UNIVERSITY OF CALIFORNIA, SAN DIEGO

A Machine Learning Framework for Classifying Vulnerabilites and Predicting Exploitability

A thesis submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Computer Science

by

Mehran Bozorgi

Committee in charge:

Professor Lawrence Saul, Chair
Professor Stefan Savage
Professor Geoffrey Voelker

2009

The thesis of Mehran Bozorgi is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California, San Diego

2009

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

This work began as a course project, and I am grateful to my colleague, Justin Ma, for his work on machine learning algorithms. I would also like to thank Professor Hovav Shacham for his help and advice during the graduate security course at UCSD. I am also indebted to my committee members, Lawrence Saul, Stefan Savage, and Geoffrey Voelker, for their enormous amount of assistance, feedback, and encouragement during the development and completion of this work.

Finally, I am grateful to Ken J. Roff for his continuous support and patience throughout the duration of this work.

ABSTRACT OF THE THESIS

A Machine Learning Framework for Classifying Vulnerabilites and Predicting Exploitability

by

Mehran Bozorgi

Master of Science in Computer Science

University of California San Diego, 2009

Professor Lawrence Saul, Chair

Vulnerabilities are flaws in software that open avenues of attack for operations that are not initially intended to be performed. Exploits are means that attackers develop to take advantage of vulnerabilities. Many vulnerabilities are publicly documented on the web in databases such as OSVDB and CVE. Due to large number of vulnerabilities and limited resources available to software companies, it is critical to be able to measure the severity of vulnerabilities to schedule fixes for them accordingly. In this research, we introduce a machine learning framework for classifying vulnerabilities and predicting exploitability. We examine vulnerabilities available on the web, extract features from them, and build a classifier that predicts whether a given vulnerability will be exploited, and if so, how far in the future it will be exploited. Moreover, we introduce a new system for scoring the severity of vulnerabilities. We use Support Vector Machines (SVMs) to build our classifier due to the nature of the problem. We find that our classifier achieves high accuracy, is stable over time, and is extensible to other prediction scenarios.

Chapter 1

Introduction

Vulnerabilities are flaws in software that allow attackers to perform risky operations. The cause of vulnerabilities can range from simple programing bugs in an application to fundamental problems in the integration of an information system. Attackers can take advantage of these flaws or, in other words, "exploit" them to cause harm in the system. Many vulnerabilities are publicly documented on the web by various software companies, security vendors, or independent security organizations. The public documentation of vulnerabilities raises awareness and allows users to take necessary actions to protect their systems. It also allows security researchers to create new methods for preventing them from happening. The Open Source Vulnerability Database (OSVDB) and Common Vulnerabilities and Exposures (CVE) are two standard databases that identify and document these vulnerabilities.

It is important to quantify the severity of vulnerabilities for various reasons. First, it allows the end users of software to assess the magnitude of risk they take by using vulnerable software. Second, due to limited resources available to software companies, it allows them to prioritize fixes for their software depending on the severities of their various vulnerabilities. Finally, it allows security researches to focus on those vulnerabilities that have highest risk. Ranking vulnerabilities,

however, is not an easy task. Many security vendors, software companies, and independent security organizations have developed their own ranking system. These efforts have been independent and lack cohesion. The Computer Emergency Response Team (CERT) and Common Vulnerabilities Scoring System (CVSS) are two examples of independent rating systems. CERT only focuses on very narrow aspects of vulnerabilities such as their threat to Internet as a whole. CVSS uses different metrics to score vulnerabilities but is limited to those few metrics.

In this work, we present a new approach for classifying vulnerabilities based on machine learning. We present an overview of our vulnerability classification system, and an overview of our database and data flow. Next, we present our methodology for solving the problem using Support Vector Machines (SVMs) in a supervised learning setting. We discuss how we define and extract features from the data to use as input to our learning algorithm. We describe the "bag of words" model that is one of the major methods used in our feature extraction engine. We also describe, in detail, how our learning algorithm works.

Next, we discuss the design and implementation of our experiments. We explain the reasoning behind using category information as part of our feature extraction process. We discuss how we can predict the exploitability of vulnerabilities with a 90% success rate. We present the most influential features in our training and testing and present, in detail, their description, their degree of importance, and the number of vulnerabilities they affect. Moreover, we discuss how we adjust our classification process for an online setting in a week-to-week, month-to-month, and year-to-year basis. The online experiments show that our algorithm is stable over time. Moreover, we perform a "time-to-exploit" experiment. In this experiment, we predict whether a vulnerability that is expected to be exploited will be exploited in time t, where t can be any number of days. Our results show that we achieve high accuracy in this experiment over time. In addition, we present a methodological comparison of our classification model and CVSS. This comparison

shows our model predicts the exploitability of vulnerabilities significantly better. Then, we discuss the results of our experiments in detail. We introduce a new scheme for scoring vulnerabilities. The accuracy, efficiency, ease of generalization, and ease of integration are among the advantages of our system over the existing systems.

Finally, we discuss the future directions with this work. Using semi-supervised or unsupervised learning methods as well as feature enhancement methods by data source integration are among the approaches that can be taken in the future to further enhance the quality of our system. We conclude that the machine learning framework is suitable for solving this problem and has the potential to be the main industry practise for classification and scoring of vulnerabilities.

Chapter 2

Background: Survey of Past Work

In this chapter, we present an overview of what vulnerabilities and exploits are. We discuss the purpose and advantages of public documentation of vulnerabilities as well as the necessity for quantifying their severity. Finally, we point out the shortcomings of current scoring systems.

2.1 Software Vulnerabilities and Exploits

The term *vulnerability* refers to a flaw in an information system which allows an attacker to perform unwanted operations and to violate the integrity of the system. Software vulnerabilities result from different causes such as programing bugs, high-risk passwords, and viruses. A very simple programing bug can result in a severe vulnerability in a system. There are many different categories of vulnerabilities. These categories include memory safety violations such as buffer overflows, input validation errors such as SQL injection, race conditions such as symlink races, privilege-confusion bugs such as FTP bounce attacks, privilege escalation, and user interface failures such as user conditioning [20]. These flaws expose security risks and provide a means of attack to a system. Attackers can then perform unwanted operations that affect the integrity, availability, and confidentiality of the system. An example of an attack that affects availability is denial

of service.

An attack that is fully implemented is referred to as an *exploit*. An exploit can be a piece of software, a sequence of instructions, or a piece of data that causes unintended behavior. Exploits are possible because of the existence of vulnerabilities in a system. Exploits are usually categorized either by the way they interact with the vulnerable system or by the way they harm the system. For instance, some exploits are called "remote exploits" because they contact the vulnerable software over a network. "Arbitrary code execution", however, is an example of an exploit that is categorized based on its influence.

It is often the case that vulnerabilities are unknown until software is released to public and examined by computer users and experts. However, in some cases, software companies release their software while the existence of vulnerabilities is known. This fact is due to pressure and tight competition that these companies face in today's competitive market. Moreover, it helps detect vulnerabilities in their software early, while it might be difficult to catch them in testing stages. In both cases, these vulnerabilities can cause substantial problems ranging from damaging users' computers and exposing their data to causing significant costs for the companies and hurting their reputation. Online software has the advantage that it can be fixed remotely without users having to download a patch for the vulnerability. Desktop software does not benefit from this advantage. This fact creates additional sources of problem for users of desktop software since they may continue using vulnerable software for a very long time even though its patch and exploit are available.

2.2 Public Documentation of Vulnerabilities

Vulnerabilities are often publicly documented on the web. For example, the Common Vulnerabilities and Exposures (CVE) [2] is a database that identifies and

stores information about all publicly known vulnerabilities in commercial software. CVE assigns unique common identifiers to vulnerabilities that enable ease of reference and better organization. Other repositories that store vulnerabilities and provide additional details and analysis on them include the Open Source Vulnerability Database (OSVDB) [1], Secunia [3], Security Tracker [4], and Bugtraq [5]. Many companies in the software industry such as Microsoft, Mozilla, and Adobe publicly document information about security vulnerabilities in their software. It is widely believed that companies do not disclose information about all of the vulnerabilities that they are aware of. They choose based on their own criteria whether they will publicly announce each vulnerability or not. There are advantages and disadvantages to both full disclosure (public disclosure of all known vulnerabilities in the company's software) and limited disclosure (public disclosure of a subset of known vulnerabilities) as we describe next.

The benefit of full disclosure of all vulnerabilities is that users of the software have the ability to view all known security problems in the software. If a user does suffer from an exploit to a publicly disclosed vulnerability, the company has less liability to the user. At the same time, the company may come across as unreliable for having software vulnerabilities in the first place. In addition, full disclosure of vulnerabilities allows attackers to implement exploits for them earlier and easier than they could do so without knowing about the vulnerabilities. The benefit of limited disclosure of vulnerabilities is that the company appears to be more reliable. On the other hand, if a user suffers from an exploit to a vulnerability that the company was well aware of and was not publicly disclosed due to its limited disclosure policy, then the user could blame the company for hiding information that would have prevented the attack.

2.3 Rating and Quantifying the Severity of Vulnerabilities

The ability to rank the severity of vulnerabilities is critical for the computer industry. A software company would logically allocate its resources to fix the vulnerabilities of its software based on their severity and importance. Software packages like Microsoft Windows are often released while there are many known vulnerabilities in them. Due to limited resources such as head count, software companies prefer to exert their time and effort on those vulnerabilities that are more sever and have higher security risks than others. Ranking vulnerabilities is a difficult task because it is often hard to predict ahead of time how an attacker could leverage a vulnerability to perform malicious tasks, or what an attacker could do by exploiting a vulnerability until they actually do it. Therefore, the risk that different vulnerabilities expose is often unknown until they are exploited. Moreover, it is not easy to assess by the description of two vulnerabilities as to which one is more severe. It is very possible that a simple programing bug can lead to more harm than a major system flaw.

2.4 Shortcomings of Existing Scoring Systems

A number of commercial security vendors and non-commercial organizations have developed systems for ranking vulnerabilities. These efforts have been independent and lack cohesion. Moreover, they differ significantly on what they measure. In this section, we discuss two known scoring systems in the field: the vulnerability scoring system provided by the Computer Emergency Response Team (CERT) [19] and the Common Vulnerability Scoring System (CVSS) [6].

CERT generates a score ranging from 0 to 180 that is supposed to represent the severity of a vulnerability. This number is calculated from several but a very

limited number of factors. These factors include answers to the following questions [10]:

- Is information about the vulnerability widely available or known?
- Is the vulnerability being exploited in incidents reported to CERT or other incident response teams?
- Is the Internet infrastructure at risk because of this vulnerability?
- How many systems on Internet are at risk from this vulnerability?
- What is the impact of exploiting the vulnerability?
- How easy is it to exploit the vulnerability?
- What are the preconditions required to exploit the vulnerability?

Here we discuss a number of potential problems with this scoring system. First, the factors that determine the severity of a vulnerability are very limited. Only a few questions are asked and these questions are not a representative set. For example, the percentage of vulnerabilities that may put the Internet infrastructure at risk is very low, so it may not be a good distinguishing factor between vulnerabilities. Second, these questions are answered with approximate values based on the judgment of people answering the questions. Therefore, the scores may be significantly different from one site to another. Users of these scores cannot rely too heavily on them for prioritizing their vulnerabilities.

CVSS is the other scoring system that we examine here. In an attempt to implement a unified ranking system and provide a common language, CVSS has been developing an open standard for scoring vulnerabilities. CVSS uses three general criteria for scoring vulnerabilities. These general criteria are: base, temporal, and environmental metrics. Each general metric has subcategories. For example, exploitability, remediation level, and report confidence are the subcategories of the

temporal metric. CVSS uses these metrics to assign a score to each vulnerability, representing its severity. Although the effort to define and implement a unified scoring system is reasonable, it is not clear whether the scores generated by CVSS are close to reality. CVSS uses limited criteria to calculate the score of a vulnerability. Although, it allows its users to adjust the metrics to compute a customized score, there are only a few criteria for each metric to set. More specifically, according to CVSS 2.0 complete guide [15], the base, temporal, and environmental metrics are each only a function of 6, 3, and 3 variables respectively. The 6 variables for base metric are: access vector, access complexity, authentication, confidentiality impact, integrity impact, and availability impact. Each of these variables can only take 3 possible values. Although there is a history of why these specific variables are picked for each metric function, it is not clear why only these variables are important in scoring vulnerabilities, and nothing else. Note that these variables measure completely different quantities than those measured by CERT. Another source of ambiguity in CVSS is the formulas used for calculating scores such as exploitability score. According to "A Complete Guide to the Common Vulnerability Scoring System Version 2.0" [15], the formula for calculating the score of exploitability is the following:

$$Exploitability = 20 * AV * AC * Authentication \tag{2.1}$$

where AV stands for Access Vector and AC stands for Access Complexity.

It is not mentioned in the document how Equation 2.1 is derived and, for example, how the constant, 20, is computed. Another equation introduced in this guide under the umbrella of temporal metric is the following:

$$Exploitability = \begin{cases} 0.85 & \text{if the case of exploitability is unproven} \\ 0.9 & \text{if there is proof of concept for exploitability} \\ 0.95 & \text{if exploitability is functional} \\ 1.00 & \text{if exploitability is high} \\ 1.00 & \text{if the case of exploitability is not defined} \end{cases}$$

It is unclear what the relationship between the above equation and Equation 2.1 is. It is also unclear what the rationale behind the specific scores assigned to each case of exploitability is. Assigning the most severe score to a case of exploitability that is not defined is reasonable but the difference of 0.05 between different cases of exploitability seem to be arbitrary. The question is whether it is possible to create a system that can take advantage of the similarities between vulnerabilities to create scores rather than assigning arbitrary constant numbers to them. There are more formulas in the CVSS guide that also seem to be derived under many assumptions and seem to be arbitrary in some cases.

Other security and software vendors have developed their own scheme of ranking vulnerabilities. For example, the Microsoft TechNet security website [7] contains a list of vulnerabilities affecting Microsoft software with their corresponding CVE entries. Each Microsoft vulnerability also has a "Bulletin Rating" which is intended to rank its severity. The ratings can be critical, important, moderate, or low. In this scheme, each vulnerability is only mapped to one of the four possible ratings. For this reason, this scheme lacks the precision needed to further distinguish numerous vulnerabilities.

Another security vendor that has attempted to rate the severity of vulnerabilities is Secunia [3]. Secunia also has a database of vulnerabilities and assigns scores to them. The scores can be extremely critical, highly critical, moderately critical, less critical, and or not critical. These scores are vaguely defined and

again lack the precision needed to differentiate further between a large number of vulnerabilities.

Chapter 3

Overview of Our Vulnerability Classification System

In this chapter, we present the details of the database we used for our work. In addition, we discuss the overview of data flow in our classification system and the relationship between different relations in our database scheme.

3.1 Database Source and Details

The vulnerability data used for this work is taken from multiple sources. The two main sources of data are the Open Source Vulnerability Database (OSVDB) [1] and the Common Vulnerabilities and Exposures (CVE) database [2]. OSVDB contains 50,933 vulnerabilities with a disclosure year of 2008 or prior. Each vulnerability is given a unique OSVDB ID. Other pieces of information associated with each vulnerability include disclosure, discovery, exploit, and solution dates if known. Moreover, a detailed description, classification information such as location, attack type, impact, exploit, and disclosure, a technical description, a solution description, affected products, associated references, and notes are documented for each vulnerability. Figure 3.1 shows an example of a vulnerability from

OSVDB.

OSVDB also documents exploit information for all vulnerabilities. We will explain how this information plays an important role in predicting exploitability of new vulnerabilities. In Table 3.1, we show the breakdown of vulnerabilities according to their exploit categories.

Table 3.1: Breakdown of Exploit Categories

Exploit Category	**Num Vulnerabilities**	**Label**
Exploit Available	16255	Positive
Exploit Rumored / Private	1918	Positive
Exploit Unavailable	801	Negative
Exploit Unknown	4617	Negative
No Category	27342	Not Used

Note that assigning a positive label to a vulnerability means we have high confidence that the vulnerability is exploited and assigning a negative label means we have high confidence that the vulnerability is not exploited. These labels will be used as part of our training process in the binary classification task of determining exploitability. We explore this concept further in the next chapters.

The CVE database is the other main source of vulnerability data that we use in our system. We integrate CVE with OSVDB using CVE identifiers. Most OSVDB vulnerabilities contain references to their corresponding CVE entries. Also, some CVE entries have references to their corresponding OSVDB vulnerability examples. We use these references to integrate the two databases together. The use of CVE data allows us to extract more features that can potentially be useful for our classification task. These additional features are extracted from various parts of CVE entries such as summary, related products, references, classifications, dates, and scores. We believe this integration allows our system to identify more important features that are essential to the success of our classification process.

Info			
Views This Week	Views All Time		
1	44		
Last Modified	Percent Complete		
about 1 year ago	90%		

Dates			
Disclosure	Discovery		
Sep 03, 2007	Unknown		
Exploit	Solution		
Sep 03, 2007	Unknown		

Description	Speedtech STPHPLib contains a flaw that may allow a remote attacker to execute arbitrary commands. The issue is due to 'stphptablecell.php' not properly sanitizing user input supplied to the 'STPHPLIB_DIR' variable. This may allow an attacker to include a file from a remote host that contains arbitrary commands which will be executed by the vulnerable script.
Classification	**Location**: Remote/Network Access Required **Attack Type**: Input Manipulation **Impact**: Loss of Integrity **Solution**: No Solution **Exploit**: Exploit Available **Disclosure**: Uncoordinated Disclosure **OSVDB**: Web Related
Technical	This vulnerability is only present when the register_globals PHP option is set to 'on'. This has not been the default setting for PHP installs since version 4.2.0 (22-Apr-2002).
Solution	Currently, there are no known upgrades, patches, or workarounds available to correct this issue.
Products	SpeedTech +WATCH / STPHPLib +WATCH / 0.8.0
References	• CVE ID: 2007-4738 (see also: NVD) • Bugtraq ID: 25525 • Secunia Advisory ID: 26658 • ISS X-Force ID: 36417 • Vendor URL: http://stphplib.sourceforge.net/
Notes	http://[victim]/[stphplib_path]/stphptablecell.php? STPHPLIB_DIR=[CODE]

Figure 3.1: A Vulnerability Example from OSVDB

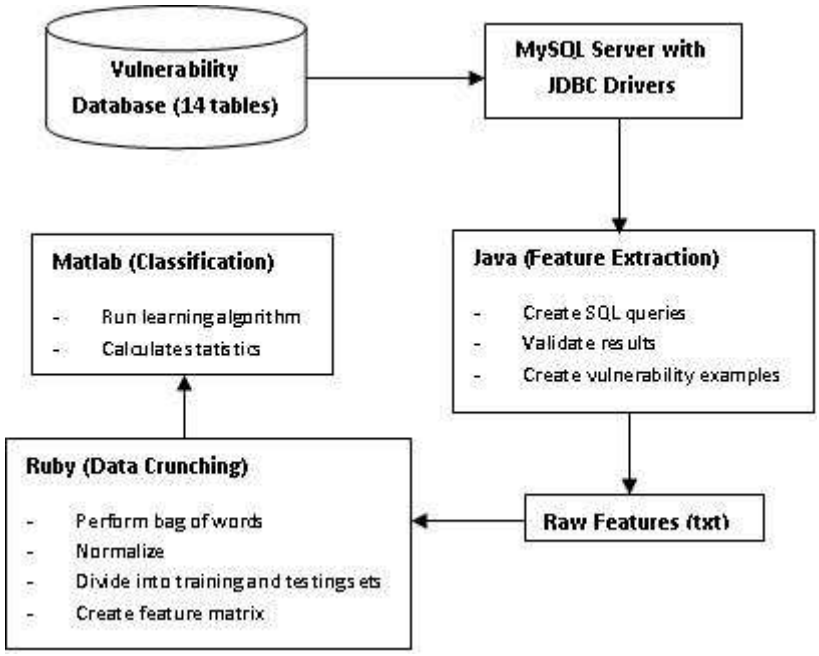

Figure 3.2: The Overview of Our Vulnerability Classification System

3.2 Overview of System Implementation and Data Flow

Here we discuss how we develop a vulnerability classification system based on our vulnerability database. All vulnerabilities are stored in a MySQL database. The MySQL database sits on top of a MySQL server that makes the data available to a Java client using JDBC drivers. The Java client is responsible for multiple tasks. It creates relevant SQL queries and sends them to the database server. The server responds with results. Then, the Java client validates results and creates individual vulnerability examples. Each example contains the features associated with the vulnerability. The details of these features are described in the next chapter.

The end product is now a massive text file that includes all vulnerability examples with their raw features. The next component of this system is a Ruby program that performs data crunching on the text file. This includes performing

bag of words, normalizing the features, dividing the data into training and testing sets, and finally creating a MATLAB matrix that includes all examples with their associated normalized features. After the matrix is ready, the last component of the system, a MATLAB program, starts executing. It runs the learning algorithm on the data matrix to learn weights for a classifier that will be used in the classification process. It also calculates statistics on the data based on the classifier. We used an implementation of LIBLINEAR [12] that interfaces with MATLAB for our learning algorithm. An overview of our vulnerability classification system is shown in Figure 3.2.

Due to the large amount of data and the complex relationship between different features of vulnerabilities, OSVDB has organized this data using relational databases. OSVDB database contains 14 relations (tables), each characterizing a different aspect of vulnerabilities. In Figure 3.3, we show the structure of the database relations taken from the OSVDB website [1]. Our database engine is mainly built based on this schema. However, we make some additional modifications that include integrating this database with a few other databases such as CVE.

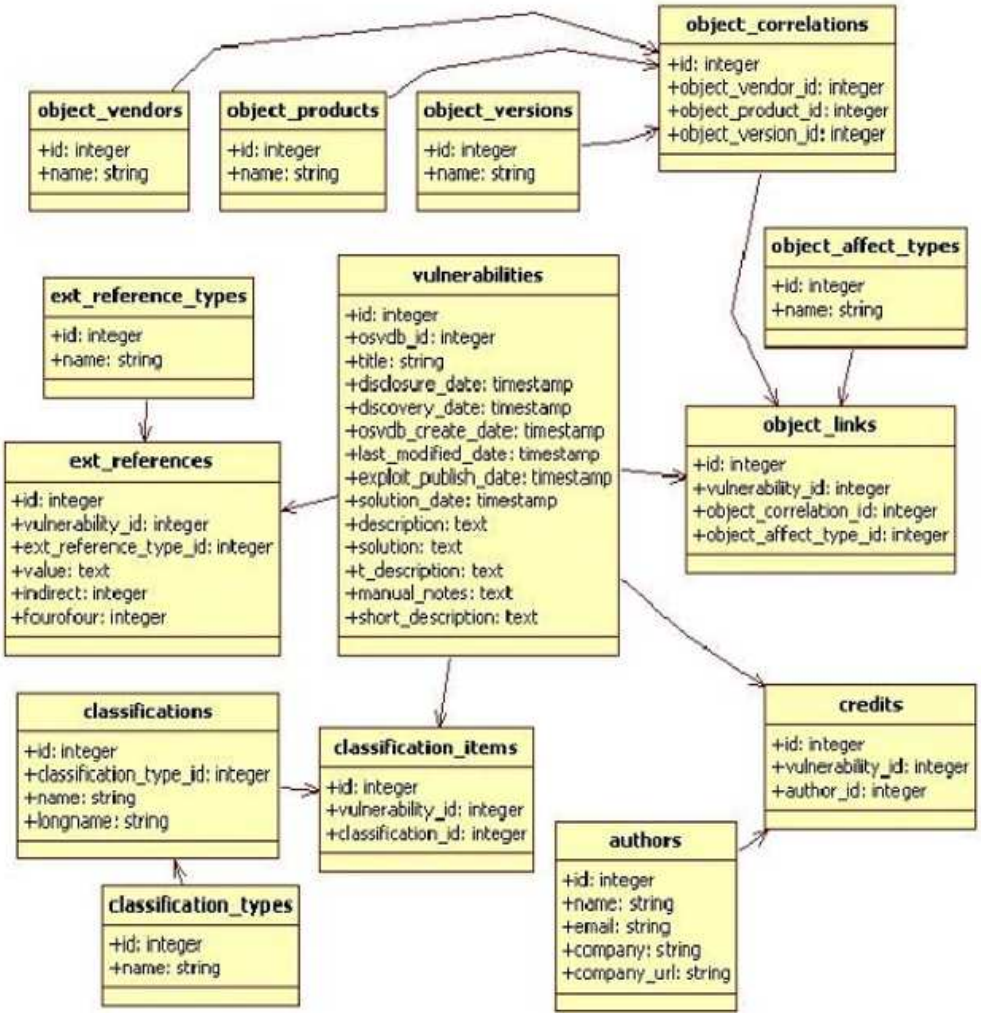

Figure 3.3: The Structure of Database Relations [1]

Chapter 4

Using Machine Learning for Classification

In this chapter, we present the approach we take to solve the problem using supervised machine learning. Moreover, we discuss the details of our process for feature definition and extraction including the "bag of words" model. We also describe our learning algorithm, Support Vector Machines (SVMs), and the reasoning behind choosing this algorithm.

4.1 Supervised Machine Learning Approach

In this section, we introduce a new scheme for classifying vulnerabilities using supervised machine learning. Machine learning focuses on automatically recognizing complex patterns and making intelligent decisions based on data. A supervised learning method is a technique for learning a function (or a classifier) from training examples when each example is associated with a true label. We show how to learn patterns in the vulnerability data in order to classify vulnerabilities.

Supervised machine learning is composed of two main phases. The first phase is the learning or training phase. In this phase, a machine learning algorithm

is run on a fraction of data, training data, to learn a classifier. The training data consist of pairs of input data (summarized as a vector of integers) and their associated output. A classifier is a function that maps input to desired output. The output can be continuous or can be a discrete label. The first case is called regression and the second case is called classification. The main goal of the training phase is to learn suitable parameters for this function such that the classifier can correctly classify the training data with a high precision. Note that the goal is not necessarily to achieve the highest possible precision on training data since it may lead to overfitting [17] which will result in low precision on testing data. The algorithm should be able to generalize given training data to unseen situations in a reasonable way.

The second phase of the machine learning approach is testing. In this phase, the learned classifier is tested on the rest of the data to determine the testing precision of the classifier. The quality of a classifier is judged by its performance on test data. In more advanced approaches, online methods are used. Online methods are appropriate for data that get generated over time in a regular basis. In this approach, testing data are added to previous training data to be used for the next training phase. The training and testing phases can then continue indefinitely and the quality of the classifier can be judged by its accuracy and stability over time.

Here, we explain how this approach can be taken for the task of classifying vulnerability data. Suppose one way of scoring a vulnerability is predicting whether or not it will be exploited in the future. In this scoring system, there are 2 categories: severe and moderate. If a vulnerability is predicted to be exploited, it should be ranked as severe (positive) and otherwise it should be ranked as moderate (negative). Using our vulnerability data, we can randomly extract two sets of positive and negative vulnerability examples. Then, we can learn a classifier that separates these two sets. To make our point more clear, note that each example is

a vector of integer values. Therefore, we can consider each example as a point in some space. The dimensionality of the space depends on the number of features that vulnerability examples will have. In the simplest case, the learned classifier is a hyperplane that separates positive and negative examples in the space such that all positive examples lie on one side of the hyperplane and all negative examples lie on the other side of the hyperplane. The learning task is to learn the parameters of such a hyperplane. To illustrate this point, we present a figure in this section.

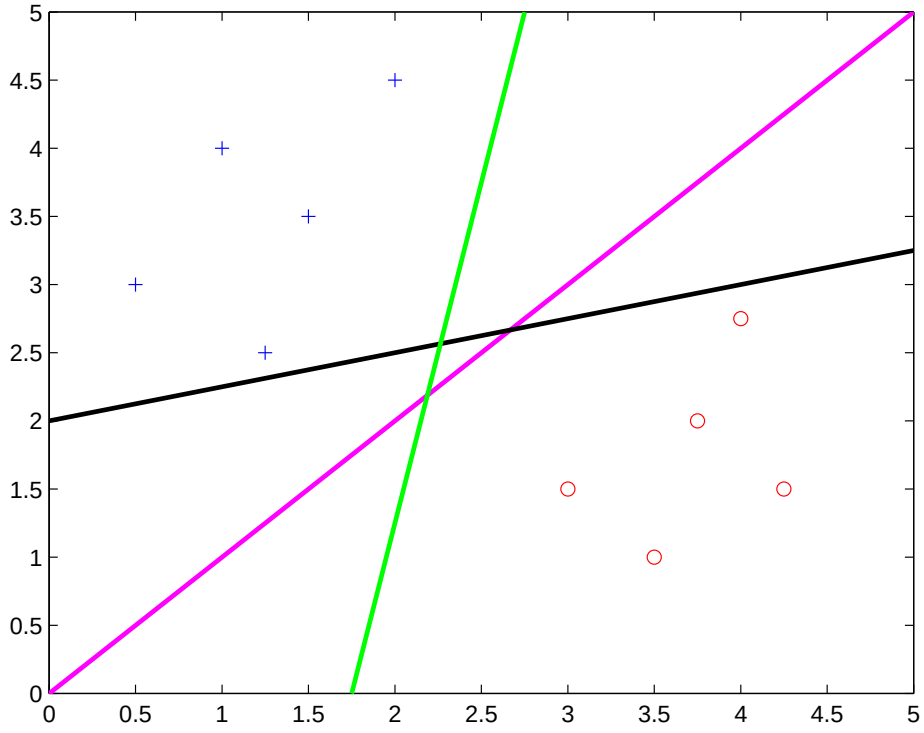

Figure 4.1: Infinitely Many Hyperplanes Separating Positive & Negative Examples

In Figure 4.1, positive examples are shown by blue positive signs and negative examples are shown by red circles. Each example is a point in the two-dimensional space shown in the figure. Note that in this particular example, the data is separable. That is it is possible to learn infinitely many classifiers (hyperplanes) that can completely separate positive and negative examples. In such a

situation, it is reasonable to learn the classifier that maximizes the total distance from all examples, or in other words, the classifier that creates the largest margin. The margin of a point (example) in the space is the distance from the data point to the decision boundary, that is, the distance between the single point and the hyperplane. It makes sense to learn a maximum-margin classifier since it is more likely to minimize the testing error than other classifiers may do. For example, in Figure 4.1, the pink hyperplane is the maximum-margin classifier. Note that the black and green lines are too close to a few of the points. A Support Vector Machine (SVM) algorithm [18] is an example of a maximum-margin classifier. We will discuss this algorithm in more detail in Section 4.4.

4.2 Feature Definition and Extraction

As described in the previous section, we define various features for vulnerabilities to be used during the learning process. These features are extracted from our database and are based on all information we have about the vulnerabilities. Ultimately, all features have to be converted to numbers so that they can be included in the feature vectors associated to the vulnerabilities.

Each vulnerability has a feature vector that characterizes its various quantities. As it can be seen in Table 4.1, we define a list of feature families. Each feature family has a different number of features. Note that the number of features for some feature families are obtained based on a "bag of words" process. In the next section, we define and describe this process in detail. In brief, a bag of words process allows us to quantify those features of vulnerabilities such as text that are not easily convertible to numbers.

A large amount of data available in our database is text describing different aspects of vulnerabilities. For example, title, description, notes, solutions, and a few other feature families are in this category. Table 4.1 also shows the type of

Table 4.1: Breakdown of Extracted Features from Vulnerability Data

Feature Family	**Family Size**	**Type**	**Database Source**
Summary	14883	Binary	CVE
Full Product Name	13040	Binary	OSVDB - Obj. Correlations
Description	11573	Binary	OSVDB - Vulnerabilities
Title	9812	Binary	OSVDB - Vulnerabilities
Short Description	9761	Binary	OSVDB - Vulnerabilities
Manual Notes	6576	Binary	OSVDB - Vulnerabilities
Product Versions	5388	Binary	OSVDB - Obj. Versions
Related Products	5057	Binary	CVE
Product Names	3661	Binary	OSVDB - Object Products
Technical Description	3479	Binary	OSVDB - Vulnerabilities
Solution	3474	Binary	OSVDB - Vulnerabilities
Product Vendors	2500	Binary	OSVDB - Obj. Vendors
Authors	2368	Binary	OSVDB - Credits
Keywords	1556	Binary	OSVDB - Online
References	267	Binary	CVE
Classifications	69	Binary	CVE
External References	31	Binary	OSVDB - Ext. References
OSVDB Dates	15	Integer	OSVDB - Vulnerabilities
Attack Type	11	Binary	OSVDB - Classifications
Category	9	Binary	OSVDB - Classifications
Location	8	Binary	OSVDB - Classifications
Solution Category	8	Binary	OSVDB - Classifications
Disclosure Type	8	Binary	OSVDB - Classifications
CVE Dates	6	Integer	CVE
Impact	4	Binary	OSVDB - Classifications
Scores	3	Integer	CVE
Effect on Products	3	Binary	OSVDB - Obj. Aff. Types
Number of Classifications	1	Integer	OSVDB - Classifications
Number of Class. Types	1	Integer	OSVDB - Classifications
Number of Products	1	Integer	OSVDB - Obj. Correlations
Number of References	1	Integer	OSVDB - Ext. References
Number of Authors	1	Integer	OSVDB - Credits
Number of Views	1	Integer	OSVDB - Online
Percent Complete	1	Integer	OSVDB - Online
Number of References	1	Integer	CVE
Total	93578		

features that we use, which could be binary or integer. Features that are binary signify the existence or non-existence of a certain characteristic in a vulnerability. For example, if a vulnerability exists in some software produced by vendor A, then the binary feature for vendor A is set to 1 in the feature vector of the vulnerability. This feature is set to 0 otherwise. For dates such as disclosure and discovery, we count the number of days passed from that date to the present date and use the result as an integer feature. This will help our algorithm determine whether the magnitudes of the resulting numbers have a positive or negative correlation with corresponding data labels. We also specify the source of the database relation, where the features are extracted from, in Table 4.1.

Note that some of these features may seem to be redundant or irrelevant. This fact will not affect the output of our algorithm since our classifier will be able to detect such features and assign very low (close to zero) weights to them so that they will not affect the outcome. Indeed, one of the main functions of our supervised learning algorithm is to identify and rank important features, and separate them from the ones that are not as important. Many existing scoring systems guess in advance what features are likely to be most important and only use those feature to determine the score of vulnerabilities. We believe, however, feature extraction and identification should be an automatic process and the importance (or weight) of different features should be determined automatically using a fair algorithm such as the one we describe.

4.3 Bag of Words Model

In order to convert a pool of text into a numerical feature vector, we use a bag of words model [14]. Note that most data available for each vulnerability is in the form of text since it comes from different descriptions written about that vulnerability. We perform a significant amount of text processing to transform

Table 4.2: An Example of a "Bag of Words" Feature Vector

BUFFER	HEAP	MEMORY	...	PROXY	DNS
5	7	8	...	1	0

these descriptions to vectors of word counts. Suppose D is a dictionary that contains the set of all distinct words found in the descriptions of vulnerabilities. We build an integer vector v of dimension $|D|$ for each vulnerability feature that naturally comes as text. v has an entry for each word in D which corresponds to the number of occurrences of that entry in the text describing the vulnerability. This model produces vectors of word checks for each feature that comes as text, which can be later used for classification. In other words, we associate the occurrences of words in vulnerabilities to their labels and try to learn the weight for each word feature during the training phase.

Table 4.2 shows an example of a bag of words feature. The first row represents the words in the dictionary and the second row represents the number of occurrences of those words in the text given. This feature is more likely to be describing a vulnerability involving a buffer overflow, and less likely to be describing a vulnerability related to a web service.

A feature vector for a single vulnerability will consist of a number of bag of word features as well as a handful of single-valued features. The dimensionality of a feature vector is the number of entries in it. Note that the dimensionality of these feature vectors will be very high. In Table 4.1, we show how many "bag of words" features are extracted from each feature family in the vulnerability data. The size of each feature family is equal to the number of words that exist in the dictionary associated with that family.

4.4 Discussion of Learning Algorithm (SVMs)

In this section, we describe what Support Vector Machines (SVMs) [18] are and what the motivation is behind choosing them as our learning algorithm. SVMs are a set of supervised learning methods used for classification and regression. The distinction between supervised and unsupervised methods comes from whether there are labels available for the input data. If the labels are binary, that is they are either positive or negative, the input data can be viewed as two sets of vectors in a high dimensional space. An SVM will construct a hyperplane in that space that separates these two sets, and at the same time, maximizes the margin between them. To calculate the margin, two parallel hyperplanes are constructed, one on each side of the separating hyperplane, which are pushed up against the two data sets. A good separation is achieved by the hyperplane that has the largest distance to the neighboring data points of both classes. This is intuitively reasonable since, in general, the larger the margin between the two sets, the lower the generalization error of the classifier.

SVMs are appropriate for very high dimensional spaces since it is possible to completely separate positive and negative examples in such spaces. An infinite number of such separating hyperplanes may exist. However, SVM finds the one classifier that maximizes the margin so that the classification error is minimized. Consider the example in Figure 4.1. The blue plus signs represent positive examples, and the red circles represent negative examples. There exist infinitely many lines that create a valid classifier for this set of points, 3 of which are displayed in the figure. However, only one line, the pink line, is the single best classifier that minimizes the margin of error since it is the furthest from the points.

We believe SVMs are appropriate to use for our problem since the dimension of our problem space is very large due to the use of bag of words model. As shown in Table 4.1, we extract 93578 distinct features from our vulnerability data, which

means the dimension of our space is 93578. We use a significantly smaller number of examples (a maximum of 10% of the dimension of the space) in our experiments. The high dimension of the space and the small number of examples in this problem create high sparsity that requires a model such as SVM for learning.

Chapter 5

Experiment Design and Implementation

In this chapter, we discuss how we design and implement our experiments. We also briefly discuss and interpret the results. First, we present the design of our experiments. We partition the data into a training set and a test set. We leave out the test set while we learn our classifier. We then test the classifier against the data we left out. We also perform cross-validation in which we learn k different classifiers. For each of the k classifiers, a different set of the data will be held out for testing. This way we have a better idea of how our overall classifier will perform. In most cases k is set to 5 or 10.

We refer to examples that have exploits as "positive examples" and examples that do not have exploits as "negative examples". In Table 3.1, we show a breakdown of our data based on their label. Note that we have a total of 18172 positive examples and a total of 5418 negative examples. For balanced experiments, we randomly choose the same number of examples from both sets since the number of positive examples is almost three times as large as the number of negative examples. We also perform experiments using an unbalanced number of positive and negative examples.

5.1 Placing "Bag of Words" Model in Context

Before running our main experiments, we perform two experiments to decide how to use the bag of words model, which we discussed in the previous chapter. Note that each vulnerability example includes many words under different categories. Title, description, technical description, solution, and summary are examples of these categories. One approach to use the bag of words model is to ignore these categories and only count the number of words seen in each vulnerability example despite what category they come from. Another approach is to distinguish between the same words under different categories. For example, under this scheme, if two occurrences of the word "validate" appear once in title and once in solution, they are considered to be two different words. Here we explore which scheme is more successful. More specifically, we determine whether it would be beneficial to ignore the categories from which words come from or whether words should be bound to the context they are used in. We do not include the details of this experiment here, but we discuss the summary of the results.

In the first experiment, we treat all words the same way regardless of their categories. In other words, we do not differentiate between two occurrences of the word "buffer" whether they are both in the same category or they are in two different categories. This means all word counts across different categories are added together to obtain an aggregate count in this experiment. In the second experiment, however, we treat words differently depending on what category they come from. For example, the word "buffer" appearing under category "title" would be different from the same word appearing under category "solution". This allows to further exploit the context from which each word comes from. The results of our experiments suggest that the second scheme leads to significantly more accurate results. We get a higher accuracy of about 10% by putting the words into context. For the rest of this work, we follow the second scheme.

5.2 Predicting Exploitability in a Balanced Experiment

The first main experiment we present here shows how the machine learning framework, more specifically linear SVMs, is useful in predicting the exploitability of vulnerabilities. In other words, we can predict with high accuracy whether given vulnerabilities will be exploited or not in the future. In this experiment, we used about 4000 positive examples (vulnerabilities with exploits) and about 4000 negative examples (vulnerabilities without exploits). Note that we train our algorithm on 40% of the data (training data) and we test it on 50% of the data (testing data). We keep 10% of the data as validation set that will be used during the training process to validate the actions taken by our algorithm to build a classifier. In addition, note that we only use vulnerabilities that were disclosed after year 1990. We exclude all positive examples that do not have a description to allow our bag of words model to be effective. We select a portion of remaining examples randomly to eliminate bias. We present our results in table 5.1.

Table 5.1: Results on Training and Test Sets

Training Phase		**Testing Phase**	
Positive Examples	1407	Positive Examples	1760
Negative Examples	1600	Negative Examples	1999
Total Examples	**3007**	**Total Examples**	**3759**
TN Percentage	100.00%	TN Percentage	91.55%
TP Percentage	100.00%	TP Percentage	85.06 %
FP Percentage	0.00%	FP Percentage	8.45%
FN Percentage	0.00%	FN Percentage	14.94%
Total Accuracy	**100.00%**	**Total Accuracy**	**89.52%**

In Table 5.1, TP, FP, TN, and FN stand for True Positives, False Positives, True Negatives, and False Negatives respectively. TP is the percentage of positive examples that were correctly classified. FP is the percentage of negative examples that were incorrectly classified as positive. TN is the percentage of negative examples that were correctly classified. Finally, FN is the percentage of positive

examples that were incorrectly classified as negative.

In this section, we discuss a few points about the results of this experiment based on Table 5.1. First, note that there are a total of 3007 training examples. We extract 93,578 features from these examples. In other words, we have a space with 93,578 dimensions and only 3007 points in it. Note that the total training accuracy is 100%. This means the training data is completely separable, and, therefore, there are infinitely many hyperplanes that can separate positive and negative examples. The fact that the training data is completely separable is expected since the data resides in a highly sparse space. The linear SVM model is able to find one hyperplane that has the largest possible margin. The accuracy of 100% shows that our algorithm works correctly. Second, note that our algorithm is able to "learn" and use what it learns during the testing phase. We can conclude this fact by comparing our classifier with a random classifier. Suppose we used a random classifier or a biased positive classifier (in the best case) instead of our SVM classifier. Then , the accuracy of such classifier would have been at most $\frac{1999}{1760+1999} = \frac{1999}{3759} = 53.18\%$ on average since there are 1760 positive and 1999 negative test examples. Therefore, our classifier works at least 36.34% ($89.52\% - 53.18\%$) better than the random classifier. This very significant improvement shows that our algorithm is able to take advantage of useful features in the training data to learn a suitable model for testing. Third, note that the accuracy of 89.52% is high enough to allow us to predict with high confidence whether a vulnerability will be exploited or not. Finally, we point out that the experiment is almost completely balanced, so the overall data bias is minimal.

5.3 Most Influential Features

In this section, we discuss what features turned out to be most effective for predicting exploitability. We take two different approaches for this. In the

first approach, we calculate the mutual information between labels and each set of features. In the second approach, we consider the features associated with highest absolute weights.

5.3.1 Mutual Information

One approach for assessing the most effective features is to calculate the "mutual information" between each set of features and true labels of the data. Mutual information is a property used in information theory that measures the mutual dependence of two random variables. Let $I(X, Y)$ denote the mutual information between variables X and Y. Then, $I(X, Y)$ is defined in Equation 5.1 [9].

$$I(X, Y) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log(\frac{p(x, y)}{p(x)p(y)}) \tag{5.1}$$

Note that if X and Y are independent, then $I(X, Y)$ will be zero by Equation 5.1. In this case, X and Y do not share any mutual information or, in other words, are mutually independent. If the value of $I(X, Y)$ is large for a specific $X = x$, then x and Y are highly dependent. For our purposes, x is the values of a specific feature across our examples and Y is the true labels of our examples. We calculate $I(X, Y)$ for all our features.

We present 50 features in Figure 5.1 that have the highest mutual information values with true data labels. Note that the x axis shows the value of $I(X, Y)$ and the y axis lists top features with highest mutual information values. Also, note that the number in brackets in front of each feature represents the number of vulnerability examples for which the associated feature is on.

5.3.2 Highest Absolute Weight

Note that we use a linear SVM as our learning algorithm. After training phase, the algorithm learns specific weights that are used later for testing. Each

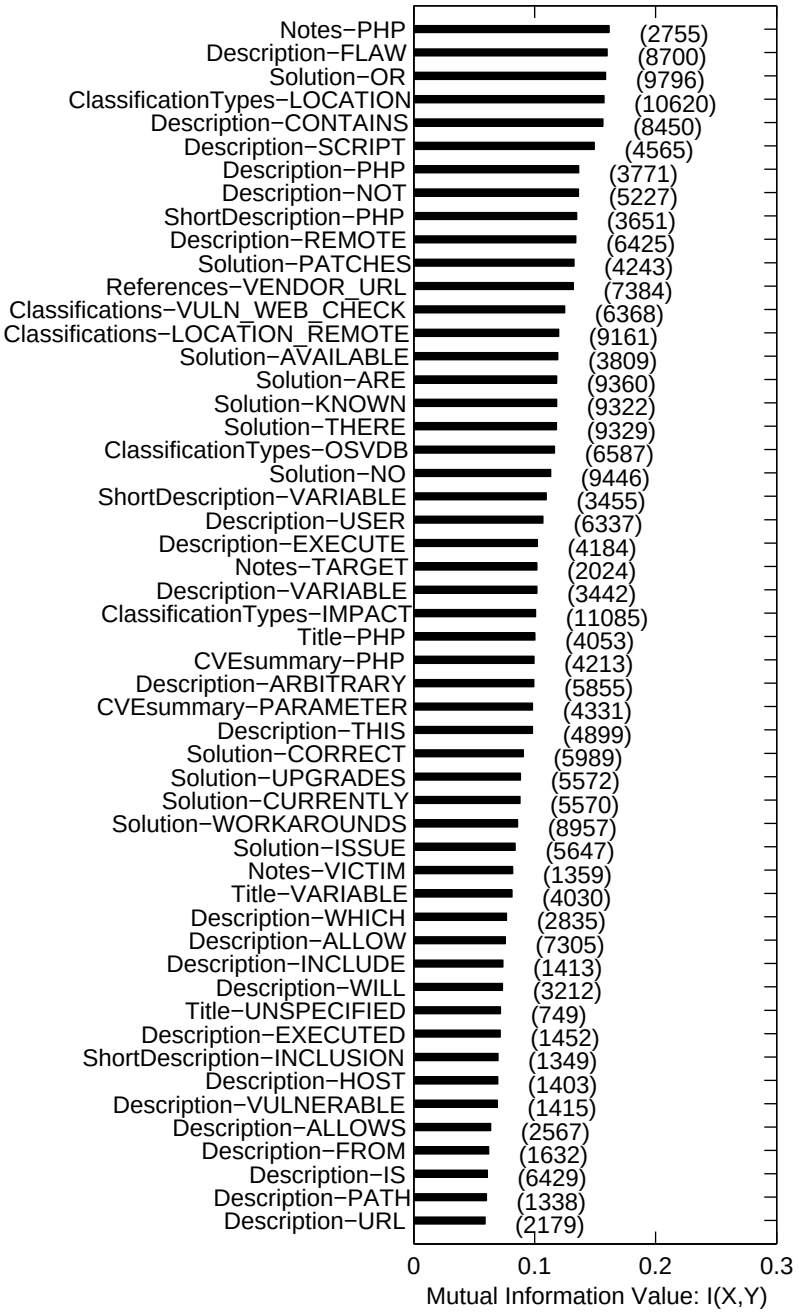

Figure 5.1: Most Effective Features Obtained by Mutual Information Calculations. Note that the integer in brackets in front of each feature represents the number of examples that have that specific feature on.

weight is associated with a certain feature. These weights determine the position of hyperplane in the space. The magnitude of the weight of each feature signifies its importance in the classification process based on the training data. Therefore, it is particularly interesting to examine the features associated with highest absolute weights. This assessment also helps verify the correctness of the algorithm and its effectiveness in identifying important features.

In the following section, we present 4 tables. They show the top positive and negative features together with their weights and the number of vulnerabilities they affect. The first two tables list the features associated with top positive and negative weights. The next two tables are adjusted so that they take into account the number of vulnerabilities affected by high-weight features.

The first column of each table lists the category and name of the feature. For example, "Title-REMOTE" refers to word "remote" appearing in title, which is one of the many features used by the classifier. The second column of each table lists the weight assigned to the feature. The higher the weight, the more important its corresponding feature is to determine the label of the vulnerability example. The third column of the table, $N(j)$, lists the number of vulnerabilities that have the associated feature. For example, this number is 935 for the feature "Title-REMOTE", meaning that the word remote appears in 935 titles of the examined vulnerabilities. Note that $N = 3007$ in all tables.

In Table 5.2, we present the necessary notation.

Table 5.2: Feature Notations

j	Index of a feature
w(j)	Raw weight learned by the classifier for feature j
N(j)	Number of vulnerabilities for which feature j is active
N	Total number of vulnerabilities examined
w'(j)	Effective weight of feature j (w'(j) = w(j) * (N(j) / N))

In Table 5.3, we present the list of top 100 features with highest positive weight. Note that these weights are raw and are not corrected for the number of

vulnerabilities they affect. A positive weight means that the presence of the associated feature has a positive correlation with the exploitability of the vulnerability in the future.

Table 5.3: Top 100 Features (Highest Positive Raw Weight)

Feature (j)	**w(j)**	**N(j)**
References-BUGTRAQ ID	0.492093	2831
defined-Notes-numtokens	0.41733	1582
Title-GET	0.394249	40
References-MILW0RM	0.370689	323
Title-DUNE	0.337216	1
References-GENERIC INFORMATIONAL URL	0.282405	113
ClassificationTypes-LOCATION	0.251214	3656
Title-REMOTE	0.248994	935
defined-AuthorIDs-numtokens	0.240384	2819
Title-EMULE	0.225038	2
CVEproducts-EMULE	0.225038	2
CVEsummary-EMULE	0.225038	2
CVEsummary-PASSWORD	0.224769	161
CVEsummary-RELATED	0.220629	111
Description-CONTAINING	0.219041	83
Title-APPLICATION	0.215975	34
ShortDescription-MALFORMED	0.215725	73
ShortDescription-WITH	0.21395	109

Continued on next page

Table 5.3 – continued from previous page

Feature	Weight	Total
CVEsummary-DIFFERENT	0.21124	172
Title-numtokens	0.210323	4299
CreateModifiedDifference	0.209629	4300
Title-OVERFLOW	0.208896	730
NumberOfReferences	0.203822	4278
CVEsummary-XSS	0.203174	682
CVEreferences-BUGTRAQ	0.201295	2131
Solution-AVAILABLE	0.198297	1400
Title-REQUEST	0.197481	181
CVEsummary-INFO	0.196947	58
Title-EMPTY	0.196933	7
Description-FILENAMES	0.19629	12
Description-SITE	0.196268	756
ShortDescription-PHP	0.195823	1356
CVEsummary-SERVICES	0.194679	44
Title-HANDLING	0.192857	97
CVEsummary-RELOAD	0.192278	14
ShortDescription-RESULT	0.191111	45
Title-LSMCODE	0.187897	1
Description-LSMCODE	0.187897	1
ShortDescription-LSMCODE	0.187897	1
CVEsummary-LSMCODE	0.187897	1
CVEsummary-ARGUMENT	0.187584	101
References-MAIL LIST POST	0.18694	2371

Continued on next page

Table 5.3 – continued from previous page

Feature	Weight	Total
AuthorIDs-4931	0.185325	11
Description-TO	0.184418	3352
Description-BROWSER	0.182664	732
defined-Description-numtokens	0.18062	3522
CVEsummary-SP	0.180527	86
CVEsummary-G	0.179623	143
Title-PARSING	0.179335	22
Description-PERMISSIONS	0.177749	42
Keywords-CSCDX24632	0.176688	1
Description-OF	0.173795	2476
Notes-HTTP	0.170971	1330
CVEsummary-FTP	0.168795	64
Title-CREDENTIAL	0.168337	14
ShortDescription-THE	0.16618	450
CVEsummary-ADMINISTRATION	0.165518	19
Classifications-ATTACK TYPE MISS CONFIG	0.164154	80
Keywords-CLSID	0.16344	3
Description-PASSWORD	0.162997	125
CVEsummary-CERTAIN	0.162888	183
Versions-1.0.0	0.161335	18
Title-PASSWORD	0.160259	82
Description-WHEN	0.159563	949
Description-COULD	0.158852	793
Keywords-BUG	0.158749	3

Continued on next page

Table 5.3 – continued from previous page

Feature	Weight	Total
defined-ShortDescription-numtokens	0.158494	3514
Title-PDF	0.158452	7
Title-YVERINFO	0.157149	1
Keywords-D5184A39-CBDF-4A4F-AC1A-7A45A852C883	0.157149	1
CVEsummary-YVERINFO	0.157149	1
CVEsummary-FVCOM	0.157149	1
CVEsummary-BY	0.155958	588
ShortDescription-MAY	0.155648	187
Title-FUNCTIONALITY	0.155007	2
Classifications-LOCATION LOCAL REMOTE	0.153557	4
CVEsummary-LIST	0.153399	130
Title-BANNER	0.153237	5
CVEsummary-BANNER	0.153237	13
CVEsummary-USERS	0.152858	634
CVEsummary-SUITE	0.15241	32
CVEsummary-IMAGE	0.151351	93
CVEsummary-LIBRARY	0.150947	74
CVEsummary-HTML	0.150606	698
Title-DISCLOSURE	0.149223	351
CVEsummary-LOGGING	0.148785	16
CVEproducts-SECURITY	0.147448	21
Description-RESOURCES	0.1472	29
T Description-THE	0.146402	539
Description-EXECUTE	0.14597	1571

Continued on next page

Table 5.3 – continued from previous page

Feature	Weight	Total
Keywords-AKA	0.145939	28
CVEsummary-THESE	0.145527	45
Title-PROPERTY	0.144248	17
ShortDescription-ARE	0.143327	37
Description-USERNAME	0.143213	72
CVEsummary-ENCRYPTION	0.143143	11
Description-BACK	0.143099	10
ShortDescription-DUE	0.142998	101
CVEproducts-PRODUCTIONS	0.142725	3
Title-ATOMIXMP	0.142725	1

In Table 5.4, we present the list of top 100 features with highest negative weight. Note that these weights are raw and are not corrected for the number of vulnerabilities they affect. A negative weight means that the presence of the associated feature has a negative correlation with the exploitability of the vulnerability in the future.

Table 5.4: Top 100 Features (Highest Negative Raw Weight)

Feature (j)	w(j)	N(j)
LastModifiedTodayDifference	-0.995518	4299
CVEsummary-ATTACK	-0.317767	228

Continued on next page

Table 5.4 – continued from previous page

Feature	Weight	Total
References-KEYWORD	-0.27737	646
ShortDescription-HANDLING	-0.255853	61
Description-UNSPECIFIED	-0.245239	88
Title-INTERFACE	-0.236987	17
CVEgenerateTodayDifference	-0.229159	4240
CVEreferences-VUPEN	-0.218924	1322
Title-IMAP	-0.206173	14
ShortDescription-V	-0.196617	41
Title-UNSPECIFIED	-0.196306	231
CVEsummary-SECURITY	-0.192916	92
CVEsummary-SERVER	-0.191932	538
Keywords-CSCDU15622	-0.191424	1
CVEsummary-PACKETS	-0.184309	59
CVEsummary-SOLELY	-0.184097	51
ShortDescription-B	-0.181523	55
Description-PARAMETER	-0.179622	95
CVEsummary-PATCH	-0.178428	15
Description-HANDLE	-0.17738	39
CVEmodifiedGenerateDifference	-0.175707	4239
CVEsecurityProtection-ALLOWS USER ACCESS	-0.175607	348
AuthorIDs-3888	-0.174809	13
CVEproducts-PRO	-0.17462	64
CVEreferences-IDEFENSE	-0.173793	82
Description-HANDLING	-0.172156	58

Continued on next page

Table 5.4 – continued from previous page

Feature	Weight	Total
Title-PARSER	-0.171851	7
Versions-3.X	-0.17116	19
CVEsummary-COMPONENTS	-0.169271	27
ShortDescription-VALIDATE	-0.167757	49
Description-INFORMATION	-0.167228	370
Title-ERROR	-0.167174	45
Solution-S	-0.166931	146
Description-MEMORY	-0.166257	76
CVEproducts-GNU	-0.163459	23
Solution-FOR	-0.16249	165
Description-DETAILS	-0.161848	140
CVEsummary-ISAKMP	-0.161635	11
CVEsummary-ON	-0.159787	314
Description-AVAILABILITY	-0.15972	437
CVEproducts-WINDOWS	-0.159554	81
ShortDescription-ESCALATION	-0.158997	99
CVEsummary-SUBSYSTEM	-0.158187	5
CVEsummary-OPERATIONS	-0.158168	9
Description-READ	-0.156712	73
ShortDescription-PIOOUT	-0.156709	1
Description-PIOOUT	-0.156709	1
CVEproducts-RC	-0.156133	69
Description-DENIAL	-0.155254	400
CVEsummary-MS	-0.155017	43

Continued on next page

Table 5.4 – continued from previous page

Feature	Weight	Total
Description-FILES	-0.154173	285
CVEsummary-SENT	-0.153681	22
ShortDescription-SP	-0.15308	36
References-CIAC ADVISORY	-0.152796	132
ShortDescription-ARBITRARY	-0.152702	446
Description-BEEN	-0.152693	161
ShortDescription-WOULD	-0.15245	40
CVEsummary-PROCESS	-0.152407	54
CVEreferences-APPLE	-0.151227	140
CVEsummary-UNKNOWN	-0.150995	286
Title-LONG	-0.149348	56
CVEsummary-HALF	-0.146645	3
Description-HALF	-0.146645	3
CVEsummary-CONSUMPTION	-0.146235	69
Title-VALIDATE	-0.146128	2
CVEsummary-EMAIL	-0.145018	124
Title-INCLUSION	-0.144689	604
Title-COMMAND	-0.14454	185
Description-MALICIOUS	-0.143853	538
CVEsummary-INPUT	-0.143757	30
ShortDescription-HAS	-0.143281	39
Keywords-GSA2001-01	-0.142861	1
CVEsummary-BEFORE	-0.142717	681
CVEsummary-SERVICE	-0.142316	843

Continued on next page

Table 5.4 – continued from previous page

Feature	Weight	Total
CVECWE-CWE-264	-0.141327	11
ShortDescription-IS	-0.141055	219
CVEreferences-CERT-VN	-0.140825	318
Classifications-VULN VERIFIED	-0.139019	2092
CVEproducts-X	-0.138862	150
CVEsummary-TEST	-0.137914	35
ID	-0.136271	4300
ShortDescription-S	-0.135438	79
ShortDescription-VARIABLE	-0.134627	1289
Solution-ON	-0.132668	54
Title-MODIFICATION	-0.132532	28
CVEreferences-DEBIAN	-0.132211	310
CVEsummary-LATER	-0.131696	36
CVEsummary-WEBCAM	-0.131371	6
Description-EXPLOIT	-0.131019	20
CVEsummary-WRITE	-0.130879	41
CVEsummary-TITLE	-0.130826	55
CVEsummary-VULNERABILITY	-0.13068	1121
CVEsummary-ICQ	-0.129738	8
Description-LEVEL	-0.128702	11
Description-RESULT	-0.128498	302
CVEsummary-TRAFFIC	-0.128396	15
CVEsummary-SUN	-0.128219	44
CVEsummary-E	-0.128112	207

Continued on next page

Table 5.4 – continued from previous page

Feature	Weight	Total
Description-TAG	-0.128085	18
ShortDescription-C	-0.12793	42
CVEsummary-OVERFLOWS	-0.127191	178

In Table 5.5, we present the list of top 100 features with highest positive weight. Note that these weights are not raw and are corrected for the number of vulnerabilities they affect. Note that the difference between Table 5.3 and Table 5.5 comes from weights. In Table 5.5, we have adjusted weights such that the number of vulnerabilities they affect is taken into account. Note that the equation for calculating the adjusted weights are as follows:

$$w'(j) = w(j) * (N(j)/N) \tag{5.2}$$

Now $w'(j)$ has a direct relation with $N(j)$, the number of vulnerabilities for which feature j is active. For this reason, most numbers in the third column of this table are higher than those in the third column of Table 5.3.

Table 5.5: Top 100 Features (Highest Positive Corrected Weight)

Feature (j)	w'(j)	N(j)
References-BUGTRAQ ID	0.323905	2831
ClassificationTypes-LOCATION	0.21354	3656

Continued on next page

Table 5.5 – continued from previous page

Feature	Weight	Total
Title-numtokens	0.210225	4299
CreateModifiedDifference	0.209581	4300
NumberOfReferences	0.202732	4278
defined-AuthorIDs-numtokens	0.157555	2819
defined-Notes-numtokens	0.153503	1582
defined-Description-numtokens	0.147906	3522
Description-TO	0.143727	3352
defined-ShortDescription-numtokens	0.129493	3514
References-MAIL LIST POST	0.103054	2371
References-numtokens	0.10036	4268
Description-OF	0.10005	2476
CVEreferences-BUGTRAQ	0.0997346	2131
References-SECUNIA ADVISORY ID	0.0875413	2870
CVEsummary-A	0.0820409	2635
ClassificationTypes-numtokens	0.0756017	3919
CVEsummary-REMOTE	0.066929	3407
Solution-AVAILABLE	0.0645467	1400
ShortDescription-PHP	0.0617381	1356
References-OTHER ADVISORY URL	0.0616209	2032
Solution-WORKAROUNDS	0.0576108	3120
ClassificationTypes-IMPACT	0.0552338	3839
Description-ATTACKER	0.0547921	1967
Title-REMOTE	0.0541291	935
Description-EXECUTE	0.0533177	1571

Continued on next page

Table 5.5 – continued from previous page

Feature	Weight	Total
Notes-HTTP	0.0528693	1330
CVEreferences-BID	0.0502184	3024
Description-A	0.0454048	3500
Description-AND	0.0452197	1700
CVEdbID	0.0443583	4240
Solution-CURRENTLY	0.0439625	1971
CVEpublishedGenerateDifference	0.0439558	4238
CreateTodayDifference	0.0433703	4300
CVEExpScore	0.0425557	4236
Description-ALLOW	0.0423486	2536
Description-IS	0.0411538	2268
CVEYear	0.0410842	4240
Title-VARIABLE	0.0372205	1449
CVEproducts-numtokens	0.0367501	4147
CVEsummary-ARBITRARY	0.0364049	2624
References-SECURITY TRACKER	0.0363192	1418
ClassificationTypes-DISCLOSURE	0.0363004	2174
Description-ISSUE	0.0361285	2032
Title-OVERFLOW	0.0354555	730
Description-WHEN	0.035207	949
Description-PHP	0.0347021	1404
Description-SITE	0.0344986	756
References-ISS X-FORCE ID	0.0338644	2568
References-CVE ID	0.0338172	4247

Continued on next page

Table 5.5 – continued from previous page

Feature	Weight	Total
Classifications-LOCATION REMOTE	0.033201	3129
Description-CONTAINS	0.0328037	2963
CVEsummary-XSS	0.0322169	682
Description-numtokens	0.0313342	3522
Solution-TO	0.031153	3543
Description-BROWSER	0.0310881	732
Solution-VERSION	0.0303481	1475
References-VENDOR URL	0.0300072	2664
Description-COULD	0.0292884	793
CVEsecurityProtection-ALLOWS OTHER ACCESS	0.0289775	1266
References-MILW0RM	0.0278383	323
CVEaccessVector-NETWORK	0.0273535	3671
Description-WILL	0.0273169	1240
CVEintegrityImpact-NONE	0.0268437	1081
Classifications-numtokens	0.0265741	3922
NumberOfClassifications	0.0265741	3922
NumberOfClassificationTypes	0.0265741	3922
CVEsummary-numtokens	0.0265633	4240
Solution-IS	0.0255258	1676
References-FRSIRT ADVISORY	0.0254377	1151
CVEsummary-HTML	0.0244415	698
Views	0.0242967	4300
CVEconfImpact-PARTIAL	0.0238653	2347
CVEnumCVEIDs	0.0237214	4240

Continued on next page

Table 5.5 – continued from previous page

Feature	Weight	Total
Description-LOSS	0.0231467	2187
CVEsummary-USERS	0.0225324	634
Solution-PATCHES	0.0222872	1574
CVEsummary-BY	0.0213213	588
Solution-UPGRADES	0.0201518	1969
Notes-PHP	0.0200055	1068
CVEreferences-OSVDB	0.019864	2205
Solution-OR	0.0197987	3440
Description-SERVER	0.0196074	1097
Notes-numtokens	0.0193207	1582
Description-FILE	0.019001	903
ShortDescription-TO	0.0183595	589
T Description-THE	0.018347	539
Description-BY	0.0176833	859
Solution-AS	0.0174679	1543
ShortDescription-THE	0.0173869	450
Solution-ARE	0.0169669	3283
CVEaccessComplexity-LOW	0.0168228	3099
CVEsummary-EARLIER	0.016186	941
Notes-VICTIM	0.0155342	524
CVEOSVDBIDExists	0.0155155	2266
Description-URL	0.0148986	801
CVECVSSScore	0.0139927	4238
T Description-THIS	0.0139226	476

Continued on next page

Table 5.5 – continued from previous page

Feature	Weight	Total
Solution-THERE	0.0137935	3272
CVEaccessComplexity-MEDIUM	0.0133271	853

In Table 5.6, we present the list of top 100 features with highest negative weight. Note that these weights are not raw and are corrected for the number of vulnerabilities they affect.

Table 5.6: Top 100 Features (Highest Negative Corrected Weight)

Feature (j)	w'(j)	N(j)
LastModifiedTodayDifference	-0.995055	4299
CVEgenerateTodayDifference	-0.225909	4240
CVEmodifiedGenerateDifference	-0.173174	4239
ID	-0.136239	4300
defined-CVEreferences-numtokens	-0.106238	4196
CVEproducts-A	-0.0931626	3501
CVEsummary-IN	-0.0892791	3648
CVElastModifiedTodayDifference	-0.0820007	4240
defined-FullProductName-numtokens	-0.0817736	3477
defined-Vendors-numtokens	-0.0817736	3477
defined-Products-numtokens	-0.0817501	3476
ClassificationTypes-ATTACK TYPE	-0.0686285	4249

Continued on next page

Table 5.6 – continued from previous page

Feature	Weight	Total
Classifications-VULN VERIFIED	-0.0676188	2092
CVEreferences-VUPEN	-0.0672909	1322
defined-Versions-numtokens	-0.0652164	3476
Description-THE	-0.0629843	3448
CVEintegrityImpact-PARTIAL	-0.0601103	2573
CVEsummary-TO	-0.0522635	4114
References-RELATED OSVDB ID	-0.0513479	2090
DisclosureCreateDifference	-0.0481163	4300
Description-SCRIPT	-0.0479875	1644
References-KEYWORD	-0.0416603	646
Description-NOT	-0.0415933	1857
Description-FLAW	-0.0407322	3042
ShortDescription-VARIABLE	-0.0403474	1289
Description-IN	-0.0365922	2534
CVEsummary-THE	-0.0362433	3303
CVEsummary-ALLOW	-0.0353709	1346
DisclosureTodayDifference	-0.0346146	4300
CVEsummary-VULNERABILITY	-0.03406	1121
CVEsummary-AND	-0.033552	2641
CVEavailabilityImpact-PARTIAL	-0.032373	2290
defined-NumberOfProducts	-0.0301617	4301
defined-NumberOfAuthors	-0.0301617	4301
defined-NumberOfReferences	-0.0301617	4301
defined-References-numtokens	-0.0301617	4301

Continued on next page

Table 5.6 – continued from previous page

Feature	Weight	Total
defined-NumberOfClassificationTypes	-0.0301617	4301
defined-ClassificationTypes-numtokens	-0.0301617	4301
defined-NumberOfClassifications	-0.0301617	4301
defined-Classifications-numtokens	-0.0301617	4301
defined-CreateModifiedDifference	-0.0301617	4301
defined-DisclosureModifiedDifference	-0.0301617	4301
defined-DisclosureCreateDifference	-0.0301617	4301
defined-LastModifiedTodayDifference	-0.0301617	4301
defined-CreateTodayDifference	-0.0301617	4301
defined-DisclosureTodayDifference	-0.0301617	4301
defined-PercentComplete	-0.0301617	4301
defined-Views	-0.0301617	4301
defined-Title-numtokens	-0.0301617	4301
defined-ID	-0.0301617	4301
Description-CODE	-0.0281752	1290
Description-MAY	-0.0280771	2084
CVEreferences-SECTRACK	-0.0279126	1119
CVEsummary-SERVICE	-0.0278941	843
CVEreferences-CONFIRM	-0.0277145	1135
Description-AN	-0.0266497	1992
CVEsummary-CODE	-0.0254521	1341
defined-CVEproducts-numtokens	-0.0254205	4147
CVEproducts-CPE	-0.0254205	4147
Products-numtokens	-0.0252053	3476

Continued on next page

Table 5.6 – continued from previous page

Feature	Weight	Total
CVEImpactScore	-0.0248608	4235
CVEpublishedTodayDifference	-0.024456	4240
CVEsummary-SERVER	-0.0240083	538
CVEsummary-ATTACKERS	-0.0231929	3405
Description-INTEGRITY	-0.0229365	1449
CVEsummary-BEFORE	-0.0225971	681
FullProductName-numtokens	-0.0219631	3477
CVEsummary-EXECUTE	-0.0217715	1802
NumberOfProducts	-0.0216543	3477
Title-INCLUSION	-0.020319	604
defined-CVEsecurityProtection-numtokens	-0.0198893	2092
CVEsecurityProtection-numtokens	-0.0198893	2092
Classifications-IMPACT AVAILABLE	-0.0197537	742
CVEsummary-PARAMETER	-0.0186912	1555
Description-MALICIOUS	-0.0179941	538
Description-SPECIALLY	-0.0179906	1269
Description-WITHIN	-0.0179308	742
Solution-NO	-0.0175604	3301
Description-TRIGGERED	-0.0171295	939
CVEsummary-ATTACK	-0.0168451	228
CVEsummary-OVERFLOW	-0.0167532	587
CVEsummary-CAUSE	-0.0164868	798
Description-AVAILABILITY	-0.0162282	437
CVEsummary-WHICH	-0.0161003	838

Continued on next page

Table 5.6 – continued from previous page

Feature	Weight	Total
ShortDescription-ARBITRARY	-0.0158348	446
Description-S	-0.0156555	899
Solution-VULNERABILITY	-0.0155886	1998
CVEreferences-numtokens	-0.0151568	4196
Solution-REQUIRED	-0.0151222	1329
Solution-ISSUE	-0.0148528	1996
Description-THIS	-0.0145613	1768
Description-THAT	-0.0144428	3037
Description-DENIAL	-0.0144389	400
Description-INFORMATION	-0.0143861	370
CVEsecurityProtection-ALLOWS USER ACCESS	-0.0142086	348
Solution-THIS	-0.0135877	3521
Description-ALLOWS	-0.0131317	928
CVEreferences-SECUNIA	-0.0130317	2257
Description-OR	-0.0130314	729
AuthorIDs-numtokens	-0.0129849	2819
ClassificationTypes-OSVDB	-0.012622	2316

5.4 Online Time-to-Time Experiment

One important aspect of a classification system is its ability to maintain good prediction results over time. In a real world scenario, we expect our classifier to be used constantly over time to produce predictions. Suppose we want to use this classifier in industry. We can use all existing vulnerabilities as training examples

to build a classifier, which can be served as our base classifier. Then, as time progresses, new vulnerabilities are discovered, for which we can make predictions using our base classifier. Then, after the true labels of those vulnerabilities are determined, that is, it is determined whether they are exploited or not, they can be included in the pool of existing vulnerabilities that we had before, and use them all together as training examples to rebuild our classifier. This task can continue indefinitely as time progresses.

To formulate this experiment, suppose $\{X\}_0^t$ represents all examples, X, seen between time 0 and time t. Suppose $\{Y\}_0^t$ is their corresponding labels. Then, we build a classifier C_t for $\{X, Y\}_0^t$. Now, suppose new examples $\{X\}_{t+1}^{t+T}$ arrive after a duration of T time elapses. We use C_t to predict the labels $\{Y'\}_{t+1}^{t+T}$ for these examples. Now, once we know what the true labels, $\{Y\}_{t+1}^{t+T}$, are, we calculate the error of our classifier C_t. Then, we include all new examples into our training pool, $\{X, Y\}_t^{t+T}$, to build a new classifier C_{t+T}. Note that T is a constant time period that we can adjust as we wish. We call this process a time-to-time experiment. At each point of time, we calculate the error made by our classifier and plot it on a figure. Note that the error is the percentage of misclassified examples over all examples. Also note that we have backdated the experiment as if it started in January 2005. Therefore, the base classifier is trained on all examples prior to year 2005, and then, the online classification continues from January 2005 to the end of year 2007.

We have performed this experiment for three values of T. In the first experiment, T is one week. In the second experiment, T is one month, and, in the third experiment, T is one year. Figure 5.2 shows the results of this time-to-time process.

Note that the x axis shows years, months, and weeks respectively. There are a total of 3 years, 36 months, and 160 weeks between year 2005 and year 2007. The y axis shows the error percentage calculated at each point as well

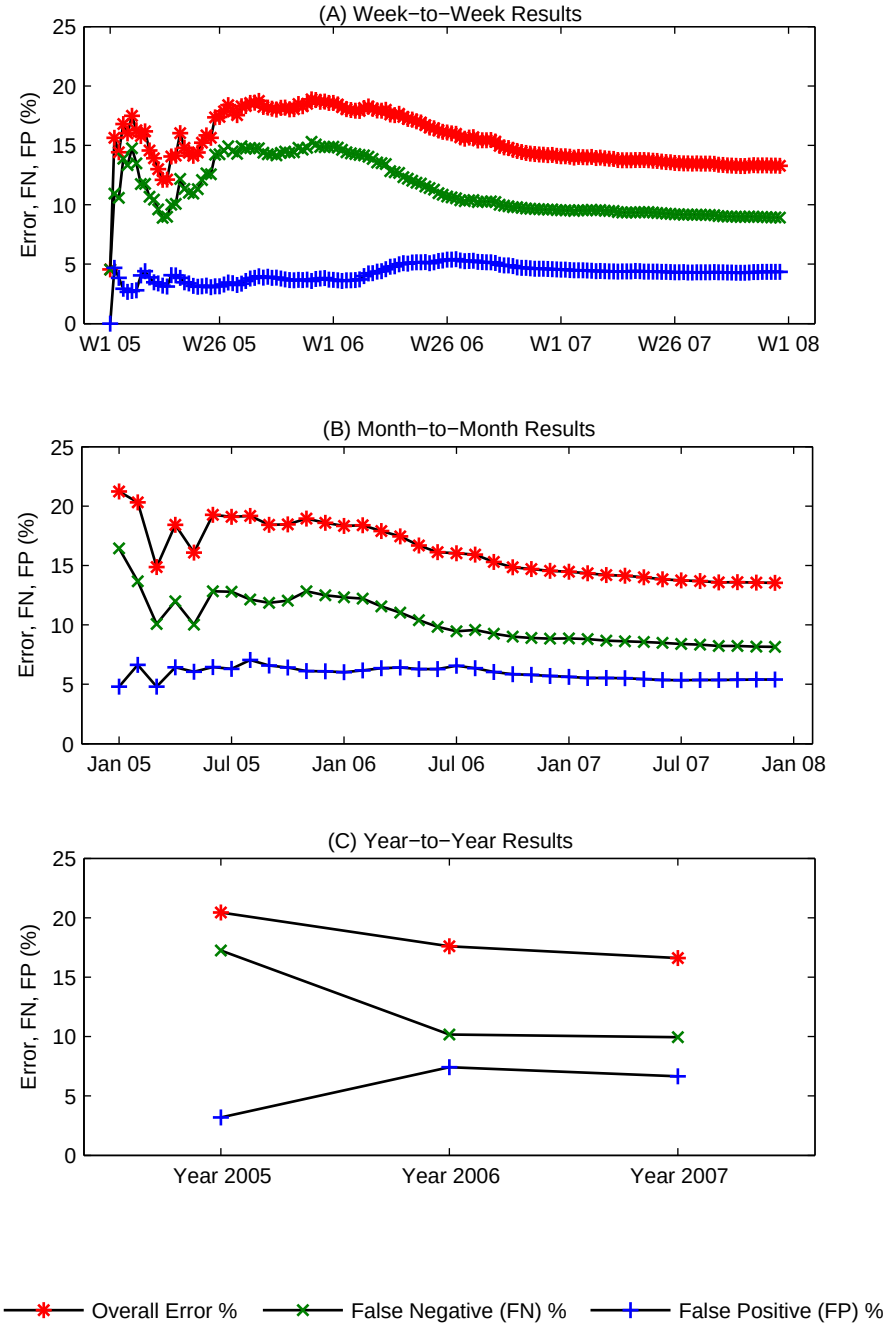

Figure 5.2: Cumulative Error, FN, and FP Percentage in Predicting Exploitability. This figure shows the results of a real-world experiment, where each point in the figure at time t is obtained by training the classifier on all vulnerabilities from time 0 to time t and testing on vulnerabilities from time t to time $t + T$, where T is 1 week in (A), 1 month in (B), and 1 year in (C). Note that the error is decreasing over time, so the classifier becomes more stable over time.

as the False Negative (FN) and False Positive (FP) rates. The red lines show the overall cumulative test errors, which are compromised of the percentage of positive misclassified examples (FN) and negative misclassified examples (FP). We show the cumulative error to demonstrate the stability of our algorithm. The green lines show the FN rate and the blue lines show the FP rate. Note that the FP rate is always significantly lower than the FN rate. Also, note that our algorithm remains stable over time and on average has an overall test error rate of 14% for this experiment.

5.5 Predicting Time to Exploitability

In this section, we explain how to extend and adjust the presented machine learning framework to predict other metrics that help assess the severity of vulnerabilities. As mentioned before, exploitability is probably the most important factor in determining the severity of a vulnerability. However, there might be other factors that potentially affect its severity. To clarify this fact, we present an example. Imagine a scenario where our classifier predicts that the vulnerabilities v_1 and v_2 will be both exploited. Let R be our ranking or severity function. Suppose $R(v_1) > R(v_2)$ according to our algorithm. That is v_1 is more severe than v_2. Then a software vendor that has to fix v_1 and v_2 should rationally fix v_1 before fixing v_2. Let $T(v)$ denote the time elapsed between the discovery of the vulnerability v and the availability of its exploit. Now if $T(v_1) >> T(v_2)$, that is, it takes much longer for the exploit of v_1 to become available than that of v_2, then it makes more sense to fix v_2 before v_1. This example shows why the "time to exploit" of a vulnerability is also an important factor in determining its severity.

It is easy to extend our machine learning framework to predict the "time to exploit" for vulnerabilities. We can still use exactly the same features as we used to predict exploitability. There is no need to perform extra task of feature

definition and extraction. All that needs to be done is changing the labels of the vulnerability examples and, in some cases, adjusting the learning algorithm. For this prediction scenario, we can simply change the binary labels of vulnerabilities. Note that the binary labels previously denoted whether or not a vulnerability will be exploited. Now, we can change them to denote whether or not a vulnerability will be exploited in time t, where t is the difference between the exploit date and disclosure date of the vulnerability. Of course, in this scenario, we cannot use vulnerabilities that do not have an accurate disclosure and exploit dates.

Here we perform two sets of experiments. In the first set, which is based on random permutation of data, we perform a number of experiments with different strategies to predict time to exploitability of vulnerabilities. Table 5.7 lists the descriptions of these experiments. Note that P denotes the set of positive examples and N denotes the set of negative examples.

Table 5.7: Experiment Names and Descriptions

Name	**Description**
+ day (Strategy A)	$t > 0$; excluding 0 or negative-day exploit examples in P
+ day (Strategy B)	$t > 0$; including 0 or negative-day exploit examples in P
0 day (Strategy A)	$t = 0$; excluding negative-day exploit examples in N
0 day (Strategy B)	$t = 0$; including negative-day exploit examples in N
- day (Strategy A)	$t < 0$; excluding 0-day exploit examples in N
- day (Strategy B)	$t < 0$; including 0-day exploit examples in N
- and 0 day	$t <= 0$; including positive-day exploit examples in N

Next, we present the experiment results in Table 5.8. Note that $|P|$ denotes the number of the positive examples and $|N|$ denotes the number of the negative examples. "Base" is the error obtained by a random classifier in the best case. "Error" is the error of our classifier. "Learning" shows how much our classifier has been able to learn, which is simply the difference between "Base" and "Error".

In the second set of experiments and results, we show the results based on online "time-to-time" training and testing. Figure 5.3 shows the results of the second set for $t = 2$ days. Note that for the experiments in this section, we

Table 5.8: Results of "Time-to-Exploit" Experiments

Experiment Name	t	$\|P\|$	$\|N\|$	**Base**	**Error**	**Learning**
+ day (Strategy A)	30	2737	1303	32.26%	20.18%	12.08%
+ day (Strategy A)	14	2332	1706	42.25%	22.94%	19.31%
+ day (Strategy A)	7	1960	2076	48.56%	24.22%	24.34%
+ day (Strategy A)	2	1404	2632	34.79%	21.99%	12.80%
+ day (Strategy B)	30	11508	1303	10.17%	2.66%	3%
+ day (Strategy B)	14	11104	1706	13.32%	8.27%	5.05%
+ day (Strategy B)	7	10733	2076	16.21%	9.35%	6.86%
+ day (Strategy B)	2	10179	2632	20.54%	10.75%	9.79%
0 day (Strategy A)	0	7210	3823	34.65%	18.05%	16.60%
0 day (Strategy B)	0	7210	5627	43.83%	23.22%	20.61%
- day (Strategy A)	< 0	1811	3823	32.14%	15.22%	16.92%
- day (Strategy B)	< 0	1811	11019	14.12%	9.28%	4.84%
- and 0 day	$<= 0$	8996	3823	29.82%	14.88%	14.94%

integrated our database with the security ecosystem data from Techzoom [13] to further improve the accuracy of the exploit and disclosure dates of vulnerabilities.

5.6 Correlation Comparison of SVM Model and CVSS Exploitability Scores

In order to evaluate our model for the task of exploitability prediction and ranking, we devote this section to compare the performance of our model with CVSS. We perform an experiment to make this comparison. Note that CVSS assigns exploitability scores to each vulnerability. We compute the correlation [16] of those scores with true labels of vulnerabilities to determine the validity of those scores and compare them to correlation scores obtained for our model. We can use two different measures as scores by our model for the vulnerabilities. One measure is the predicted label of each vulnerability. Another measure is the dot product of the feature vector, x, with the weight vector, w. We denote this dot product by $x.w$. This is also a valid measure as our SVM learning algorithm uses this measure to determine the label of a test example. If the sign of $x.w$ is positive, the

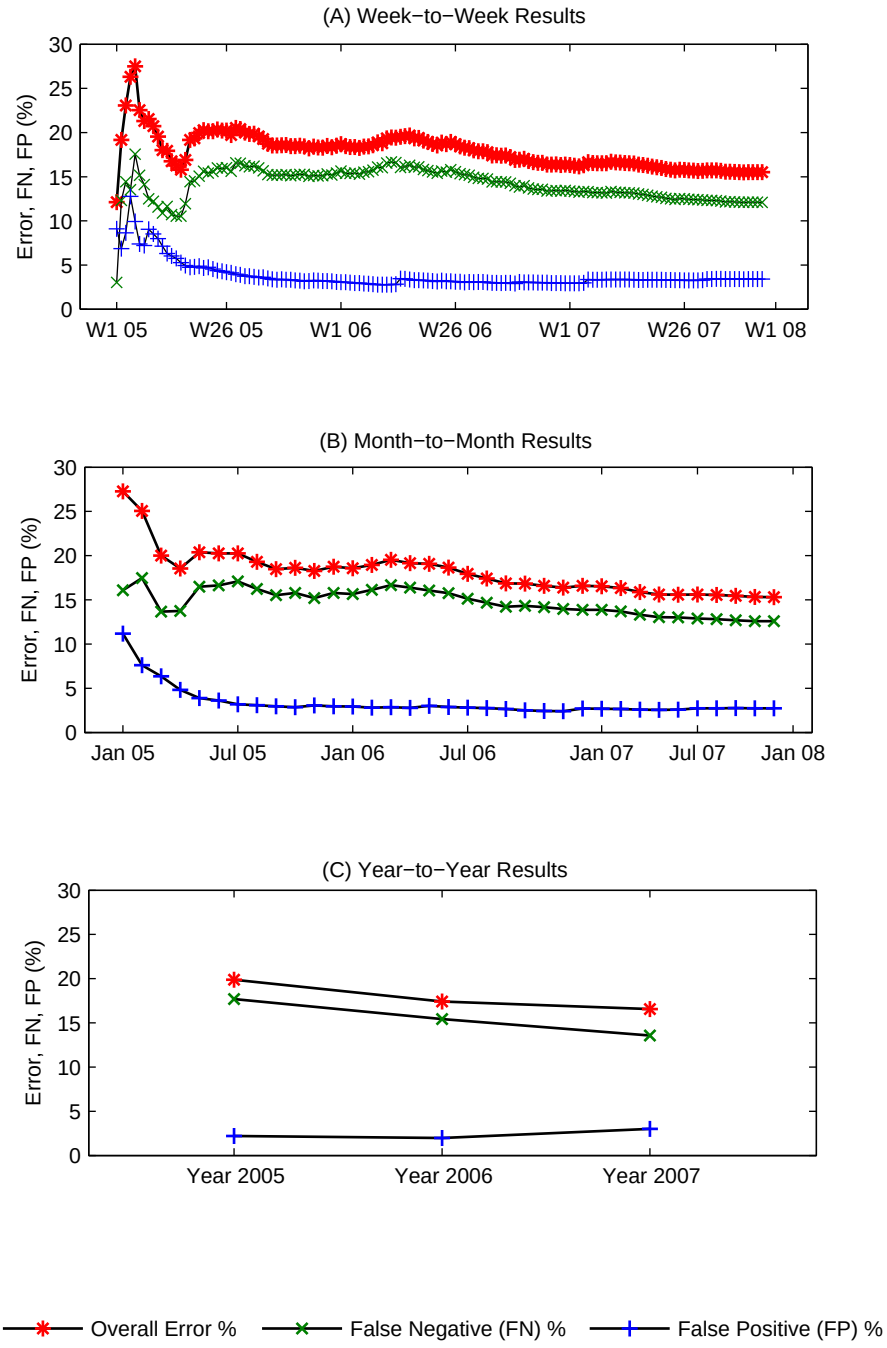

Figure 5.3: Error, FN, and FP Percentage in Predicting "Time to Exploitability." "Time to exploitability" for this experiment is 2 days. This experiment simulates a real-world scenario, where each point in the figure at time t is obtained by training the classifier on all vulnerabilities from time 0 to time t and testing on vulnerabilities from time t to time $t + T$, where T is 1 week in (A), 1 month in (B), and 1 year in (C). Note that the error is decreasing over time, so the classifier becomes more stable over time.

vulnerability is predicted to be positive and if the sign is negative, it is predicted to be negative. Table 5.9 shows the result of this experiment.

Table 5.9: Correlation of True Labels against Learned Model and CVSS Exploitability Scores

Model	**Training**	**Testing**	**Validation**	**All**
CVSS Exp. Scores (all)	0.0900	0.1115	0.0963	0.1013
CVSS Exp. Scores (- {r=10})	0.3375	0.3501	0.3723	0.3469
x.w	0.9380	0.7962	0.8042	0.8550
Sign(x.w)	1.0000	0.8003	0.7989	0.8800

Note that we show the correlation analysis based on different datasets: training, testing, and validation. Here we discuss a few points about Table 5.9. The first row shows the correlation between true data labels and CVSS exploitability scores. In the second row, we exclude all CVSS scores of 10, which is the maximum possible score. Note that the correlation significantly increases after this exclusion. The intuition behind this fact is that CVSS automatically assigns a score of 10 to all newly discovered vulnerabilities. This is justified as a cautious action but, as seen in Table 5.9, it significantly reduces the accuracy of the scores. The third row shows the correlation between the dot product of feature vector and weight vector ($x.w$) learned by our model and true data labels. Finally, the last row shows the correlation of predicted labels, using our SVM model, and true data labels. Note that the correlation numbers for our model is significantly higher than those by CVSS, and therefore, our model can predict the exploitability of vulnerabilities with higher accuracy. Moreover, note that we use CVSS scores as features in our model. However, the presence of these scores does not significantly affect the prediction power of our model due to their relative low correlation with true labels.

In Table 5.10, we show an interesting fact. We analyze the relationship between the CVSS impact scores and the true labels of examples. We see that this correlation number is negative, which means, according to CVSS, the impact of a vulnerability has a negative correlation with its exploitability. One reason why this fact is true may be because low-impact vulnerabilities are generally easier to

exploit. This fact opens a possible research direction that will be interesting to explore.

Table 5.10: Correlation of True Labels and CVSS Impact Scores

	Training	**Testing**	**Validation**	**All**
CVSS Impact Scores	-0.1913	-0.2191	-0.1857	-0.2046

Chapter 6

Results and Discussion

In this chapter, we discuss the results of our experiments presented in the previous chapter and further explore their implications. First, we point out the high accuracy of our machine learning framework for predicting the exploitability of vulnerabilities as well as time to exploitability. Second, we discuss how this framework can be used to introduce a new scheme for scoring and ranking vulnerabilities. Then, we discuss the advantages of our system over existing scoring systems. Finally, we point out how our machine learning framework can be easily adjusted to predict metrics other than exploitability that potentially impact the severity of a vulnerability. This generalization will show the usefulness of our framework for other prediction scenarios.

6.1 Accurate Exploitability Prediction

Predicting whether a vulnerability will be exploited or not is an important task in determining the importance and severity of a vulnerability. Our algorithm is very successful in this task as shown by the results of our experiments in the previous chapter. Note that we get 100% accuracy on the training data and almost 90% accuracy on our test data. Therefore, our algorithm can predict with high confidence whether a given vulnerability will be exploited or not. Based on our

knowledge, no other existing method can achieve such high accuracy in predicting exploitability. To further validate our results, we present a correlation analysis in the previous chapter that shows the most well-known vulnerability scoring system, CVSS, predicts exploitability with significantly lower accuracy. Note that we get an overall correlation of 0.88 between true labels and our scores, while that number is 0.1013 for CVSS. This fact signifies the success of our system in predicting exploitability.

6.2 Introducing a New Scoring Scheme

We showed how the machine learning framework can be used to classify vulnerabilities as positive or negative. We are yet to answer the question that how this framework can be used to rate vulnerabilities against one another. Note that it is obvious to define a binary scoring scheme in this framework. Simply assign severe scores to those vulnerabilities that are predicted to be exploited in the future, and assign moderate scores to other vulnerabilities. However, we can define a more precise scoring system based on the feature and weight vectors associated with vulnerabilities. Note that, in the linear SVM model, the sign of the dot product, $x.w$, determines whether a vulnerability is classified as positive or negative. We can use the magnitude of this dot product as the score assigned to each vulnerability. Suppose v_1 and v_2 are two vulnerabilities. Assume $x_1.w_1 = -1$ and $x_2.w_2 = -4$, where x_1 and w_1 are the feature and weight vectors associated to v_1, and x_2 and w_2 are the feature and weight vectors associated to v_2. Based on our model, both v_1 and v_2 will be classified as negative. However, there is an obvious difference between them. v_2 is much more likely to have a true negative label since it has a higher absolute dot product value. This scenario shows why it is reasonable to use the magnitude of this dot product as the score for severity of a vulnerability. Note that this scoring system is no longer binary and will be very

precise since all scores will be real numbers. It is easy to obtain a total ordering of vulnerabilities based on these scores, which allows the users of this scheme to rank all vulnerabilities against one another.

6.3 Advantages over Existing Scoring Systems

Our algorithm has a few clear advantages over existing scoring systems such as CVSS. We point out a few of these advantages. First, our algorithm is highly accurate in comparison to CVSS. We discussed this fact in the above section. Second, our algorithm can easily take advantage of all information available for a given vulnerability without loss of speed and efficiency or fear of irrelevance and duplication. For example, with arrival of a new feature for a vulnerability, that feature can be easily added to the pool of existing features. A new weight will be learned for that feature and other existing features. Then based on this weight, the corresponding feature will be automatically used or ignored in the classification process. There is no need to worry about whether this feature is a duplicate of another feature or is irrelevant. If the feature happens to be duplicate or irrelevant, the algorithm will automatically learn appropriate weights. This high flexibility allows us to use every bit of information available for a vulnerability ranging from its discovery date to the number of occurrences of a specific word in its technical description. Third, our algorithm can be generalized to other prediction scenarios. Some security researchers believe the impact of a vulnerability should also be taken into account in determining its rank. For this scenario, we can simply change the labels of our vulnerability examples to their impact, train our algorithm on those examples, and then predict the impact of future vulnerabilities. Note that the labels of vulnerabilities denote that ground truth. This ground truth can be set to be their exploitability, time to exploit, impact, or any other aspect of vulnerabilities that are seen important for ranking them against each other. Finally, our algorithm

can be easily integrated with other scoring systems to create a more accurate and robust classifier. The integration can be done by simply using the metrics of other systems as new features in this machine learning framework. This way we can even integrate, for instance, the top 10 well-known scoring systems together to build one system that maximizes accuracy and is still very efficient.

Chapter 7

Future Work

This work has high potential for future work. We believe several directions may be taken to continue and improve the current state of research explained in previous chapters. In this section, we briefly describe these directions.

7.1 Semi-Supervised Learning

Semi-supervised methods [8] might be especially useful for vulnerability data. These methods take a balanced approach between supervised and unsupervised learning. Since the data we have for vulnerabilities are not completely labeled, this method may significantly help obtain higher precision by using all of the data. For example, in the case of predicting exploitability, more than half of the data is not labeled. That is, it is not known whether exploits exist for corresponding vulnerabilities or not. This large part of dataset can be used in a semi-supervised setting to accelerate learning and increase the validity and generality of the results.

7.2 Unsupervised Learning

Note that we used a supervised learning method, SVMs, to learn a classifier for this problem. Another approach is to use unlabeled data instead and use an unsupervised learning scheme [11]. In such a scheme, the algorithm will look for natural similarities in the input data and will try to separate data into multiple groups. This method is specifically beneficial to determine how many groups are reasonable to define and what group each example belongs to. This method, which is widely referred to as clustering, will allow to classify vulnerabilities more precisely into multiple clusters. The downside to this approach is that the underlying process may not generate the data in different clusters and, therefore, the resulting clusters will not be valid.

7.3 Data Source Integration and Feature Enhancement

Another direction to take in the future to improve the accuracy and stability of the machine learning framework is to integrate data from other sources into our system to extract more useful features. Note that the data we use for this work mainly comes from OSVDB and a few other sources such as CVE. There are other security vendors or organizations that store data about vulnerabilities and do analytics on top of the data. All these sources of information can also easily be used to further enhance our system and extract more useful features. Having the machine learning framework set up, it allows us to easily integrate our system with these additional data sources. This direction allows us to improve the learning process and feature selection so that it can capture more characteristics of vulnerability examples in order to achieve higher accuracy. This is especially important since we might not currently be using some features that could potentially

have a high correlation with our labels. In addition, patch information could be a valuable source of learning. For instance, whether or not there exists a patch for a vulnerability or the time between the discovery of a vulnerability and when the patch for that vulnerability was released are both possibly very useful features.

Note that there are many hyperlinks associated with each vulnerability in the OSVDB database. Each hyperlink points to another security site that lists information about that vulnerability. We can create an automated process that follows these hyperlinks and adds the additional information to our database. It will not be surprising if very valuable features can be obtained as a result of this task.

Chapter 8

Conclusion

Ranking vulnerabilities is a critical task for software companies. With thousands of vulnerabilities in hand and limited resources to fix them, it is important to prioritize these vulnerabilities. Vulnerabilities not only hurt the reputation of a company, they can also lead to very significant financial damages and create tremendous liability that not all companies are able to handle. Moreover, it is important for the end users of software to be able to assess the severity of the vulnerabilities in software they use and predict their risk. Vulnerable software can potentially damage the computers of end users as well as servers, and, in some cases, negatively affect a portion of Internet.

The efforts for ranking and scoring vulnerabilities have been independent and lack cohesion. Many security vendors have tried to score vulnerabilities independently. Some software vendors such as Microsoft rate the vulnerabilities of their software on their own. Standard systems such as CVSS have been developed to create industry metrics for scoring vulnerabilities. However, these systems have not been successful to capture every aspect of vulnerabilities for scoring, and, as we showed in previous chapters, they are not highly accurate.

The introduction of machine learning framework allows to consolidate all these efforts together to create a system that not only produces highly accurate re-

sults, but also integrates the existing systems together to leverage from their data and methods. The flexibility of feature selection and extraction process allows the machine learning framework to create a stable system for vulnerability classification and prediction. The prediction system is not only limited to predicting the exploitability of vulnerabilities. It can also predict any vulnerability-related metric using the same features if there is enough information available for that metric. The metric can be the time it takes for a vulnerability to be exploited or its impact in a local or global information system environment.

This work has high potential for future work. We suggest extending the machine learning framework to use semi-supervised and unsupervised methods in the future to leverage from vulnerabilities that have incomplete information. We also suggest integrating other prediction and classification systems into this framework to obtain more accurate results and more stable classifiers.

Bibliography

[1] OSVDB. The Open Source Vulnerability Database. http://osvdb.org/.

[2] CVE Editorial Board. Common Vulnerabilities and Exposures: The Standard for Information Security Vulnerability Names. http://cve.mitre.org/.

[3] Secunia Advisories. Secunia Corporation. http://secunia.com.

[4] Security Tracker. SecurityGlobal.net LLC. http://securitytracker.com.

[5] Security Focus. Symantec Corporation. http://www.securityfocus.com.

[6] Forum of Incident Response and Security Teams (FIRST). Common Vulnerabilities Scoring System. http://www.first.org/cvss/.

[7] TechNet Security Team. Microsoft Security Bulletin. Microsoft Corporation. http://www.microsoft.com/technet/security/current.aspx.

[8] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006.

[9] T. Cover and J. Thomas. Elements of information theory. 1991.

[10] C. Dougherty. Vulnerability metric, Updated on July 24, 2008. https://www.securecoding.cert.org/confluence/display/seccode/Vulnerability+Metric.

[11] R. O. Duda, P. E. Hart, and D. G. Stork, editors. *Unsupervised Learning and Clustering*. Wiley, New York, 2001.

[12] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. A library for large linear classification. http://www.csie.ntu.edu.tw/cjlin/liblinear/.

[13] S. Frei, D. Schatzmann, B. Plattner, and B. Trammel. Modeling the security ecosystem - the dynamics of (in)security. *Workshop on the Economics of Information Security (WEIS)*, 2009.

[14] D. Lewis, editor. *Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval*. Proceedings of ECML-98, 10th European Conference on Machine Learning, 1998.

[15] P. Mell, K. Scarfone, and S. Romanosky. A complete guide to the common vulnerability scoring system version 2.0, June, 2007. http://www.first.org/cvss/cvss-guide.html.

[16] J. L. Rodgers and W. A. Nicewander, editors. *Thirteen ways to look at the correlation coefficient. The American Statistician.* 1988.

[17] S. Russell and P. Norving, editors. *Artificial Intelligence: A Modern Approach.* Prentice Hall Series in Artificial Intelligence, Englewood Cliffs, New Jersey, 1995.

[18] B. Schlkopf and A. J. Smola, editors. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, Cambridge, MA, 2002.

[19] US-CERT. Computer Emergency Readiness Team. http://www.us-cert.gov.

[20] Wikipedia. The free encyclopedia. Vulnerability (Computing), Updated June 26, 2009. http://en.wikipedia.org/wiki/Vulnerability_(computing).