output is chosen from S2. Details on how the NNM was developed are in chapter 7.

8.5 Analysis

For both scenarios (S1 and S2), we used the eight VDMs for the discovery process of vulnerabilities on eight well-known software (four OSs and four web browsers). The VDMs were fitted to the datasets using a non-linear regression method described in [24]. In addition, for the first scenario (S1) we also used one NNM, which is capable of modeling nonlinearities. Since the NNM is a data driven model, we could not use it for scenario S2 due to lack of exploited vulnerabilities.

We started the analysis by splitting the data into two groups of training and testing data. For scenario S1, both the VDMs and the NNM use a dataset that includes all vulnerabilities reported for all versions of a given software. For scenario S2, the VDMs use the data associated with exploited vulnerabilities reported for those versions. The training period for both scenarios starts from the time when the first exploited vulnerability associated with a given software was reported and continues until 12/31/2014. We made the predictions for the years 2015, 2016, and 2017. We then split the vulnerability data into intervals of 30 days as is common in the vulnerability analysis literature [18], [24], [25].

For scenario S1, for the VDMs, during the training period, the training data was used to estimate model parameters. Using the estimated parameters and the TTNV ratios we found from Section 8.4.1, we estimated the number of exploited vulnerabilities. Then, the estimations for each 30-day interval produced by the eight models were compared with the actual number of exploited vulnerabilities to calculate the prediction accuracy. For the NNM, for each software, we used the training data to train the NNM. The process is like feeding the NNM by one time series and comparing

the outputs with values associated with another time series. Using the trained NNM, we predicted the number of exploited vulnerabilities for the next steps. We calculated the prediction accuracy by comparing the obtained estimation and the actual number of exploited vulnerabilities.

For scenario S2, for the VDMs, during the training period, the training data was used to estimate model parameters. The estimated final values for each 30-day interval produced by the eight models were compared with the actual number of exploited vulnerabilities to calculate the prediction accuracy.

The Chi-square (χ^2) goodness of fit test [24] was utilized for evaluating the quality of fit of each model on training datasets. For the training part, for the NNM, we used the MSE value to select the optimal analytical model, out of the models trained with different combinations of lags. Then, for each software, the best model was selected to make the prediction for the testing data set (the vulnerabilities reported in 2015, 2016, and 2017). In this chapter, regarding the NNMs, we report the results associated with the best NNM.

For the prediction part, we calculated the two normalized predictability measures, AE, AB. These indicators and their associated equations were introduced in Chapters 4 and 6. For the VDMs associated with each scenario, we also report ΔVAE_i^k, which shows the difference between the AE of the i-the VDM and the VDM with minimum AE in the scenario to choose the best VDM(s) for each scenario.

$$\%\Delta\text{VAE}_i^k = (\text{VAE}_i^k - \text{VAE}_{min}^k) * 100 \quad (29)$$

where k is the k-th scenario, VAE_i is the AE of the i-th VDM, and VAE_{min} is the lowest AE found in the set of VDMs examined in the scenario (i.e., the best model). Thus, the $\Delta\mathrm{VAE}_i^k$ of the best VDM in a scenario is 0.

To highlight the difference between the AE of the k-th model and the overall best model in both scenarios, we report $\Delta\mathrm{AE}_i^G$, which is defined as follows:

$$\%\Delta\mathrm{AE}_j^G = (\mathrm{AE}_j \quad - \mathrm{AE}_{min}^G) * 100 \tag{30}$$

where AE_j is the AE of the j-th model, and AE_{min}^G is the lowest AE found in the set of models examined (i.e., the best model). Thus, $\Delta\mathrm{AE}_j^G$ of the best overall model is 0. In addition, if for a given model we have $\Delta\mathrm{AE}_j^G = 1.2$, it means than the model has 1.2% higher prediction error than the best overall model.

8.6 Results

Tables 41-42 present the values of AE, AB, $\Delta\mathrm{VAE}_i^k$, and $\Delta\mathrm{AE}_j^G$ for the cases we analyzed per scenario per model (VDMs and NNM), respectively. Regarding p-values, we used * to show the models with p<0.05. AB can be positive (for overestimation) or negative (for underestimation), while AE is always positive. In each case, we first found the best VDM(s) per scenario by comparing their prediction accuracy and then compared the accuracy of those models with the NNM results. In other words, for each software, for the VDMs associated with each scenario, the models that had the smallest values of AE were selected as the best VDMs in terms of prediction and were highlighted in yellow. In addition, the VDMs with $\Delta\mathrm{VAE}_i^k < 2$ were also selected as

the best VDMs, which we assume, show similar prediction capability compared to the best VDM (the VDM(s) with $\Delta \mathrm{VAE}_i^k = 0$). For each software, the best overall model in both scenarios is colored in green (the model with $\Delta \mathrm{AE}_j^G = 0$). If a VDM is the best model of a scenario and simultaneously is the best overall model, is only colored in green.

For each software, the normalized error values $((\Omega_t - \Omega)/\Omega)$ over prediction time are plotted in Figure 16 and Figure 17. The models with less fluctuations lead to higher accuracy.

Table 41: PREDICTION ACCURACY FOR OSS PER SCENARIO (VDMS & NNM)

Windows	**S1**				**S2**			
	AE	***AB***	$\%\Delta AE_i^1$	$\%\Delta AE_j^G$	***AE***	***AB***	$\%\Delta AE_i^2$	$\%\Delta AE_j^G$
Gamma	0.187	0.187	8.126	12.111	0.101	-0.097	1.703	3.539
Weibull	0.148	0.148	4.245	8.230	0.139	-0.139	5.432	7.268
AML	0.106	0.083	0.000	3.984	0.145	-0.145	6.021	7.856
Normal	0.106	0.083	0.000	3.984	0.145	-0.145	6.021	7.856
Power-law	0.277	0.277	17.076	21.061	0.084	0.084	0.000	1.836
RE	0.387	0.387	28.122	32.106	0.138*	0.138	NS	NS
RQ	0.274	0.274	16.766	20.750	0.113	0.113	2.892	4.727
YF	0.122	0.118	1.576	5.560	0.140	-0.140	5.579	7.415
NNM	0.066	-0.024	NA	0.000	NA	NA	NA	NA
Mac	**S1**				**S2**			
	AE	***AB***	$\%\Delta AE_i^1$	$\%\Delta AE_j^G$	***AE***	***AB***	$\%\Delta AE_i^1$	$\%\Delta AE_j^G$
Gamma	0.215	-0.215	14.203	14.985	0.261	-0.261	0.000	19.599
Weibull	0.251	-0.251	17.772	18.553	0.282	-0.282	2.038	21.637
AML	0.257	-0.257	18.399	19.180	0.277	-0.277	1.577	21.177
Normal	0.257	-0.257	18.399	19.180	0.277	-0.277	1.577	21.177
Power-law	0.073	-0.008	0.000	0.781	0.101*	-0.008	NS	NS
RE	0.081*	0.081	NS	NS	0.092*	0.005	NS	NS
RQ	0.077	-0.011	0.341	1.122	0.094*	0.017	NS	NS
YF	0.248	-0.248	17.513	18.295	0.280	-0.280	1.834	21.433
NNM	0.065	0.026	NA	0.000	NA	NA	NA	NA
IOS	**S1**				**S2**			
	AE	***AB***	$\%\Delta AE_i^1$	$\%\Delta AE_j^G$	***AE***	***AB***	$\%\Delta AE_i^2$	$\%\Delta AE_j^G$
Gamma	0.149	0.146	10.441	13.206	0.032	0.018	0.445	1.524
Weibull	0.156	0.154	11.126	13.891	0.029	0.001	0.182	1.260
AML	0.185	0.185	14.066	16.831	0.028	-0.014	0.000	1.079
Normal	0.185	0.185	14.066	16.831	0.028	-0.014	0.000	1.079
Power-law	0.156	0.154	11.153	13.918	0.240*	0.240	NS	NS
RE	0.301	0.301	25.647	28.412	0.279*	0.279	NS	NS
RQ	0.044	-0.007	0.000	2.765	0.153*	0.153	NS	NS
YF	0.220	0.220	17.555	20.321	0.028	-0.014	0.080	1.158
NNM	0.017	-0.002	NA	0.000	NA	NA	NA	NA
Linux	**S1**				**S2**			
	AE	***AB***	$\%\Delta AE_i^1$	$\%\Delta AE_j^G$	***AE***	***AB***	$\%\Delta AE_i^1$	$\%\Delta AE_j^G$
Gamma	0.132	0.132	8.919	11.140	0.116	-0.116	7.620	9.542
Weibull	0.140	0.140	9.732	11.954	0.128	-0.128	8.874	10.796
AML	0.043	0.037	0.000	2.221	0.168	-0.168	12.811	14.733
Normal	0.043	0.037	0.000	2.221	0.168	-0.168	12.811	14.733
Power-law	0.182	0.182	13.914	16.135	0.040	0.040	0.000	1.922
RE	0.282	0.282	23.969	26.190	0.045	0.045	0.540	2.462
RQ	0.196	0.196	15.291	17.513	0.050	0.050	1.031	2.953
YF	0.084	0.084	4.135	6.356	0.158	-0.158	11.843	13.765
NNM	0.020	0.019	NA	0.000	NA	NA	NA	NA

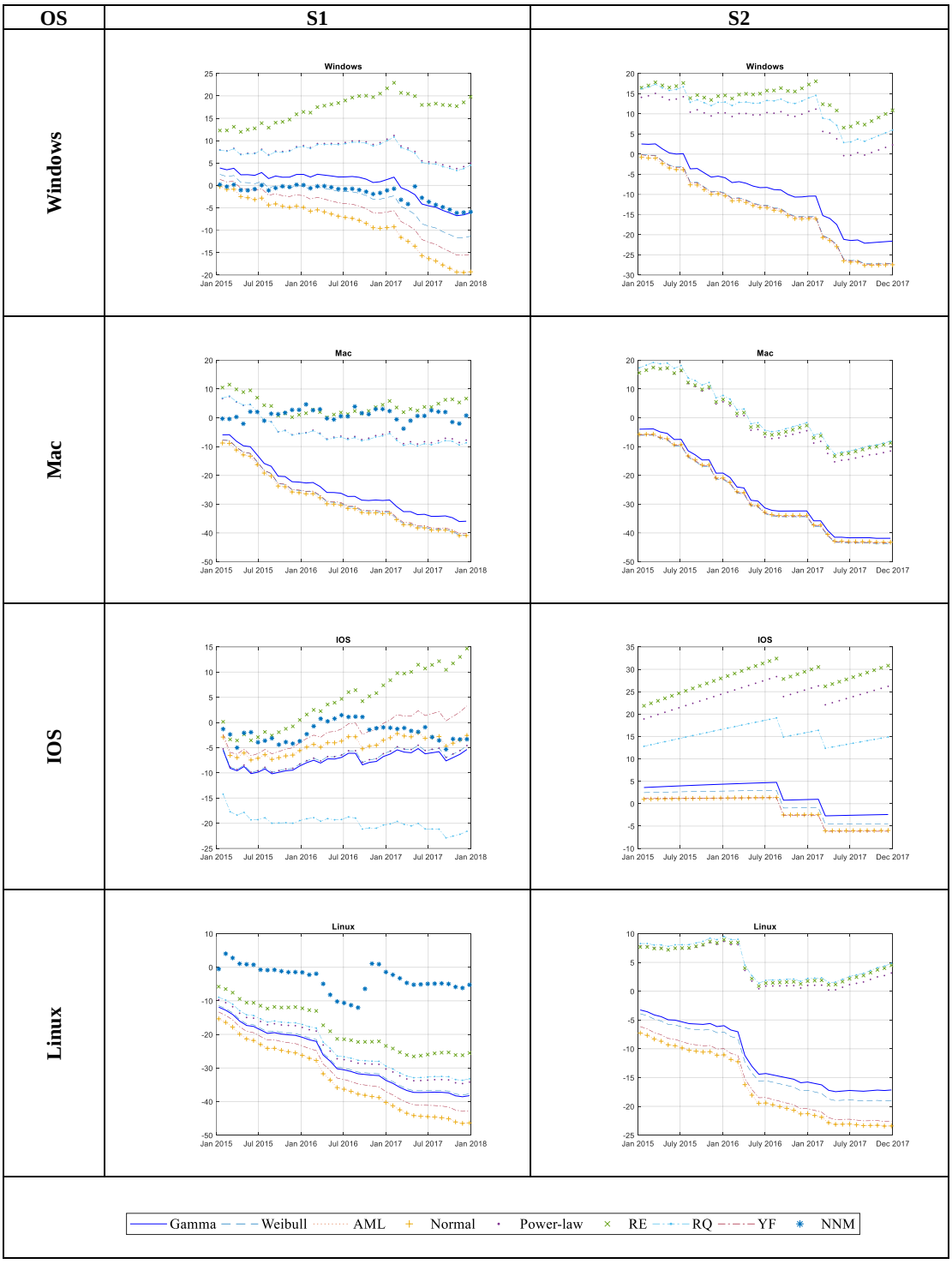

Figure 16. Prediction errors for Oss per scenario. The X-axis indicates time (Year). The Y-axis represents normalized prediction error values $((\Omega_t - \Omega)/\Omega)$.

Table 42: PREDICTION ACCURACY FOR WEB BROWSERS PER SCENARIO (VDMS & NNM)

IE	S1				S2			
	AE	***AB***	$\%\Delta AE_i^1$	$\%\Delta AE_j^G$	***AE***	***AB***	$\%\Delta AE_i^2$	$\%\Delta AE_j^G$
Gamma	0.121	-0.121	7.189	9.018	0.175	-0.175	11.004	14.475
Weibull	0.120	-0.120	7.138	8.967	0.182	-0.182	11.700	15.170
AML	0.188	-0.188	13.922	15.752	0.256	-0.256	19.048	22.519
Normal	0.188	-0.188	13.922	15.752	0.256	-0.256	19.048	22.519
Power-law	0.120	-0.120	7.113	8.942	0.087	-0.087	2.172	5.643
RE	0.049	-0.049	0.000	1.829	0.065	-0.065	0.000	3.471
RQ	0.102	-0.102	5.292	7.121	0.069	-0.069	0.386	3.856
YF	0.126	-0.126	7.717	9.547	0.228	-0.228	16.253	19.724
NNM	0.030	0.010	NA	0.000	NA	NA	NA	NA
Safari	**S1**				**S1**			
	AE	***AB***	$\%\Delta AE_i^1$	$\%\Delta AE_j^G$	***AE***	***AB***	$\%\Delta AE_i^2$	$\%\Delta AE_j^G$
Gamma	0.498	-0.498	20.921	41.101	0.140	-0.076	1.386	5.308
Weibull	0.528	-0.528	23.943	44.123	0.126	-0.106	0.000	3.923
AML	0.533	-0.533	24.492	44.672	0.131	-0.095	0.500	4.423
Normal	0.533	-0.533	24.492	44.672	0.131	-0.095	0.500	4.423
Power-law	0.357	-0.357	6.898	27.078	0.285	0.224	15.963	19.886
RE	0.288	-0.288	0.000	20.181	0.265	0.193	13.924	17.847
RQ	0.351	-0.351	6.312	26.493	0.270	0.202	14.462	18.384
YF	0.527	-0.527	23.863	44.043	0.127	-0.104	0.101	4.024
NNM	0.087	0.042	NA	0.000	NA	NA	NA	NA
Firefox	**S1**				**S1**			
	AE	***AB***	$\%\Delta AE_i^1$	$\%\Delta AE_j^G$	***AE***	***AB***	$\%\Delta AE_i^2$	$\%\Delta AE_j^G$
Gamma	0.324	0.324	15.768	31.419	0.010	0.005	0.000	0.000
Weibull	0.324	0.324	15.743	31.394	0.029	-0.029	1.912	1.912
AML	0.167	0.167	0.000	15.651	0.064	-0.064	5.344	5.344
Normal	0.167	0.167	0.000	15.651	0.064	-0.064	5.344	5.344
Power-law	0.355	0.355	18.888	34.539	0.209*	0.209	NS	NS
RE	0.492	0.492	32.510	48.162	0.199*	0.199	NS	NS
RQ	0.393	0.393	22.610	38.261	0.170*	0.170	NS	NS
YF	0.238	0.238	7.097	22.748	0.064	-0.064	5.370	5.370
NNM	0.026	-0.024	NA	1.594	NA	NA	NA	NA
Chrome	**S1**				**S1**			
	AE	***AB***	$\%\Delta AE_i^1$	$\%\Delta AE_j^G$	***AE***	***AB***	$\%\Delta AE_i^2$	$\%\Delta AE_j^G$
Gamma	0.531	-0.531	0.000	35.283	0.363	-0.363	9.162	18.482
Weibull	0.557	-0.557	2.635	37.918	0.359	-0.359	8.724	18.045
AML	0.544	-0.544	1.324	36.606	0.409	-0.409	13.722	23.043
Normal	0.544	-0.544	1.324	36.606	0.409	-0.409	13.722	23.043
Power-law	0.210*	-0.204	NS	NS	0.285	-0.231	1.404	10.725
RE	0.108*	-0.052	NS	NS	0.271	-0.148	0.000	9.321
RQ	0.325*	-0.325	NS	NS	0.330	-0.328	5.831	15.152
YF	0.544	-0.544	1.275	36.557	0.473*	-0.473	NS	NS
NNM	0.178	-0.157	NA	0.000	NA	NA	NA	NA

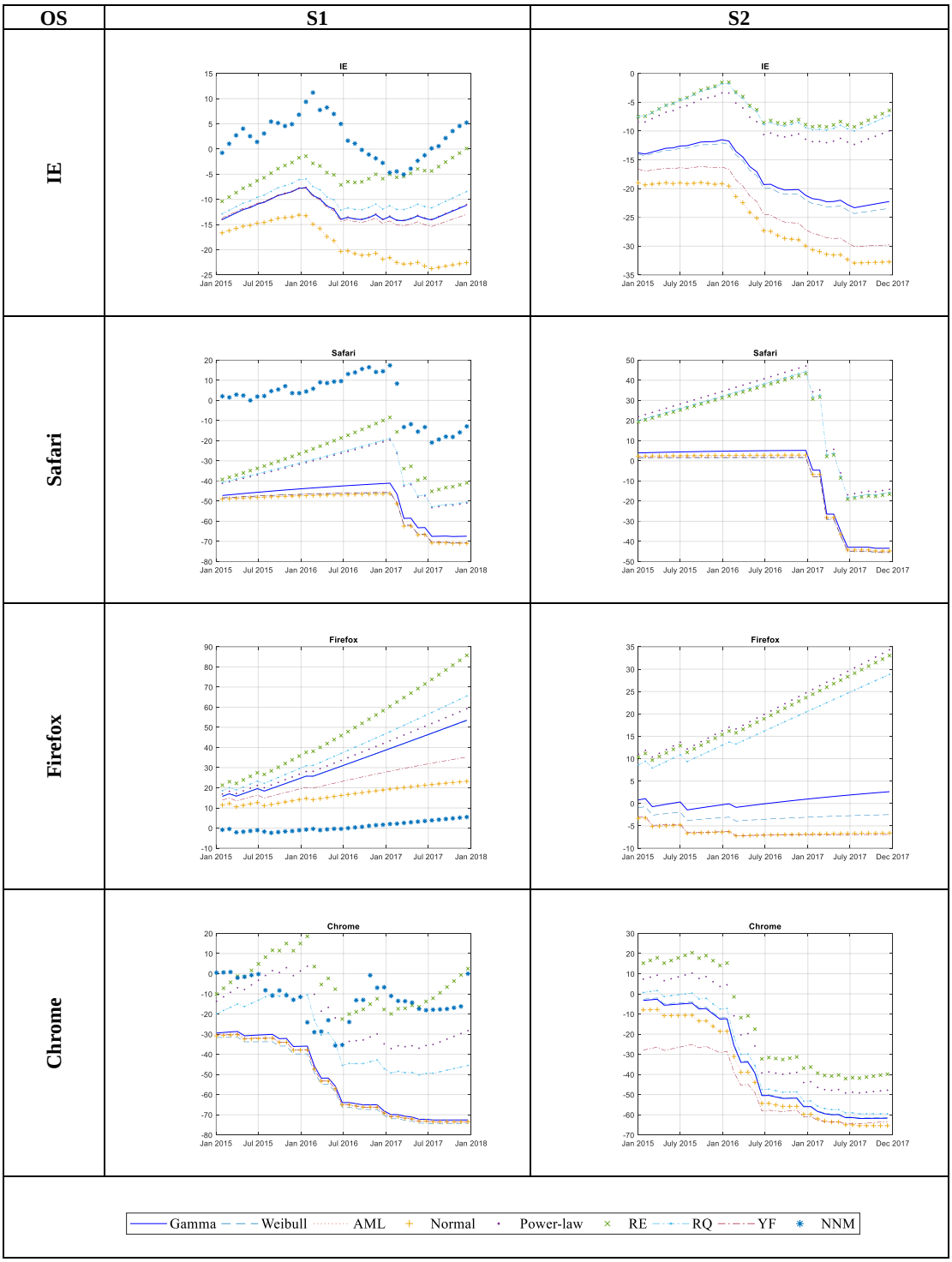

Figure 17. Prediction errors for web browsers per scenario. The X-axis indicates time (Year). The Y-axis represents normalized prediction error values $((\Omega_t - \Omega)/\Omega)$.

Based upon the results provided by Tables 41-42, in terms of prediction accuracy (AE and HH), out of eight software we analyzed, scenario S1 led to the most accurate results in seven cases. Only for Firefox, the best VDM from scenario S2 was more accurate than the best model of scenario S1, which is NNM. In addition, considering both scenarios, the NNM was selected as the best prediction model in seven cases. As mentioned before, the VDMs with the * superscript are the models that had a p-value less than 0.5 and will not be considered in our analysis. In Tables 41-42, we used the term "NS" for these models, which stands for Not Satisfactory.

For Windows, the best model from scenario S1, which is NNM ($\Delta AE_j^G = 0$), is 1.8% more accurate than the one from scenario S2 (the model with smallest AE in scenario S2, $\Delta AE_j^G = 1.836$). For Mac, the best model is also NNM by having 19.59% smaller average prediction error (AE) than the best model from scenario S2. For IOS, Linux, IE, Safari, and Chrome the stories are like what happened for Windows and Mac by NNM (from S1) as being the best model, which comes up with 1.1%, 1.9%, 3.5%, 3.9%, and 9.3% smaller prediction errors than the best models from scenario S2. For Firefox, the model with smallest AE ($\Delta AE_j^G = 0$) belongs to scenario S2 by having 1.6% smaller AE than the best model from scenario S1, which is NNM ($\Delta AE_j^G \approx 1.6$). Overall, scenario S1 provides more accurate results in seven cases (out of eight cases) for the number of future exploited vulnerabilities. In the only case that the best model from scenario S2 provided most accurate predictions, the performance of the best model from scenario S1 was only 1.6% worse.

To evaluate the performance of our approach among VDMs, we also need to compare the results we found only from VDMs. Considering only VDMs, in terms of

prediction accuracy (AE and HH), out of eight software we analyzed, scenario S1 led to most accurate results in only two cases. In other words, for Mac, and IE, the best VDM from S1 had higher accuracy than the best VDM from scenario S2 by having 18.8%, and 1.6% smaller prediction errors, respectively. However, the VDMs from scenario S1 were less than 2.2% different in prediction error in three cases compared to the best VDM from scenario S2. The error differences for Windows, IOS, and Linux are 2.2%, 1.6%, and 0.3%, respectively. Only for Safari, Firefox, and Chrome this difference is high and the best VDM from scenario S2 outperformed the best VDM from scenario S1 by having 16.2%, 15.7%, and 26% smaller prediction error, respectively. Overall, comparing only VDMs, scenario S1 was able to perform better than or as well as scenario S2 (with less than 2.2% error difference) in five cases.

Another important factor, which plays a role in model selection is the tendency of a given model to overestimate or underestimate the results. In this research, we provided the average bias values (AB) as well as the visual fluctuation trend of normalized prediction errors (Figure 16 and Figure 17).

Now, for each software, we compare the best overall model and the models of similar prediction power (those with $\Delta AE_j^G \leq 2$), in terms of average bias. For a given software, if there are multiple models that satisfy the mentioned condition, we consider the model with lowest AB. There are five software, which are qualified for this condition (i.e. Mac, IOS, Linux, IE, and Firefox). For Linux, IE, and Firefox, the absolute value of AB for the best overall model was smaller than the other candidates with $\Delta AE_j^G \leq 2$ by 2.1%, 3.9%, and 1.9%, respectively. This For Mac and IOS, the best overall model has higher absolute bias by 1.8%, and 0.1% difference, respectively.

However, in terms of bias, the final decision is depend on researcher's priorities to select the best model.

8.6.1 Summary of Results

In terms of prediction accuracy (AE and HH), considering the OSs and web browsers (eight cases), our presented approach led to more accurate results in seven cases. Out of those cases, the NNM provided the best model in all the cases. Comparing only VDMs, in terms of prediction accuracy, scenario S1 was able to perform better than or as well as scenario S2 (with less than 2.2% error difference) in five cases.

Please remember that all the conclusions we draw from this research and their validity are limited by our database uncertainties.

8.7 Discussion

We believe that the NNM's better execution contrasted with VDMs originates from the capacity of the NNM in foreseeing the nonlinearity nature of the vulnerability discovery process as a time series. Moreover, a common assumption in most VDMs is the pure S-shaped curve for vulnerability discovery process or considering a discovery function with a monotonic disclosure rate with constant total number of vulnerabilities. However, in practice, the vulnerability discovery process of a given software may have several linear and saturation phases as the total number of vulnerabilities may change as the result of introducing newer software versions. Furthermore, VDMs and traditional time-series functions only have one set of parameters for estimation. NNMs due to having multilayer perceptron structure, having various neurons per layer, and

utilizing diverse arrangement of parameters per neuron yield a structure with higher complexity for prediction.

In terms of overall magnitude of bias (i.e., absolute value of AB), out of the seven cases that scenario S1 performed better, the best model from scenario S1 outperformed the best VDMs from scenario S2 (those with $\Delta AE_j^G \leq 2$) in five cases.

We believe that, in equivalent precision conditions, in terms of bias, the final decision is up to the specialist to pick the best model dependent on his/her priorities. Be that as it may, from a security perspective, it is better to pick a model, which gives more conservative results, in terms of prediction accuracy. In the current study, out of the seven NNMs that were chosen as the best models, the AB value in three cases (Windows, IOS, and Chrome) is negative. In other words, in these cases, the predictor underestimated the results. It can also be easily inferred from Figure 16 and Figure 17, where for Windows, IOS, and Chrome most of the prediction points associated with the NNMs are located under the X=0 axis. For rest of the cases, the best overall model has come up with positive ABs or conservative results.

8.8 Limitations

There are a few limitations to our work that prevents us from expanding our conclusions in a more generalized manner. Like previous chapters, there are some common limitations, as well. One of which is using announced published date of vulnerabilities as their discovery date (cf. Section 4.7). Another limitation is associated with the uncertainties of vulnerability databases and the way in which we combined all vulnerabilities reported for all versions of a given software to have sufficient data for

training the models. However this limitation does not only apply to our research and was employed by other researches, as well (cf. Section 4.7).

Another limitation is with regard to the availability of public information for exploits. Many vendors and public repositories, with good reason, may not publish information on exploits as that is likely to increase the security risks for the end users of those systems. Responsible hackers are also more likely to not publish their exploits in public fora, as they can report them to the vendors directly. Malicious hackers are more likely to attempt to monetize their discoveries via dark web fora. Hence the predictions we make of publicly known exploits are likely to be underestimates of the true number of all vulnerabilities with exploits. Nevertheless, the approach we describe in this dissertation can be used by vendors and organization who have more information about exploits that they cannot share publicly to calibrate their predictions.

8.9 Summary

In this chapter, we evaluated the capability of all vulnerabilities associated with a software in predicting the number exploited ones. We compared two scenarios: S1 (use of all vulnerabilities) and S2 (use only of exploited vulnerabilities). We used eight common vulnerability discovery models (VDMs) for both scenarios as well as a non-linear neural network model (NNM) for the first scenario. Due to insufficient number of exploited vulnerabilities, it was not conceivable to use NNM for the second scenario. We used the aforementioned models for predicting the total number of future vulnerabilities over a prediction period of three years. The mentioned models were applied to vulnerability data associated with four well-known OSs and four well-known web browsers. We evaluated the models in terms of prediction accuracy and prediction

bias. The results showed that, out of eight software we analyzed, the first scenario led to more accurate results in seven cases. Moreover, out of these seven cases, the NNM was chose as the best model in all the cases. Comparing only VDMs, in terms of prediction accuracy, the first scenario was able to acceptably approximate the results from the second scenario in five cases (by performing better in two cases and providing less than 2.2% error difference in three cases). This is good since, we do not always have access to exploited vulnerability data, which are scarce, and need to predict their report time based on other publicly accessible information.

Chapter 9: Proposed Future Work and Summary of Completed Work

9.1 Introduction

Organizations face the issue of how to best allocate their security resources. Thus, they need an accurate method for assessing how many new vulnerabilities will be reported for the software they use in a given time period. Vulnerabilities reports of software systems are widely used by security researchers for security analysis (e.g. software reliability analysis). Researchers have used data from various vulnerability databases to study trends of discovery of new vulnerabilities, proposed various models for fitting the discovery times and predicting when new vulnerabilities may be discovered for a given product. Estimating the discovery times for new vulnerabilities is useful both for vendors of these products as well as the end-users as it can help them with their resource allocation strategies over time.

This chapter concludes this dissertation with a summary of the research questions and contributions (9.2), a summary of published work (9.3), and areas of future work (9.4).

9.2 Summary of the research questions and contributions

9.2.1 Summary of dissertation and research questions

Among the research conducted on vulnerability modeling, few studies have tried to provide a guideline about which model should be used in a given situation. In addition, to the best of our knowledge, there is no research focusing on modeling

exploited vulnerabilities. The only efforts in this area is probabilistic examination of intrusions by [52], [53]. Lack of data is a big barrier on the way to modeling exploited vulnerabilities using current VDMs or the machine learning techniques, which require considerable amount of data for satisfactory training. In other words, assuming the vulnerability data for a software is given, the research questions are the following:

RQ1: What models are more accurate for vulnerability discovery process modeling? Should all models be applied every time a new dataset is provided?

RQ2: Is there any feature in the vulnerability data could be used as a feature for identifying the most appropriate models for that dataset?

RQ3: Can we predict disclosure of exploited vulnerabilities having few number of data points? Is there any way we could apply machine learning algorithms to predict exploited vulnerabilities?

RQ4: Is the any link between discovery pattern of all vulnerabilities associated with a given software and its exploited vulnerabilities?

9.2.2 Summary of contributions

The goal of this dissertation is to propose a guideline to characterize the vulnerability discovery process using several common software reliability/vulnerability discovery models, also known as Software Reliability Models (SRMs)/Vulnerability Discovery Models (VDMs) a commonly used machine learning technique called Neural Networks. The proposed guideline covers the different aspects of vulnerability modeling including curve fitting and prediction. The data used in this research has been collected from six different public repositories of vulnerability data

sources including the National Vulnerability Database (NVD)[15], the Common Vulnerabilities and Exposures (CVE) database[16], the CVE Details data source[17], the Security database[18], the SecurityFocus data source[19], and the CXSecurity database[20]. The contributions of this dissertation are as follows:

- ✓ A new guideline to characterize the vulnerability discovery process has been presented using eight common SRMs/VDMs. The proposed guideline covers vulnerability discovery modeling curve fitting and prediction.
- ✓ Two strategies to improve curve fitting and prediction accuracy have been considered.
- ✓ The effect of employing a data manipulation technique (i.e. clustering) on improving the curve fitting and prediction capabilities of the current SRMs/VDMs has been investigated.
- ✓ The analysis has been implemented to two groups: operating systems (OSs), web browsers, and vendors. More specifically, we selected four OSs (Windows, Mac, IOS, and Linux), four web browsers (Internet Explorer, Safari, Firefox, and Chrome).
- ✓ We discussed the effect of another data manipulation technique (vulnerability grouping) on prediction capabilities of the VDMs

[15] https://nvd.nist.gov
[16] https://cve.mitre.org
[17] https://cvedetails.com/
[18] https://www.security-database.com/
[19] http://www.securityfocus.com/
[20] https://securityfocus.com/

- ✓ Our proposed guideline was expanded by applying the mentioned models to only the vulnerabilities in NVD that have been exploited since it is necessary for vendors to identify the vulnerabilities at risk of being exploited, and to find those with the potential of having rapidly an exploit.
- ✓ We presented a new modeling approach using neural networks and evaluate its prediction capability versus VDMs. The proposed approach provides higher curve fitting and prediction capabilities than current SRMs/VDMs.
- ✓ A new approach was introduced for modeling and predicting the total number of publicly-known exploited vulnerabilities using all publicly-known vulnerabilities reported for a given software. The proposed approach has higher curve fitting and prediction capabilities than current SRMs/VDMs.

This dissertation would contribute to the 'science of software reliability analysis' and present a new guideline for vulnerability risk assessment that could be integrated as part of security tools, such as Security Information and Event Management (SIEM) systems.

9.3 Summary of Published Work

9.3.1 Published work

A conference paper related to the research presented in this manuscript has been published. The paper, titled *Cluster-based Vulnerability Assessment Applied on*

Operating Systems was presented at the *Dependable Computing Conference (EDCC)* in September 2017 [58], investigated how cluster-based approach can improve the prediction capability of the NHPP Power-Law model in vulnerability discovery modeling of operating systems. This paper was elected as the distinguished paper of the conference.

One journal paper related to the research presented in Chapter 5 of this dissertation was published in September 2018. The paper, titled *Cluster-based Vulnerability Assessment of Operating Systems and Web Browsers*, published in *Computing Journal*, is an extend version of [83].

A conference paper, not included in this dissertation research, was presented at the *Dependable Computing Conference (EDCC)* in September 2017. The paper, titled *Application of Routine Activity Theory to Cyber Intrusion Location and Time* [84], explored the applicability of criminological theories to cybercriminals in an attempt to learn more about attacker behavior. The daily patterns of attack attempts on a network, recorded over a period of four years, were examined to identify periods of higher intrusion volume.

9.3.2 Additional completed work

A conference paper, titled *An Empirical Comparison of Grouping Strategies of Vulnerabilities for Modeling Vulnerability Discovery Processes*, will be submitted in April 2019 to the *Dependable Computing Conference (EDCC) 2019*. This paper corresponds to Chapter 6 of this thesis.

One journal paper, titled *Vulnerability Prediction Capability: A Comparison between Vulnerability Discovery Models and Neural Network Models*, will be

submitted in Summer 2019 to the *Computers & Security journal*. This paper corresponds to Chapter 7 of this dissertation.

One journal paper related to the research presented in Chapter 6 of this dissertation will be submitted in Summer 2019. The paper, titled *Predicting Exploited Vulnerabilities,* submitted to *IEEE Transactions on Dependable and Secure Computing,* corresponds to chapter 8 of this dissertation.

9.4 Future Work

It is necessary for vendors to identify the vulnerabilities and their detection pattern. Specifically, the vulnerabilities at risk of being exploited, and find those with the potential of having rapidly an exploit. We showed that we can improve the accuracy for predicting future vulnerabilities whether is it exploited or not.

Future work could be focused on exploring other nonlinear model structures using machine learning algorithms, which are capable of catching nonlinear nature of vulnerability data. Among them are Recurrent Neural Network (RNN) models, used for prediction time series, which may perform superior to NNMs at modeling dependencies between two points in a sequence. Generally, in NNMs, we have to choose the length of the input (number of inputs) beforehand. Then, it is not possible to learn functions that depends on the inputs that happened a long time ago. This problem could be solved by having an RNN, which can theoretically store information from arbitrarily long time ago.

Appendices

Appendix A: Clustering Tables

Operating Systems

Table 43: NUMBER OF VULNERABILITIES PER OS

OS	Windows	Mac	IOS	Linux
# Vulnerabilities	3434	2908	698	5812
# Labelled Vulnerabilities	2974 (86.6%)	2513 (86.4%)	643 (92.1%)	4533 (78%)
# Non-labelled Vulnerabilities	460 (13.4%)	395 (13.6%)	55 (7.9%)	1279 (22%)

Table 44: NUMBER OF VULNERABILITIES PER TYPE AND OS

Keywords	Windows	Mac	IOS	Linux
Denial of Service	900 (30.26%)	1214 (48.31%)	517 (80.40%)	2442 (53.87%)
Execute Code	1244 (41.83%)	1445 (57.50%)	71 (11.04%)	969 (21.38%)
Overflow	710 (23.87%)	1041 (41.43%)	64 (9.95%)	1211 (26.72%)
SQL Injection	7 (0.23%)	3 (0.11%)	0	13 (0.29%)
Obtain Information	387 (13.01%)	325 (12.93%)	27 (4.20%)	601 (13.26%)
Gain Privileges	579 (19.47%)	186 (7.40%)	14 (2.18%)	431 (9.51%)
Bypass Restriction or Similar	248 (8.34%)	258 (10.27%)	49 (7.62%)	340 (7.50%)
Directory Traversal	33 (1.11%)	19 (0.76%)	2 (0.31%)	56 (1.23%)
Cross Site Scripting	70 (2.35%)	58 (2.31%)	11 (1.71%)	86 (1.90%)
Http Response Splitting	0	2 (0.08%)	0	6 (0.13%)
CSRF	3 (0.10%)	3 (0.12%)	3 (0.47%)	13 (0.29%)
Memory Corruption	362 (12.17%)	758 (30.16%)	8 (1.24%)	278 (6.13%)

Table 45: NUMBER OF VULNERABILITIES PER TYPE, CLUSTER (WINDOWS)

Keywords	Windows					
	1	***2***	***3***	***4***	***5***	*6*
Denial of Service	0	1 (0.2%)	253 (88.8%)	537 (100%)	68 (16.3%)	41 (7.1%)
Execute Code	0	2 (0.4%)	262 (91.9%)	0	400 (95.7%)	580 (100%)
Overflow	62 (8.9%)	1 (0.2%)	170 (59.6%)	59 (11%)	418 (100%)	0
SQL Injection	2 (0.3%)	0	0	0	0	5 (0.9%)
Obtain Information	0	376 (81.9%)	1 (0.3%)	3 (0.6%)	0	7 (1.2%)
Gain Privileges	517 (74.4%)	10 (2.2%)	11 (3.9%)	20 (3.7%)	2 (0.5%)	19 (3.3%)
Bypass Restriction or Similar	183 (26.3%)	45 (9.8%)	1 (0.3%)	3 (0.6%)	1 (0.2%)	15 (2.6%)
Directory Traversal	1 (0.1%)	23 (5%)	0	1 (0.2%)	0	8 (1.4%)
Cross Site Scripting	3 (0.4%)	61 (13.3%)	0	0	0	6 (1%)
Http Response Splitting	0	0	0	0	0	0
CSRF	2 (0.3%)	1 (0.2%)	0	0	0	0
Memory Corruption	15 (2.2%)	1 (0.2%)	285 (100%)	11 (2%)	0	50 (8.6%)
# Vulnerabilities	695	459	285	537	418	580

Table 46: NUMBER OF VULNERABILITIES PER TYPE, CLUSTER (MAC)

Keywords	Mac					
	1	***2***	***3***	***4***	***5***	*6*
Denial of Service	624 (90.7%)	18 (14.6%)	36 (11.5%)	210 (50.2%)	322 (34.4%)	4 (11.4%)
Execute Code	615 (89.4%)	2 (1.6%)	3 (1%)	377 (90.2%)	447 (47.7%)	1 (2.9%)
Overflow	574 (83.4%)	0	14 (4.5%)	418 (100%)	0	35 (100%)
SQL Injection	0	0	0	0	3 (0.3%)	0
Obtain Information	9 (1.3%)	3 (2.4%)	312 (100%)	0	1 (0.1%)	0
Gain Privileges	40 (5.8%)	123 (100%)	0	1 (0.2%)	0	22 (62.9%)
Bypass Restriction or Similar	0	5 (4.1%)	42 (13.5%)	0	208 (22.2%)	3 (8.6%)
Directory Traversal	0	0	0	0	19 (2%)	0
Cross Site Scripting	0	0	1 (0.3%)	0	57 (6.1%)	0
Http Response Splitting	0	0	0	0	2 (0.2%)	0
CSRF	0	0	0	0	3 (0.3%)	0
Memory Corruption	688 (100%)	5 (4.1%)	1 (0.3%)	0	63 (6.7%)	1 (2.9%)
# Vulnerabilities	688	123	312	418	937	35

Table 47: NUMBER OF VULNERABILITIES PER TYPE, CLUSTER (IOS)

Keywords	IOS						
	1	***2***	***3***	***4***	***5***	*6*	***7***
Denial of Service	456 (100%)	11 (39.3%)	0	33 (94.3%)	8 (30.8%)	0	9 (23.1%)
Execute Code	0	28 (100%)	6 (46.1%)	0	0	3 (6.2%)	34 (87.2%)
Overflow	0	28 (100%)	0	35 (100%)	0	1 (2.1%)	0
SQL Injection	0	0	0	0	0	0	0
Obtain Information	0	0	0	0	26 (100%)	1 (2.1%)	0
Gain Privileges	0	0	0	1 (2.9%)	0	0	13 (33.3%)
Bypass Restriction or Similar	2 (0.4%)	0	0	0	0	47 (97.9%)	0
Directory Traversal	1 (0.2%)	0	0	0	0	1 (2.1%)	0
Cross Site Scripting	0	0	11 (84.6%)	0	0	0	0
Http Response Splitting	0	0	0	0	0	0	0
CSRF	0	0	0	0	0	0	3 (7.7%)
Memory Corruption	5 (1%)	0	0	2 (5.7%)	1 (3.8%)	0	0
# Vulnerabilities	456	28	13	35	26	48	39

Table 48: NUMBER OF VULNERABILITIES PER TYPE, CLUSTER (LINUX)

Keywords	Linux						
	1	***2***	***3***	***4***	***5***	*6*	***7***
Denial of Service	1579 (100%)	1 (1.8%)	722 (96%)	93 (87.7%)	13 (3.9%)	0	34 (6.2%)
Execute Code	68 (4.3%)	6 (11.1%)	213 (28.3%)	45 (42.4%)	23 (7%)	608 (52.1%)	6 (1.1%)
Overflow	0	0	752 (100%)	0	6 (1.8%)	433 (37.1%)	20 (3.7%)
SQL Injection	2 (0.1%)	0	0	0	2 (0.6%)	9 (0.8%)	0
Obtain Information	0	0	24 (3.2%)	2 (1.9%)	28 (8.5%)	0	547 (100%)
Gain Privileges	82 (5.2%)	1 (1.8%)	41 (5.4%)	10 (9.4%)	9 (2.7%)	275 (23.6%)	13 (2.4%)
Bypass Restriction or Similar	0	4 (7.4%)	6 (0.8%)	0	329 (100%)	1 (0.1%)	0
Directory Traversal	0	54 (100%)	1 (0.1%)	0	1 (0.3%)	0	0
Cross Site Scripting	0	0	0	0	3 (0.9%)	81 (6.9%)	2 (0.4%)
Http Response Splitting	0	0	0	0	0	6 (0.5%)	0
CSRF	0	0	0	0	4 (1.2%)	9 (0.8%)	0
Memory Corruption	0	0	162 (21.5%)	106 (100%)	2 (0.6%)	6 (0.5%)	2 (0.4%)
# Vulnerabilities	1579	54	752	106	329	1166	547

Table 49: CLUSTER COMPOSITION FOR OSS

OS	Cluster Number	Prevalent Keywords	Cluster Name
Windows	***1***	**G**ain privileges	G
	2	**O**btain Information	O
	3	**D**oS, **E**xecute code, **M**emory corruption	DEM
	4	**D**oS	D
	5	**E**xecute code, **O**verflow	EO
	6	**E**xecute code	E
Mac	***1***	**D**oS, **E**xecute code, **O**verflow, **M**emory corruption	DEOM
	2	**G**ain privileges	G
	3	**O**btain Information	O
	4	**E**xecute code, **O**verflow	EO
	5	**E**xecute code (47.7%)	E
	6	**O**verflow, **G**ain privileges	OG
IOS	***1***	**D**oS	D
	2	**E**xecute code, **O**verflow	EO
	3	**C**ross site scripting	C
	4	**D**oS, **O**verflow	DO
	5	**O**btain Information	O
	6	**B**ypass a restriction	B
	7	**E**xecute code	E
Linux	***1***	**D**oS	D
	2	**D**irectory **T**raversal	DT
	3	**D**oS, **O**verflow	DO
	4	**D**oS, **M**emory corruption	DM
	5	**B**ypass a restriction	B
	6	**E**xecute code (52.1%)	E
	7	**O**btain Information	O

Web Browsers

Table 50: NUMBER OF VULNERABILITIES PER WEB BROWSER

Web Browser	Explorer	Safari	Firefox	Chrome
# Vulnerabilities	1862	994	1784	1906
# Labelled Vulnerabilities	1601 (86.0%)	886 (89.1%)	1375 (77.1%)	1563 (82.0%)
# Non-labelled Vulnerabilities	261 (14.0%)	108 (10.9%)	409 (22.9%)	343 (18.0%)

Table 51: NUMBER OF VULNERABILITIES PER TYPE AND WEB BROWSER

Keywords	Explorer	Safari	Firefox	Chrome
Denial of Service	757 (47.28%)	649 (73.25%)	606 (44.07%)	993 (63.53%)
Execute Code	1197 (74.77%)	627 (70.77%)	724 (52.65%)	335 (21.43%)
Overflow	739 (46.16%)	453 (51.13%)	370 (26.91%)	473 (30.26%)
SQL Injection	0 (0%)	0 (0%)	0 (0%)	0 (0%)
Obtain Information	145 (9.06%)	97 (10.95%)	163 (11.85%)	102 (6.53%)
Gain Privileges	38 (2.37%)	3 (0.34%)	40 (2.91%)	9 (0.58%)
Bypass Restriction or Similar	116 (7.24%)	64 (7.22%)	188 (13.67%)	170 (10.88%)
Directory Traversal	4 (0.25%)	3 (0.34%)	9 (0.65%)	8 (0.51%)
Cross Site Scripting	48 (3.00%)	70 (7.90%)	125 (9.09%)	63 (4.03%)
Http Response Splitting	1 (0.06%)	0 (0%)	1 (0.07%)	0 (0%)
CSRF	0 (0%)	2 (0.23%)	9 (0.65%)	3 (0.19%)
Memory Corruption	885 (55.28%)	508 (57.34%)	388 (28.22%)	207 (13.24%)

Table 52: NUMBER OF VULNERABILITIES PER TYPE, CLUSTER (INTERNET EXPLORER)

Keywords	Internet Explorer				
	1	***2***	***3***	***4***	***5***
Denial of Service	644 (100%)	0	0	9 (2.5%)	104 (40.9%)
Execute Code	641 (99.5%)	2 (1.6%)	200 (89.7%)	354 (100%)	0
Overflow	502 (77.9%)	1 (0.8%)	223 (100%)	0	13 (5.1%)
SQL Injection	0	0	0	0	0
Obtain Information	0	26 (20.6%)	0	3 (0.8%)	116 (45.7%)
Gain Privileges	0	1 (0.8%)	0	5 (1.4%)	32 (12.6%)
Bypass Restriction or Similar	1 (0.1%)	102 (80.9%)	0	13 (3.7%)	0
Directory Traversal	0	1 (0.8%)	0	1 (0.3%)	2 (0.8%)
Cross Site Scripting	0	41 (32.5%)	0	3 (0.8%)	4 (1.6%)
Http Response Splitting	0	0	0	0	1 (0.4%)
CSRF	0	0	0	0	0
Memory Corruption	636 (98.8%)	0	149 (66.8%)	98 (27.7%)	2 (0.8%)
# Vulnerabilities	644	126	223	354	254

Table 53: NUMBER OF VULNERABILITIES PER TYPE, CLUSTER (SAFARI)

Keywords	Safari		
	1	***2***	***3***
Denial of Service	0	143 (79.9%)	506 (99.8%)
Execute Code	2 (1%)	123 (68.7%)	502 (99%)
Overflow	1 (0.5%)	34 (19%)	418 (82.4%)
SQL Injection	0	0	0
Obtain Information	94 (47%)	1 (0.6%)	2 (0.4%)
Gain Privileges	3 (1.5%)	0	0
Bypass Restriction or Similar	63 (31.5%)	1 (0.6%)	0
Directory Traversal	2 (1%)	1 (0.6%)	0
Cross Site Scripting	69 (34.5%)	1 (0.6%)	0
Http Response Splitting	0	0	0
CSRF	2 (1%)	0	0
Memory Corruption	0	1 (0.6%)	507 (100%)
# Vulnerabilities	200	179	507

Table 54: NUMBER OF VULNERABILITIES PER TYPE, CLUSTER (FIREFOX)

Keywords	Firefox				
	1	***2***	***3***	***4***	***5***
Denial of Service	100 (46.1%)	389 (99.2%)	0	113 (76.3%)	4 (2.6%)
Execute Code	138 (63.6%)	280 (71.4%)	208 (44.5%)	95 (64.2%)	3 (2%)
Overflow	217 (100%)	0	0	148 (100%)	5 (3.3%)
SQL Injection	0	0	0	0	0
Obtain Information	10 (4.6%)	2 (0.5%)	0	0	151 (100%)
Gain Privileges	4 (1.8%)	1 (0.2%)	31 (6.6%)	1 (0.7%)	3 (2%)
Bypass Restriction or Similar	1 (0.5%)	1 (0.2%)	152 (32.5%)	2 (1.3%)	32 (21.2%)
Directory Traversal	0	1 (0.2%)	7 (1.5%)	0	1 (0.7%)
Cross Site Scripting	0	0	122 (26.1%)	0	3 (2%)
Http Response Splitting	0	0	1 (0.2%)	0	0
CSRF	0	0	8 (1.7%)	0	1 (0.7%)
Memory Corruption	0	238 (60.7%)	2 (0.4%)	148 (100%)	0
# Vulnerabilities	217	392	467	148	151

Table 55: NUMBER OF VULNERABILITIES PER TYPE, CLUSTER (CHROME)

Keywords	Chrome				
	1	***2***	***3***	***4***	***5***
Denial of Service	868 (100%)	1 (1%)	2 (0.9%)	1 (1.8%)	121 (36.8%)
Execute Code	0	2 (2%)	2 (0.9%)	5 (9.1%)	326 (99%)
Overflow	263 (30.3%)	3 (3%)	38 (17.9%)	0	169 (51.4%)
SQL Injection	0	0	0	0	0
Obtain Information	0	99 (100%)	0	0	3 (0.9%)
Gain Privileges	0	0	9 (4.2%)	0	0
Bypass Restriction or Similar	1 (0.1%)	11 (11.1%)	154 (72.6%)	4 (7.3%)	0
Directory Traversal	0	0	8 (3.8%)	0	0
Cross Site Scripting	0	8 (8.1%)	0	55 (100%)	0
Http Response Splitting	0	0	0	0	0
CSRF	0	0	3 (1.4%)	0	0
Memory Corruption	70 (8.1%)	1 (1%)	1 (0.5%)	0	135 (41%)
# Vulnerabilities	868	99	212	55	329

Table 56: CLUSTER COMPOSITION FOR WEB BROWSERS

Web Browser	Cluster Number	Prevalent Keywords	Cluster Name
Internet Explorer	***1***	**D**oS, **E**xecute code, **O**verflow, **M**emory corruption	DEOM
	2	**B**ypass a restriction	B
	3	**E**xecute code, **O**verflow, **M**emory corruption	EOM
	4	**E**xecute code	E
	5	**O**btain Information (45.7%)	O
Safari	***1***	**O**btain Information (47%)	O
	2	**D**oS, **E**xecute code	DE
	3	**D**oS, **E**xecute code, **O**verflow, **M**emory corruption	DEOM
Firefox	***1***	**E**xecute code, **O**verflow	EO
	2	**D**oS, **E**xecute code, **M**emory corruption	DEM
	3	**E**xecute code (44.5%)	E
	4	**D**oS, **E**xecute code, **O**verflow, **M**emory corruption	DEOM
	5	**O**btain Information	O
Chrome	***1***	**D**oS	D
	2	**O**btain Information	O
	3	**B**ypass a restriction	B
	4	**C**ross Site Scripting	C
	5	**E**xecute code	E

Bibliography

[1] H. Okamura, M. Tokuzane, and T. Dohi, “Optimal Security Patch Release Timing under Non-homogeneous Vulnerability-Discovery Processes,” presented at the 20th International Symposium on Software Reliability Engineering, 2009, pp. 120–128.

[2] A. A. Y. Mussa, “Quantifying the security risk of discovering and exploiting software vulnerabilities,” Ph.D., Colorado State University, United States -- Colorado, 2016.

[3] M. R. Lyu, Ed., *Handbook of software reliability engineering*. Los Alamitos, Calif. : New York: IEEE Computer Society Press ; McGraw Hill, 1996.

[4] P. E. Verissimo *et al.*, “Intrusion-tolerant middleware: the road to automatic security,” *IEEE Secur. Priv. Mag.*, vol. 4, no. 4, pp. 54–62, Jul. 2006.

[5] J. A. Ozment, “Vulnerability discovery & software security,” University of Cambridge, 2007.

[6] E. Rescorla, “Security holes... Who cares?,” presented at the USENIX Security, 2003.

[7] E. Rescorla, “Is finding security holes a good idea?,” *IEEE Secur. Priv. Mag.*, vol. 3, no. 1, pp. 14–19, Jan. 2005.

[8] O. H. Alhazmi and Y. K. Malaiya, “Quantitative vulnerability assessment of systems software,” 2005, pp. 615–620.

[9] H. Okamura, M. Tokuzane, and T. Dohi, “Quantitative Security Evaluation for Software System from Vulnerability Database,” *J. Softw. Eng. Appl.*, vol. 06, no. 04, p. 15, Apr. 2013.

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

[11] 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*, New York, NY, USA, 2006, pp. 131–138.

[12] 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*, Springer, Boston, MA, 2010, pp. 79–106.

[13] S. Zhang, D. Caragea, and X. Ou, “An empirical study on using the national vulnerability database to predict software vulnerabilities,” presented at the International Conference on Database and Expert Systems Applications, 2011, pp. 217–231.

[14] “CVE - Common Vulnerabilities and Exposures (CVE).” [Online]. Available: https://cve.mitre.org/. [Accessed: 15-Jun-2017].

[15] N. Nagappan and T. Ball, “Use of relative code churn measures to predict system defect density,” in *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, 2005, pp. 284–292.

[16] L. Allodi and F. Massacci, “Comparing Vulnerability Severity and Exploits Using Case-Control Studies,” *ACM Trans Inf Syst Secur*, vol. 17, no. 1, pp. 1:1–1:20, Aug. 2014.

[17] S. Woo, O. Alhazmi, and Y. Malaiya, “Assessing Vulnerabilities in Apache and IIS HTTP Servers,” 2006, pp. 103–110.
[18] O. H. Alhazmi and Y. K. Malaiya, “Application of Vulnerability Discovery Models to Major Operating Systems,” *IEEE Trans. Reliab.*, vol. 57, no. 1, pp. 14–22, Mar. 2008.
[19] G. R. Hudson, “Program errors as a birth-and-death process,” System Development Corp., Report SP-3011, Dec. 1967.
[20] R. Anderson, “Security in open versus closed systems—the dance of Boltzmann, Coase and Moore,” Cambridge University, England, Technical report, 2002.
[21] O. H. Alhazmi and Y. K. Malaiya, “Modeling the vulnerability discovery process,” in *16th IEEE International Symposium on Software Reliability Engineering (ISSRE’05)*, 2005, pp. 10 pp. – 138.
[22] J. Kim, Y. K. Malaiya, and I. Ray, “Vulnerability Discovery in Multi-Version Software Systems,” in *10th IEEE High Assurance Systems Engineering Symposium, 2007. HASE ’07*, 2007, pp. 141–148.
[23] P. L. Li, M. Shaw, J. Herbsleb, B. Ray, and P. Santhanam, “Empirical Evaluation of Defect Projection Models for Widely-deployed Production Software Systems,” in *Proceedings of the 12th ACM SIGSOFT Twelfth International Symposium on Foundations of Software Engineering*, New York, NY, USA, 2004, pp. 263–272.
[24] F. Massacci and V. H. Nguyen, “An Empirical Methodology to Evaluate Vulnerability Discovery Models,” *IEEE Trans. Softw. Eng.*, vol. 40, no. 12, pp. 1147–1162, Dec. 2014.
[25] H. Joh and Y. K. Malaiya, “Modeling Skewness in Vulnerability Discovery: Modeling Skewness in Vulnerability Discovery,” *Qual. Reliab. Eng. Int.*, vol. 30, no. 8, pp. 1445–1459, Dec. 2014.
[26] J. Y. Kim, “Vulnerability discovery in multiple version software systems : open source and commercial software systems,” Thesis, Colorado State University. Libraries, 2007.
[27] A. Ozment and S. E. Schechter, “Milk or wine: does software security improve with age?,” presented at the 15th USENIX Security Symposium, 2006.
[28] K. Chan, D. Feng, P. Su, and C. . Nie, “Multicycle vulnerability discovery model for prediction,” *J. Softw.*, vol. 21, no. 9, pp. 2367–2375, 2010.
[29] 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*, New York, NY, USA, 2008, pp. 315–317.
[30] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, “Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities,” *IEEE Trans. Softw. Eng.*, vol. 37, no. 6, pp. 772–787, Nov. 2011.
[31] Y. Shin and L. Williams, “Can traditional fault prediction models be used for vulnerability prediction?,” *Empir. Softw. Eng.*, vol. 18, no. 1, pp. 25–59, Feb. 2013.
[32] V. H. Nguyen, S. Dashevskyi, and F. Massacci, “An automatic method for assessing the versions affected by a vulnerability,” *Empir. Softw. Eng.*, vol. 21, no. 6, pp. 2268–2297, Dec. 2016.

[33] M. Shahzad, M. Z. Shafiq, and A. X. Liu, “A Large Scale Exploratory Analysis of Software Vulnerability Life Cycles,” in *Proceedings of the 34th International Conference on Software Engineering*, Piscataway, NJ, USA, 2012, pp. 771–781.
[34] 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)*, 2011, pp. 10–16.
[35] M. A. McQueen, T. A. McQueen, W. F. Boyer, and M. R. Chaffin, “Empirical Estimates and Observations of 0Day Vulnerabilities,” in *2009 42nd Hawaii International Conference on System Sciences*, 2009, pp. 1–12.
[36] 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*, New York, NY, USA, 2010, pp. 105–114.
[37] C. Steve and R. A. Martin, “Vulnerability type distributions in CVE,” Mitre report, 2007.
[38] A. Vega, P. Bose, and A. Buyuktosunoglu, *Rugged Embedded Systems: Computing in Harsh Environments*. Morgan Kaufmann, 2016.
[39] M. Bernaschi, E. Gabrielli, and L. V. Mancini, “Remus: A Security-enhanced Operating System,” *ACM Trans Inf Syst Secur*, vol. 5, no. 1, pp. 36–61, Feb. 2002.
[40] R. A. Johnson and D. W. Wichern, *Applied multivariate statistical analysis*, 6th ed. Upper Saddle River, N.J: Pearson Prentice Hall, 2007.
[41] C. Sabottke, S. Octavian, and T. Dumitras, “Vulnerability Disclosure in the Age of Social Media: Exploiting Twitter for Predicting Real-World Exploits,” presented at the USENIX Security, 2015, vol. 15.
[42] A. Younis, Y. K. Malaiya, and I. Ray, “Assessing vulnerability exploitability risk using software properties,” *Softw. Qual. J.*, vol. 24, no. 1, pp. 159–202, Mar. 2016.
[43] K. Lee, J. Kim, K. H. Kwon, Y. Han, and S. Kim, “DDoS attack detection method using cluster analysis,” *Expert Syst. Appl.*, vol. 34, no. 3, pp. 1659–1665, Apr. 2008.
[44] S. Huang, H. Tang, M. Zhang, and J. Tian, “Text Clustering on National Vulnerability Database,” in *2010 Second International Conference on Computer Engineering and Applications*, 2010, vol. 2, pp. 295–299.
[45] M. Bozorgi, “A machine learning framework for classifying invulnerabilites and predicting exploitability,” University of California, San Diego, 2009.
[46] P. Mell, K. Scarfone, and S. Romanosky, “A complete guide to the common vulnerability scoring system version 2.0.” FIRST-Forum of Incident Response and Security Teams, 2007.
[47] 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*, New York, NY, USA, 2012, pp. 17–24.
[48] J. Zheng, H. Okamura, and T. Dohi, “Mean Time to Security Failure of VM-Based Intrusion Tolerant Systems,” in *2016 IEEE 36th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 2016, pp. 128–133.

[49] J. Zheng, H. Okamura, and T. Dohi, “Survivability Analysis of VM-Based Intrusion Tolerant Systems,” *IEICE Trans. Inf. Syst.*, vol. E98-D, no. 12, pp. 2082–2090, Dec. 2015.
[50] C. Luo, H. Okamura, and T. Dohi, “Optimal planning for open source software updates,” *Proc. Inst. Mech. Eng. Part O J. Risk Reliab.*, vol. 230, no. 1, pp. 44–53, Feb. 2016.
[51] A. M. Algarni and Y. K. Malaiya, “A consolidated approach for estimation of data security breach costs,” in *2016 2nd International Conference on Information Management (ICIM)*, 2016, pp. 26–39.
[52] B. B. Madan, K. Goševa-Popstojanova, K. Vaidyanathan, and K. S. Trivedi, “A method for modeling and quantifying the security attributes of intrusion tolerant systems,” *Perform. Eval.*, vol. 56, no. 1–4, pp. 167–186, 2004.
[53] H. K. Browne, W. A. Arbaugh, J. McHugh, and W. L. Fithen, “A trend analysis of exploitations,” in *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*, Oakland, CA, USA, 2001, pp. 214–229.
[54] A. A. Younis, H. Joh, and Y. Malaiya, “Modeling Learningless Vulnerability Discovery using a Folded Distribution,” in *Proceedings of the International Conference on Security and Management (SAM)*, 2011, pp. 617–623.
[55] L. Allodi, “The Heavy Tails of Vulnerability Exploitation,” in *Engineering Secure Software and Systems*, 2015, pp. 133–148.
[56] T. Y. Yang and L. Kuo, “Bayesian computation for the superposition of nonhomogeneous poisson processes,” *Can. J. Stat.*, vol. 27, no. 3, pp. 547–556, Sep. 1999.
[57] Scikit-learn developers, *sklearn.model_selection.StratifiedKFold*. 2017.
[58] Y. Movahedi, M. Cukier, A. Andongabo, and I. Gashi, “Cluster-based Vulnerability Assessment Applied to Operating Systems,” presented at the 13th European Dependable Computing Conference, Geneva, Switzerland, 2017.
[59] D. N. Gujarati and D. C. Porter, *Basic Econometrics*. McGraw-Hill Irwin, 2009.
[60] L. Mentaschi, G. Besio, F. Cassola, and A. Mazzino, “Problems in RMSE-based wave model validations,” *Ocean Model.*, vol. 72, pp. 53–58, Dec. 2013.
[61] S. R. Hanna, D. W. Heinold, A. P. I. H. and E. A. Dept, and E. R. & T. Inc, *Development and application of a simple method for evaluating air quality models*. American Petroleum Institute, 1985.
[62] Y. Movahedi, M. Cukier, A. Andongabo, and I. Gashi, “Cluster-based vulnerability assessment of operating systems and web browsers,” *Computing*, Sep. 2018.
[63] *SAS® Enterprise Miner™ 14.2: High-Performance Procedures*. Cary, NC: SAS Institute Inc., 2016.
[64] W. S. Sarle, “Cubic Clustering Criterion,” SAS Institution Inc., Cary, NC, SAS® Technical Report A-108, 1983.
[65] R. Tibshirani, G. Walther, and T. Hastie, “Estimating the number of clusters in a data set via the gap statistic,” *J. R. Stat. Soc. Ser. B Stat. Methodol.*, vol. 63, no. 2, pp. 411–423, Jan. 2001.
[66] K. P. Burnham and D. R. Anderson, “Multimodel Inference: Understanding AIC and BIC in Model Selection,” *Sociol. Methods Res.*, vol. 33, no. 2, pp. 261–304, Nov. 2004.

[67] Y. Roumani, J. K. Nwankpa, and Y. F. Roumani, “Time series modeling of vulnerabilities,” *Comput. Secur.*, vol. 51, pp. 32–40, Jun. 2015.
[68] L. Wang, Y. Zeng, and T. Chen, “Back propagation neural network with adaptive differential evolution algorithm for time series forecasting,” *Expert Syst. Appl.*, vol. 42, no. 2, pp. 855–863, Feb. 2015.
[69] N. R. Pokhrel, H. Rodrigo, and C. P. Tsokos, “Cybersecurity: Time Series Predictive Modeling of Vulnerabilities of Desktop Operating System Using Linear and Non-Linear Approach,” *J. Inf. Secur.*, vol. 08, no. 04, pp. 362–382, 2017.
[70] A. A. Adebiyi, A. O. Adewumi, and C. K. Ayo, “Comparison of ARIMA and Artificial Neural Networks Models for Stock Price Prediction,” *J. Appl. Math.*, vol. 2014, pp. 1–7, 2014.
[71] C. Bennett, R. A. Stewart, and C. D. Beal, “ANN-based residential water end-use demand forecasting model,” *Expert Syst. Appl.*, vol. 40, no. 4, pp. 1014–1023, Mar. 2013.
[72] N. Kourentzes, D. K. Barrow, and S. F. Crone, “Neural network ensemble operators for time series forecasting,” *Expert Syst. Appl.*, vol. 41, no. 9, pp. 4235–4244, Jul. 2014.
[73] A. Aslanargun, M. Mammadov, B. Yazici, and S. Yolacan, “Comparison of ARIMA, neural networks and hybrid models in time series: tourist arrival forecasting,” *J. Stat. Comput. Simul.*, vol. 77, no. 1, pp. 29–53, Jan. 2007.
[74] H. G. Hosseini, D. Luo, and K. J. Reynolds, “The comparison of different feed forward neural network architectures for ECG signal diagnosis,” *Med. Eng. Phys.*, vol. 28, no. 4, pp. 372–378, May 2006.
[75] D. N. Gujarati and D. C. Porter, *Basic Econometrics*. McGraw-Hill Irwin, 2009.
[76] R. May, G. Dandy, and H. Maier, “Review of input variable selection methods for artificial neural networks,” in *Artificial neural networks-methodological advances and biomedical applications*, InTech, 2011.
[77] S. Siami-Namini and A. S. Namin, “Forecasting Economics and Financial Time Series: ARIMA vs. LSTM,” *ArXiv Prepr. ArXiv180306386*, 2018.
[78] G. Zhang, B. Eddy Patuwo, and M. Y. Hu, “Forecasting with artificial neural networks:,” *Int. J. Forecast.*, vol. 14, no. 1, pp. 35–62, Mar. 1998.
[79] A. A. Younis and Y. K. Malaiya, “Comparing and Evaluating CVSS Base Metrics and Microsoft Rating System,” in *2015 IEEE International Conference on Software Quality, Reliability and Security*, 2015, pp. 252–261.
[80] R. Adhikari and R. K. Agrawal, “An introductory study on time series modeling and forecasting,” *ArXiv Prepr. ArXiv13026613*, 2013.
[81] C. N. Babu and B. E. Reddy, “A moving-average filter based hybrid ARIMA–ANN model for forecasting time series data,” *Appl. Soft Comput.*, vol. 23, pp. 27–38, Oct. 2014.
[82] A. Phinyomark, A. Nuidod, P. Phukpattaranont, and C. Limsakul, “Feature extraction and reduction of wavelet transform coefficients for EMG pattern classification,” *Elektron. Ir Elektrotechnika*, vol. 122, no. 6, pp. 27–32, 2012.
[83] Y. Movahedi, M. Cukier, A. Andongabo, and I. Gashi, “Cluster-based vulnerability assessment of operating systems and web browsers,” *Computing*, vol. 101, no. 2, pp. 139–160, Feb. 2019.

[84] K. Bock, S. Shannon, Y. Movahedi, and M. Cukier, “Application of Routine Activity Theory to Cyber Intrusion Location and Time,” in *2017 13th European Dependable Computing Conference (EDCC)*, 2017, pp. 139–146.