Exploitability prediction of software vulnerabilities

Navneet Bhatt[1], Adarsh Anand[1,*], V. S. S. Yadavalli[2]

[1] Department of Operational Research, University of Delhi, Delhi, India
[2] Department of Industrial and Systems Engineering, University of Pretoria, Pretoria, Republic of South Africa

*Correspondence
Dr. Adarsh Anand, Assistant Professor, Room No. 208, 2nd Floor, Department of Operational Research, Faculty of Mathematical Sciences, University of Delhi, Delhi 110007, India.
Email: adarsh.anand86@gmail.com

ABSTRACT

The number of security failure discovered and disclosed publicly are increasing at a pace like never before. Wherein, a small fraction of vulnerabilities encountered in the operational phase are exploited in the wild. It is difficult to find vulnerabilities during the early stages of software development cycle, as security aspects are often not known adequately. To counter these security implications, firms usually provide patches such that these security flaws are not exploited. It is a daunting task for a security manager to prioritize patches for vulnerabilities that are likely to be exploitable. This paper fills this gap by applying different machine learning techniques to classify the vulnerabilities based on previous exploit-history. Our work indicates that various vulnerability characteristics such as severity, type of vulnerabilities, different software configurations, and vulnerability scoring parameters are important features to be considered in judging an exploit. Using such methods, it is possible to predict exploit-prone vulnerabilities with an accuracy >85%. Finally, with this experiment, we conclude that supervised machine learning approach can be a useful technique in predicting exploit-prone vulnerabilities.

Keywords: exploits, machine learning, patches, security, vulnerability

1 INTRODUCTION

Nowadays, software system is omnipresent. From a machine to the modern human life everything highly reliant on a variety of software. Accordingly, software security becomes a curial concern for the software developers, as an exploited vulnerability not only inculcate extra cost, but also severely disrupt and weaken the targeted organization. It is then essential for software firms to utilize right tools and procedures in order to measure and anticipate the vulnerabilities present in the software. Even with all the efforts attempted to encapsulate the software security, developers still leave some loopholes that can later be exploited. Due to the complexity of software, developing a secure system is a challenging and complex task that usually is a timely task. Besides, software engineers use many methods to find the defects and vulnerabilities, still the most effective ones usually require a lot of time with specialized efforts for the removal. The impact of severity caused by a software vulnerability depends on various attributes, such as exploitability likelihood, authentication type, and attack surface.[1] To find the vulnerabilities, many techniques such as testing and code inspection are well accepted method, but these methods have their own limitations. Although software testing is easier and economical to apply, but it may result in developing its own code irrespective of its system code. Similarly, the code inspection technique may be viewed very effective in identifying the bugs, but with a larger artifact, the code inspection and reviews limit its applicability because of its cost intensive nature.

A vulnerability or a flaw when identified in a system often follows a lifecycle that starts with a discovery date to its final exploit date. Upon its discovery, the vendor is first notified and corresponding patch is released within a stipulated time frame suggested by various organization dealing with software security like: CERT, Bugtraq, iDefence, Tippingpoint, etc. After that, a vulnerability is publicly disclosed. Public disclosure has been a staple part of software vulnerability lifecycle, with thousands of new vulnerabilities identified and revealed each year.[2] In turn, these vulnerabilities are shared via various communication channels, which reaches to software vendors who then analyze their vulnerable system and decide the required action to be taken. Developing a patch early might incur significant manpower and it is costly. However, developing a patch lately may be costly for end users as the probability of exploiting an unpatched vulnerability increases with time. Software firms have the required resources to acknowledge each vulnerability discovered that might impact their users. To get this around system administrators must prioritize their efforts in fixing the most exploitable or problematic vulnerabilities. In order to do so, various metrics are defined to assess the risk of a vulnerability quantitatively such as confidentiality, integrity, authentication, and many others. To measure the scores of identified vulnerabilities, a common vulnerability scoring system (CVSS) is used that is governed by Forum of Incident Response and Security Teams (FIRST) and is a de facto standard across the security community. A CVSS evaluate the vulnerabilities based on various parameters to assess its severity and exploitability. It consists of three metrics: Base, Temporal, and Environmental. Each metric produces a numeric score between 0 and 10 and together when combined reflects the values used to derive the score. The base metric signifies those characteristics of a vulnerability that remain same with respect to the user environment and time. The temporal metric reflects those features that change over time but not with the user environment. The environmental metric as its name suggests represent the characteristics that are unique to a particular user's environment.[3] In this context, the CVSS measures typically the base score as other metrics are dependent on time and user environment.

The vulnerability types also play an important role in determining the likeliness of a vulnerability discovered that can be exploited. It has been observed in various reports of Open Web Application Security Project (OWASP) that respective types of vulnerabilities such as: SQL Injection, Cross Site Scripting, and Execute Code impact the security risk associated with a software system. Moreover, the software product category also helps in prioritizing the vulnerability mitigation based on its occurrence in any specific categories such as application, operating system, and hardware application.

The major implication of this research is to anticipate the vulnerabilities that might be exploited after being disclosed to the public. In this regard, we explore how the likelihood of vulnerabilities to be exploited is reliant on of various features considered in CVSS, vulnerability types, and of various software classes. This study incorporates the techniques defined from the literature of machine learning that can be helpful in automatically predicting the nature of vulnerabilities being encountered in the software. Our objective is to investigate whether the information published with the discovery of a vulnerability can be used to determine if a software system is likely to have an exploitable vulnerability.

In Section 2, we review some related work exist in the vulnerability prediction modeling. Section 3 deals with the building block of our paper that provides the background on CVSS and other metrics and provides an overview of the exploit prediction framework in Section 4. In Section 5, we provide a brief description of data considered in prediction of an exploit-prone vulnerabilities. In Section 6, we conclude the results obtained for exploit-based vulnerability prediction. Finally, we conclude our work followed by the references.

2 RELATED WORK

Predicting cyber exploits has received an increasing attention in the domain of cybersecurity. However, there exists a little work in this line of research as related to the works proposed for predicting vulnerabilities in a software system.[4-7] Of late, this side of coin has been given importance in order to help the vendors to prioritize their vulnerability patching phenomenon based on the discovered vulnerability's likelihood of exploitation. Many previous studies compelled with only the CVSS score as an indicator of predicting the exploit proneness of a vulnerability. Bozorgi et al[8] considered a model that gather features from a database namely Open Source Vulnerability Database (OSVDB) that is now discontinued to predict the exploits based on the Proof of Concepts (PoCs) availability. In the work of Allodi and Massacci[9] and Edkrantz and Said,[10] 73% of vulnerabilities were recorded as exploited as compared to ones recorded in the literature. Later, Sabottke et al[11] developed an exploit prediction model using a dataset acquired from Twitter having links to CVE-IDs and from Symantec threat signatures for the positive labels. Edkrantz and Said[10] also predicted cyber exploit using the CVSS scores and parameters and claimed a high prediction accuracy as compared to previous predictive studies. And, further predicted the exploited vulnerability using the CVSS metrics and their attributes using the Naïve Bayes, Support Vector Machine, and Random Forest as the prediction techniques and reported only the accuracy criteria of their study on exploit prediction.[10] As far as the accuracy is concerned Support Vector Machine performed as the best prediction technique with a score of 83.68%. Of late, Almukaynizi et al[12] deduced a model that considers data from various sources to predict the likelihood of exploitation and claimed it a highly effective approach for the exploit that could be seen in the wild. They also considered CVSS attributes acquired from National Vulnerability Database NVD and exploit details from three repositories Exploit Database (EDB), Zero Day Initiative (ZDI), and Dark Web (DW) to predict the likelihood of exploitation. They considered Random Forest, Support Vector Machine, Bayesian Network, Logistic Regression, Decision Tree, and Naïve Bayes techniques for the prediction. They compared the classifiers with respect to precision, recall, and F1-measure and conclude that the Random Forest perform best with the value of Precision and F1-measure to be 0.45 and 0.40, respectively.

3 BUILDING BLOCKS OF OUR WORK

This section provides the background on how different types of features can be hypothesized to affect exploit-proneness of a vulnerability.

3.1 The common vulnerability scoring system

To assist security administrators, vulnerability disclosures typically includes information for assessing a vulnerability's severity qualitatively or quantitatively. It includes CVSS metrics and various parameters for each CVE disclosed. The CVSS is designed to measure the severity of a software by utilizing various characteristics that affects a vulnerability. It comprises some factors that produce a numeric base score rating from 0 to 10 that reflects a qualitative severity rating. The base score is computed in terms of Impact and Exploitability factors using the following equation:

$$\text{Base Score} = \left(\frac{3I}{5} + \frac{2E}{5} - \frac{3}{2}\right) \times 1.176. \tag{1}$$

We further note that the impact and exploitability score are themselves calculated based on various set of characteristics a vulnerability have had when it is discovered. The impact metric measures the significant consequences of an exploited vulnerability and the exploitability metric measures the conditions required to exploit a vulnerability, and a detailed description of these metrics can be found in Mell et al.[3] Critical vulnerabilities are considered to be the highest priority flaws. These signifies exposures, if exploited, could lead to consequences that affect IT assets. These flaws should be addressed immediately owing to the significant threat on the software. It has been observed in CVSS guide that the critical vulnerabilities are marked with high and medium risk levels. These risk levels are set by vulnerability bulletin analysts, security product vendors, or application vendors and are dependent on various factors considering a vulnerability being successfully exploited. Several factors are evaluated when a given vulnerability is measured critical: an attacker's ability to remotely exploit a flaw, obtaining privileges after a successful attack, and the degree of loss of confidentiality, integrity, and availability. Thus, considering the severity index along with the impact and exploitability metrics may help in predicting the likelihood of an exploit proneness of a vulnerability.

3.2 Vulnerability types and software classes

In addition to classifying vulnerabilities by their severity,[23] another important characteristic that is relevant in predicting the exploit-proneness of a vulnerability is by classifying them by their types.[24] An exploit comes in all forms and extents, but it takes advantage of a weakness that is occurred more often than others.[21, 22] A software exploit can be classified by the expected consequences of an attack on a target software, such as code execution, cross site scripting, overflow, SQL injection, or some other types of vulnerabilities. Some of the most common vulnerabilities occurred in major operating system includes denial of service, code execution, overflow, memory corruption, and gain information. Similarly, application software includes SQL injection, cross-site scripting, and cross-site request forgery types of vulnerability attacks. This study focuses 13 major types of vulnerabilities that can be used in attacking a target software.

Furthermore, different types of software products like application software, operating systems, and hardware applications have been considered such that any occurrence of an exploit can be classified based on the type of attack on any respective software product.[17] Although exploiting a software can occur in variety of ways, one such process can be done using the malicious websites. Such attacks typically target software applications containing vulnerabilities in unpatched web browsers or plug-ins. Hence, in view of the type of attack on a target software class, the security managers can be benefited, in predicting, whether the vulnerability can be exploited or not.

4 OVERVIEW OF EXPLOIT PREDICTION FRAMEWORK

Software vulnerabilities are exploitable flaws that pose a substantial security risk. To anticipate the vulnerabilities in a software system there are mainly two approaches. First, count-based procedures emphasis on predicting the number of vulnerabilities present in a software system over a specified time horizon. Software managers may find them useful in understanding the security trends, allocating security resources, or expected to find the lurking vulnerabilities. Another approach deals with the classification-based methods to predict the entities that are exploit prone. The prediction based on the classification of vulnerabilities can assist managers in directing their resource allocations to the exploit prone vulnerability.

In the current work, we consider vulnerability prediction as a classification problem – that predict whether a vulnerability discovered is exploit prone or not. In our introductory investigation, we have observed that a very small proportion of vulnerabilities are exploitable. But a formal verification and validation may account a stringent security measures that would also be resourceful in facilitating the area most likely to be exploitable. Thus, these areas require most of the attention during the Software Development Life Cycle (SDLC).[18] The methodology uses binary classification approach to predict whether a vulnerability is exploitable or not. For example, we can draw a correspondence between our exploitation prediction approach and the weather prediction. Based on the previously known data accounting in raining a particular day, a classification approach can be deduced to predict will it likely to be rain today with a probability between zero and 100%.

This paper presents a framework that describe how various characteristics such as CVSS metrics, vulnerability types, and product type can be used to predict the exploit-prone vulnerabilities, as illustrated in Figure 1. In the first step, we map each discovered vulnerability with its associated CVSS metric and linked with its related vulnerability type and with the type of software product it is affecting. This mapping is the base step for setting up each vulnerability to predict its exploitability status. For mapping the metrics to vulnerabilities, we find out the respective features and the exploitability status a vulnerability has had in past. Thereby, we extract vulnerability summary from advisories, and tracing the linked entries in exploit repositories via analyzing the common vulnerabilities and exposures (CVE) entries. The exploit status helps us in determining the characteristics a vulnerability pertain in a software that can be exploited. Then, these characteristics of each discovered vulnerabilities can be used to construct and train an exploit predictor. Machine learning algorithms allow us to anticipate vulnerability exploits by considering newly discovered vulnerability entries and generates the probability that it would be exploited or not. As per the calculated probability, making an early assessment by predicting the exploit prone vulnerabilities can then help security managers to schedule resources in terms of deployment of patches.

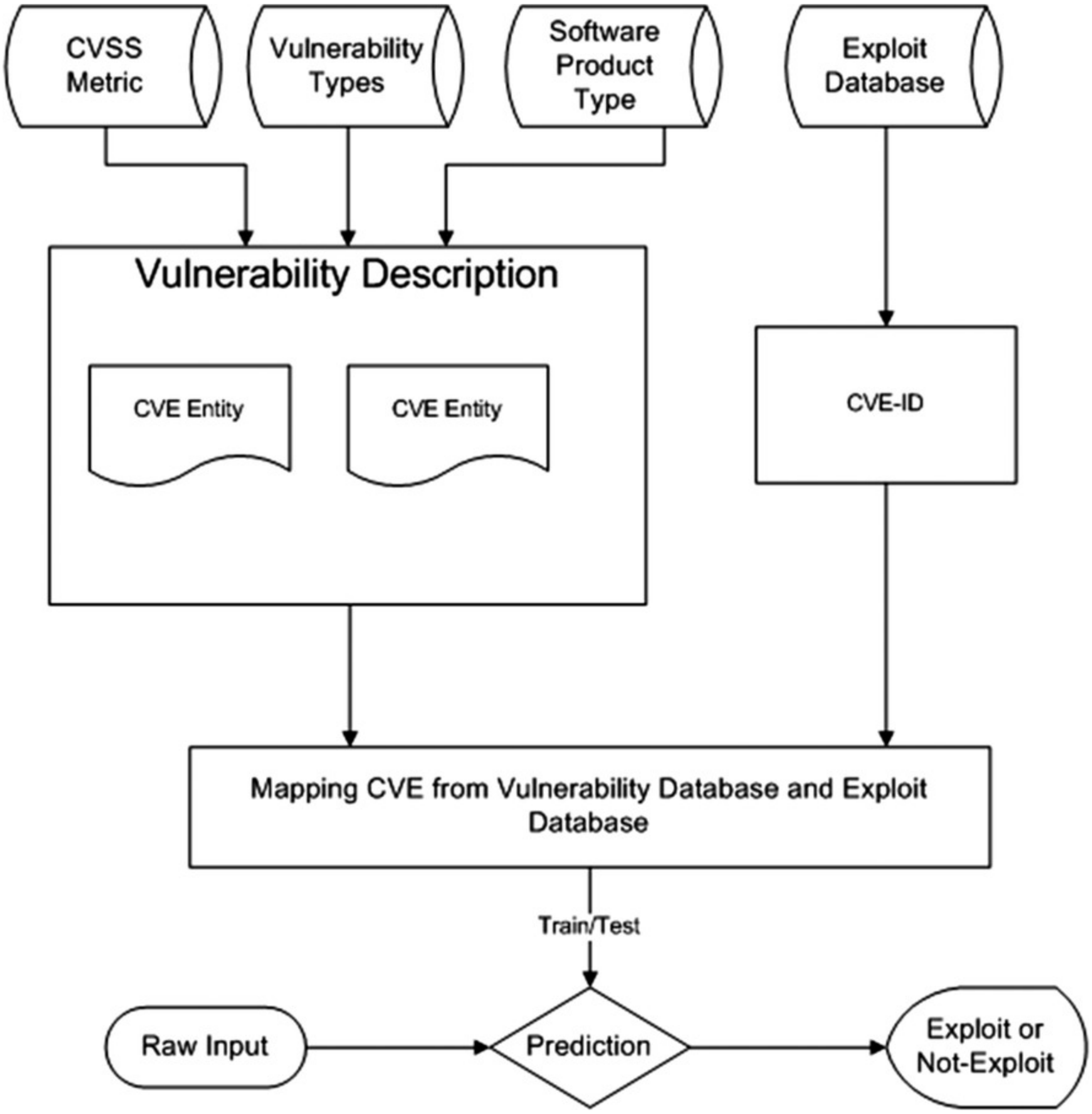


FIGURE 1. Exploit Prediction Framework

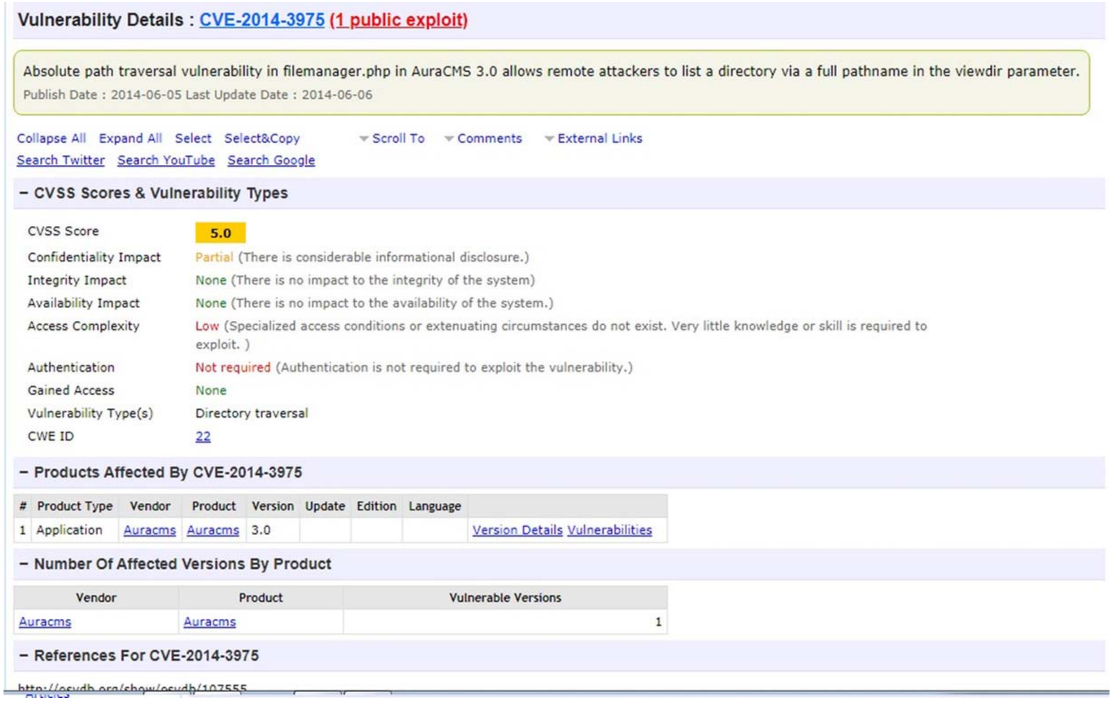
Our main challenge is to discover an algorithm that optimize the performance of our binary classification. In the field of machine learning there exist a theorem named “No Free Lunch theorem,” which states that there is no one model that will always works best for every problem. In this context, we use several machine learning and statistical techniques to examine the exploit predictor such that the inferences drawn from the outcomes are not overly impelled by any specific method. The results allow us to see if some vulnerability characteristics are more likely to be exploited or not. Lastly, we rank all the algorithms based on their performance measures by considering a statistical technique weighted criteria-based ranking method.

5 PREDICTING EXPLOITS

This section describes how to predict exploit-prone vulnerabilities in a software as sketched by the framework presented in Figure 1. To empirically evaluate the framework, we conduct an experiment using the National Vulnerability Database (NVD) to predict its exploit-prone entities. This section begins with information retrieval process, with an overview of the data used. Then, we explain the endogenous and exogenous variables of the exploit-predictor. This is followed by a detailed description of the steps in the exploit-prediction framework applied on the NVD. First, we describe how we map the vulnerability characteristics back to NVD entries and then linked the respective exploit status from the Exploit Database (EDB) to each entry. Then, we specify how we build a feature matrix along with the corresponding labels required to run the machine learning algorithms be used in predicting the exploit-predictor.

5.1 Vulnerability databases

The data considered in this study were extracted from different sources. We used two well-known online repositories, NVD[19, 20] and EDB.[19, 20] The NVD consist of information for all Common Vulnerabilities and Exposures (CVEs). With an approximate CVEs base of more than one hundred thousand entries, it is one of the most informative repositories available. Moreover, each CVE is connected to external links citing to bug trackers, exploits, vendor advisories, etc. Additionally, each CVE entry contains CVSS metrics, parameters, and a small text description, which can be found in Figure 2. Each CVE entry is assigned with a CVE-number of format CVE-Year-Number, denoting the year in which vulnerability is disclosed and its assigned number.



Vulnerability Details : CVE-2014-3975 (1 public exploit)

Absolute path traversal vulnerability in filemanager.php in AuraCMS 3.0 allows remote attackers to list a directory via a full pathname in the viewdir parameter.
Publish Date : 2014-06-05 Last Update Date : 2014-06-06

– CVSS Scores & Vulnerability Types

CVSS Score	5.0
Confidentiality Impact	Partial (There is considerable informational disclosure.)
Integrity Impact	None (There is no impact to the integrity of the system)
Availability Impact	None (There is no impact to the availability of the system.)
Access Complexity	Low (Specialized access conditions or extenuating circumstances do not exist. Very little knowledge or skill is required to exploit.)
Authentication	Not required (Authentication is not required to exploit the vulnerability.)
Gained Access	None
Vulnerability Type(s)	Directory traversal
CWE ID	22

– Products Affected By CVE-2014-3975

#	Product Type	Vendor	Product	Version	Update	Edition	Language	
1	Application	Auracms	Auracms	3.0				Version Details Vulnerabilities

– Number Of Affected Versions By Product

Vendor	Product	Vulnerable Versions
Auracms	Auracms	1

– References For CVE-2014-3975

http://osvdb.org/show/osvdb/107555

FIGURE 2. An Example of a CVE Entry

The NVD dataset also includes various parameters that influence the impact and exploitability metrics like confidentiality, integrity, availability, complexity, and authentication. These parameters are used in order to calculate the impact and exploitability score for any CVE. Further, the NVD also includes information of different affected software product type, vendor, product name, and version. A typical CVE may affect application software and operating system, and moreover a specific range of versions. There are many links to external pages related to exploit databases, advisories, and other information. The EDB is a curated repository for exploits and proof-of-concept code to 40,000 vetted software exploits. The general problem with the NVD is that it does not keep all details about vulnerabilities and exploits. The data is partial in some specific area; for example, out of >100,000 vulnerabilities, it contains only 4333 links to EDB. However, as per the EDB record for every exploit entry, there are >20,000 references back to CVE-numbers. The integrated nature of the repositories enables us to accurately map vulnerabilities to their exploits via the CVE-number.

To validate this study, we have extracted vulnerability-exploit information disclosed in years 2012-2015. Over the aforementioned period, 2203 (8%) of the total 28,158 vulnerabilities have had exploits. In total, these 2203 vulnerabilities have proof-of-concepts available for their exploitation. However, the EDB listed proof of concepts of 6197 exploits from year 2012 to year 2015. But, only 2203 exploits have links related to the CVE-number. Figure 3 provides a pictorial representation of number of vulnerabilities or exploits per year.

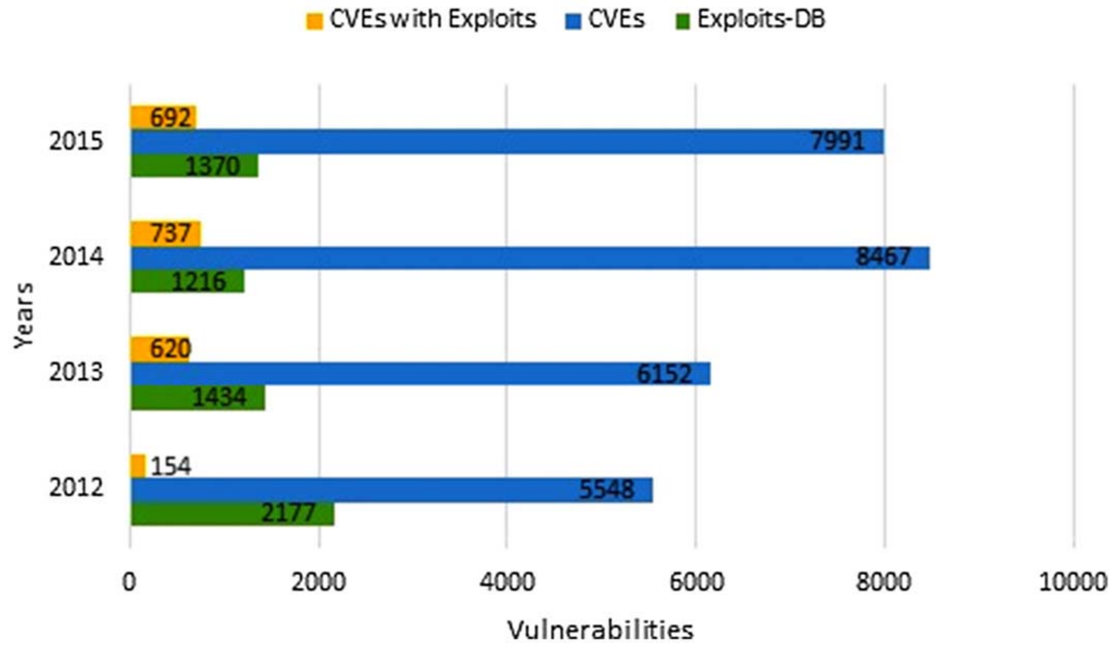
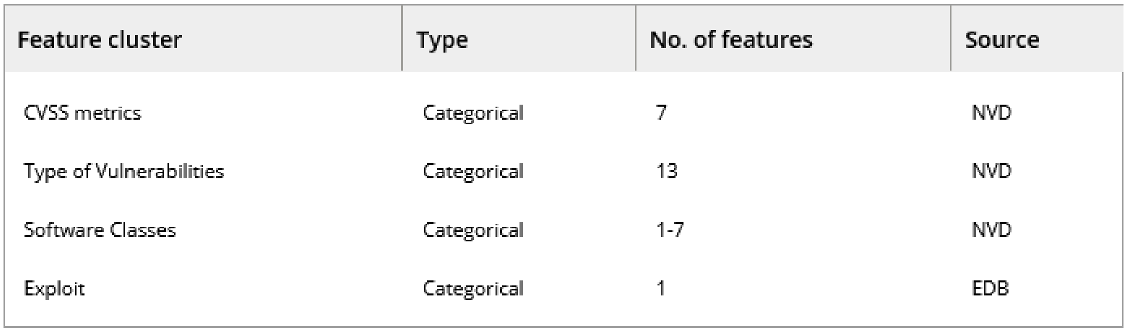


FIGURE 3. Number of Vulnerabilities and Exploits per Year

5.2 Building the feature matrix

One of the most important tasks in machine learning algorithm is selecting a good feature space. The data extracted from different sources are required to be transformed into a feature matrix, to be used for the prediction. In the feature matrix, each row constitutes a vulnerability entry and each column represents a feature dimension. Each feature dimension is represented as an exogenous variable that is used to make a prediction and the corresponding exploit label is termed as endogenous variable about which we make a prediction in our study. The feature space is presented in Table 1.

TABLE 1. Feature Space



Feature cluster	Type	No. of features	Source
CVSS metrics	Categorical	7	NVD
Type of Vulnerabilities	Categorical	13	NVD
Software Classes	Categorical	1-7	NVD
Exploit	Categorical	1	EDB

The independent variables are the CVSS metrics that comprise impact and exploitability attributes, severity level, vulnerability types, and product category. The fundamental hypothesis of our study is identifying the exploit-proneness of a vulnerability that may be affected by its underlying characteristics.

5.3 Extracting vulnerability information from NVD

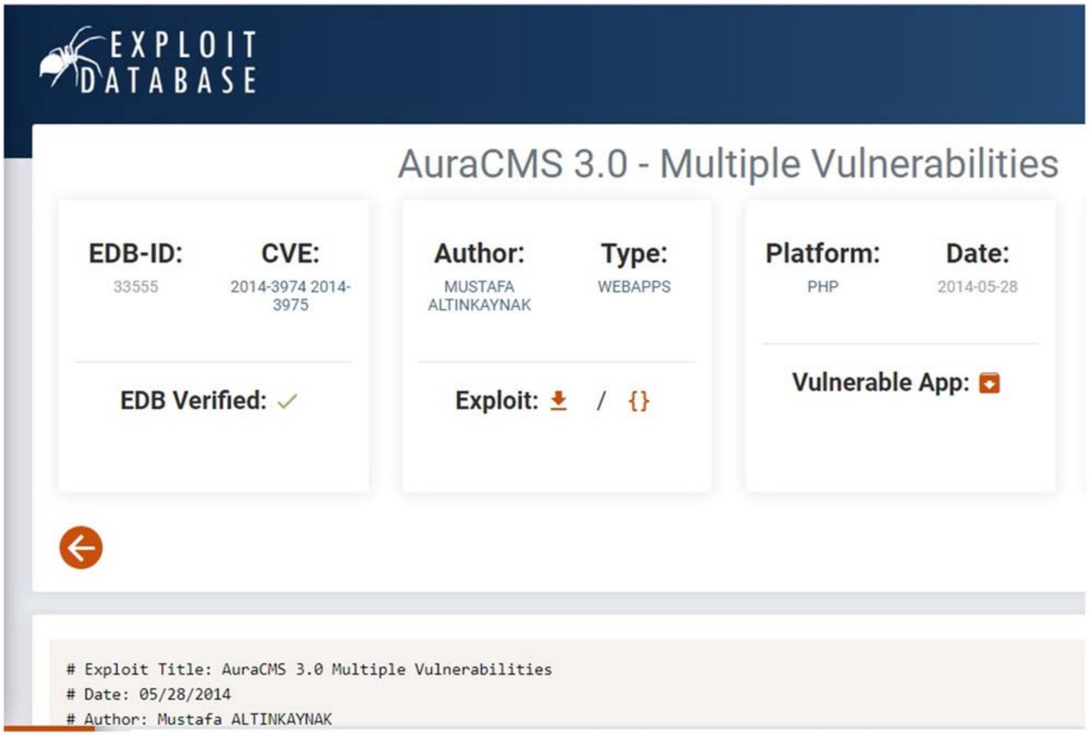
Vulnerabilities are security issues that are disclosed in advisories that help users in avoiding security problems by implementing workarounds themselves. For example, CVE lists publicly disclose information about security vulnerabilities from the NVD, which maintain CVE data feed available in JavaScript Object Notation (JSON) format. Some of the features considered in our study, such as base score, severity, impact, and exploitability metric, are available in structural form. However, other concerned features, such as types of vulnerabilities and the affected type of software products, are not mentioned in the NVD.

Apart from the structural data, NVD also contains unstructured information in the form of text description for each vulnerability. The text description field is examined with tuple of words that relate them to specific type of vulnerability that is present in a software. As mentioned in the Figure 2, the text description refers to absolute path traversal like common words that relates a vulnerability to any specific type. We consider 13 major classes of vulnerabilities that are used in exploiting any software. Each CVE entry is assigned with a binary label under the feature dimension of the respective types of the vulnerability. For example, CVE-2012-4886 lists two different execute code and overflow type of vulnerability. Therefore, the feature dimension belonging to execute code and overflow vulnerability type in the feature matrix has been assigned.

On similar lines, we utilized the information of the products affected by any CVE to classify the type of a product *viz* application software, operating system, or hardware application, a vulnerability is affecting. It has also been observed that a vulnerability might have many types and may affect different types of software product. The type of product a vulnerability is affecting is classified into different classes, viz application software, operating system, hardware application, and a collection of all multiple types. Based on a vulnerability affecting a software, the feature dimensions are assigned. For example, if a CVE is affecting both application and operating system product type then, the feature dimension corresponding to application and operating type is assigned.

5.4 Mapping vulnerabilities to exploits

To map vulnerabilities to exploit, we need exploitability status of a vulnerability. Since, there is no parameter explicitly mentioned in the NVD about the exploit information of a vulnerability, what can be found are some links to exploit information. Moreover, most of the vulnerabilities acknowledged in NVD have missing links to exploits in the EDB. Hence, due to the asymmetrical relation between EDB and NVD; we consider CVE-number in cross referencing their exploit status with each vulnerability. A vulnerability can be classified as exploited if it had a linkage via CVE-number extracted from the EDB. Similarly, if a vulnerability does not pose any information in the EDB, we label it as unexploited vulnerability. A snapshot of EBD page is shown in Figure 4, where a detailed description of an exploit referencing a CVE-number and some proof-of-concept code of a vulnerability is provided.



EXPLOIT DATABASE

AuraCMS 3.0 - Multiple Vulnerabilities

EDB-ID:	CVE:	Author:	Type:	Platform:	Date:
33555	2014-3974 2014-3975	MUSTAFA ALTINKAYNAK	WEBAPPS	PHP	2014-05-28

EDB Verified: ✓

Exploit: ⬇ / {}

Vulnerable App: ⬇

```
# Exploit Title: AuraCMS 3.0 Multiple Vulnerabilities
# Date: 05/28/2014
# Author: Mustafa ALTINKAYNAK
```

FIGURE 4. A Screenshot of Exploit Database with CVE-number

Once we have the CVSS metrics and vulnerability characteristics for each CVE entry in NVD, we can use this information to predict the exploit-proneness of a vulnerability. The next section presents a detailed illustration of the results employed in the prediction of an exploit.

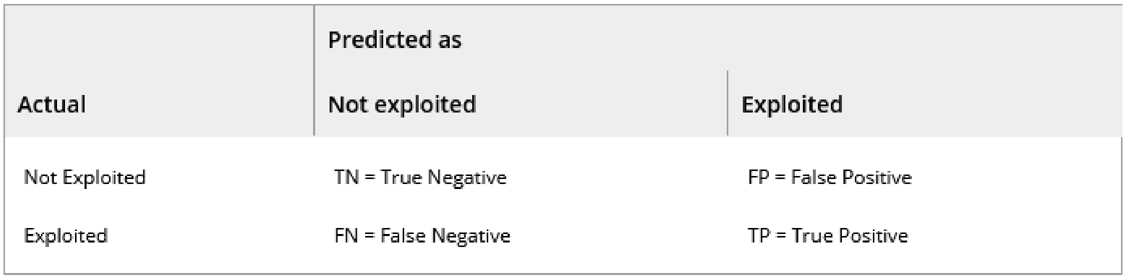
6 RESULTS AND DISCUSSION

So far, we have discussed various features required in predicting exploit proneness of a vulnerability. For our binary classification, the main challenge is to identify some algorithm and a feature set that will provide an optimal classification performance. To numerically evaluate the likelihood of exploitation, we have employed five machine learning algorithms namely Naïve Bayes, Logistic Regression, Support Vector Machine, Decision Tree, and

Random Forest. For implementing algorithms, WEKA tool is implemented. WEKA is an efficient tool for machine learning and data analysis. A brief introduction to WEKA can be found in Witten et al.[13]

We also need to know the performance of the model based on various algorithms. The most often used criterion to evaluate the performance are accuracy, precision, recall, false positive rate, and false negative rate. In our case, the two-class problem (exploit prone or not exploit prone), these performance criterions can be derived with the help of confusion matrix, shown in Table 2.

TABLE 2. Confusion Matrix



	Predicted as	
Actual	Not exploited	Exploited
Not Exploited	TN = True Negative	FP = False Positive
Exploited	FN = False Negative	TP = True Positive

The confusion matrix relates the actual versus predicted results where:

- True Negative (TN) represents the number of vulnerabilities predicted as not being exploit-prone where no exploit is occurred in those vulnerabilities.
- False Positive (FP) represents the number of vulnerabilities wrongly predicted as exploit-prone.
- False Negative (FN) denotes the number of vulnerabilities predicted as not being exploit-prone but rather turn out to have an exploit.
- True Positive (TP) represents the number of vulnerabilities predicted as exploitable that actually are exploitable.

6.1 Performance metrics

To evaluate the predictive performance, several measures such as accuracy, recall, precision, F-measure, false positive rate, false negative rate, and mean absolute error can be calculated as follow:

- Accuracy: It represents overall classification rate. It can be defined as the ratio of total number of vulnerabilities correctly predicted to the total number of vulnerabilities.

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \qquad (2)$$

- Precision: It is known as correctness; it measures the efficiency of prediction. It can be defined as the proportion of number of vulnerabilities correctly predicted as exploit prone to the total number of vulnerabilities predicted as exploit-prone.

$$Precision = \frac{TP}{TP+FP}. \qquad (3)$$

- Recall: Recall can be defined as the ratio of number of vulnerabilities correctly predicted as exploit-prone to the actual number of vulnerabilities as exploitable. Recall represents the exploit detection rate which allow us to quantify the effectiveness of prediction. It is given as:

$$Recall = \frac{TP}{TP+FN} \tag{4}$$

To fully evaluate the performance measures, both precision and recall are considered. The higher the recall, fewer exploitable vulnerabilities go undetected. And, higher the precision, a less amount of effort is wasted in inspection and testing. There exists a trade-off between precision and recall. It can be described as: If we predict only one vulnerability as exploit-prone and this vulnerability is in fact exploitable, then we will be having a 100% precision. However, if there exist other exploitable vulnerabilities then the recall will be low. In another case, if we predict all vulnerabilities as exploitable, then the recall would have 100% effectiveness but its precision would be significantly low. Hence, a measure is required that takes both false positives and false negatives in a single efficiency measure.

- F-measure: F-measure is a measure of test's accuracy and it is a weighted average of precision and recall. To obtain a trade-off between precision and recall, it is expressed in terms of percentage such that it reaches it best and worst value at 100 and 0, respectively. It is represented as given in Equation (5).

$$F_\beta - measure = \frac{1 + \beta^2 \times Precision \times Recall}{(\beta^2 \times Precision) \times Recall}. \tag{5}$$

Here, F_β measures the prediction effectiveness with respect to a user who attaches β times importance given to recall as precision. The tradition F-measure provide importance to both recall and precision and is denoted by $F_1 - measure$ We consider that it is more important to classify exploitable vulnerabilities, even at the cost of incorrectly predicting non-exploitable vulnerabilities as exploit-prone. It is because a single exploitable vulnerability may lead to serious loss and damage. In this context, we have assigned more weight to recall than precision. Therefore, we consider $F_2 - measure$ as it provides twice weight to recall in comparison to precision, in order to evaluate prediction.

- FP and FN rate: The FP rate and FN rate are defined in Equations (6) and (7), respectively. A high FN rate indicates that there is a risk of overlooking vulnerabilities, whereas a high FP rate indicates effort may be wasted in investigating the predicted exploit-prone vulnerable entities.

$$FP\ rate = \frac{FP}{FP+TN}. \tag{6}$$

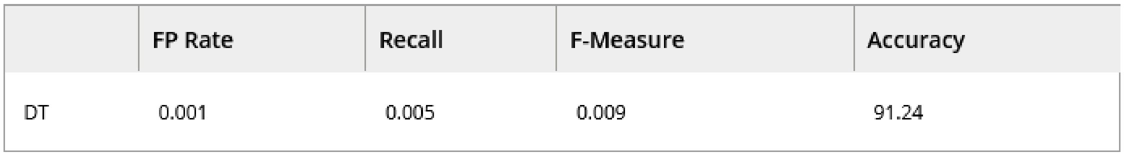
$$FN\ rate = \frac{FN}{TP+FN} \tag{7}$$

- Mean absolute error (MAE): MAE is the average of the absolute differences between the predicted and actual values. It is used for summarizing and assessing the quality of a machine learning model.

6.2 Unequal dataset

To evaluate the prediction capability, vulnerability data extracted from the year 2012 to year 2015 have been used. There are 2203 instances of exploitable vulnerabilities (minority category) as opposed to 25,955 non-exploitable vulnerabilities (majority category) in the obtained data set. We further remove those instances that have missing information, and finally the data set represents 2164 instances of exploitable vulnerabilities and a total of 20,438 instances of non-exploitable vulnerabilities. If we train a classifier on such an imbalanced dataset, it would lead to a biased prediction toward the majority class and cause overfitting.[13] We train a classifier on such a highly imbalanced data set. For the sake of showing the results on such a highly imbalanced data set we consider Decision Tree (DT) technique as shown in Table 3. Since, the predictor is biased towards the majority class and it predicts many exploitable vulnerabilities as non-exploit. Thereby missing many exploitable vulnerabilities. It can be represented by a lower recall rate of 0.5%. Naturally, the FP rate would also be low as it is rarely a vulnerability would be predicted as exploit-prone. Moreover, the accuracy of prediction is also misleading of 91.24%. Even with a low FP-rate and a high accuracy of prediction, such a biased predictor is unusable as it would miss a majority of exploitable vulnerabilities.

TABLE 3. Performance Parameters for Imbalanced Dataset



	FP Rate	Recall	F-Measure	Accuracy
DT	0.001	0.005	0.009	91.24

In order to facilitate building of an unbiased predictor, a balanced subset of 4328 instances has been created that consists of 2164 instances of exploitable vulnerabilities and a random selection of 2164 instances representing non-exploitable vulnerabilities. In many prior studies, under sampling of majority category has been performed to obtain a balanced set.[14, 15] With the balanced dataset, the proportion of exploitable vulnerabilities and non-exploitable vulnerabilities is accurately 50%. Therefore, a prediction is likely to be useful, if it correctly classifies a vulnerability to be exploited or non-exploited >50% of the time.

6.3 Vulnerability types and their exploitability

To analyze the effect of types of vulnerabilities identified in a particular software class, we try to segment the vulnerability types based on their severity identified in each software class. It is easy to come up that some specific types of vulnerabilities have a higher likelihood of getting exploited in a particular software class. Figure 5 represents a visualization of different types of vulnerabilities exploited and not-exploited in a various software class.

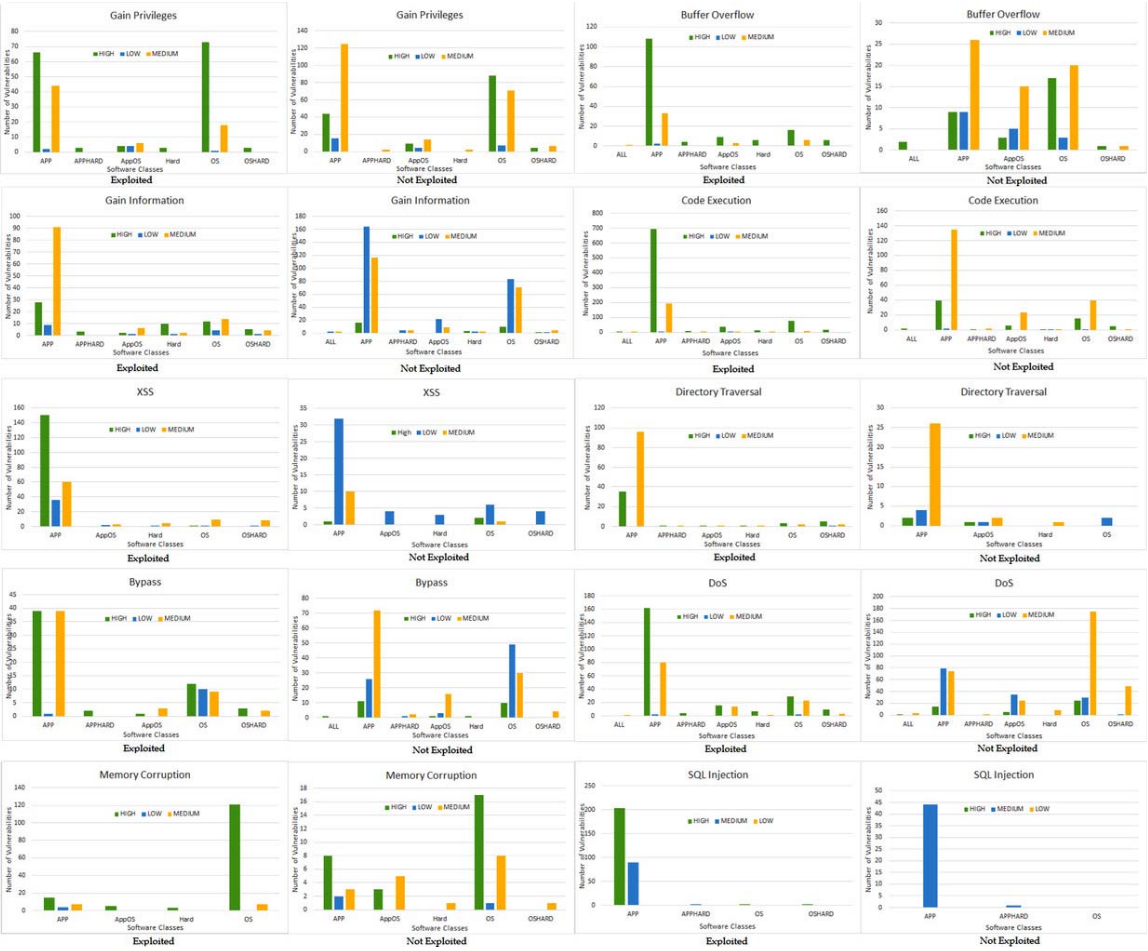


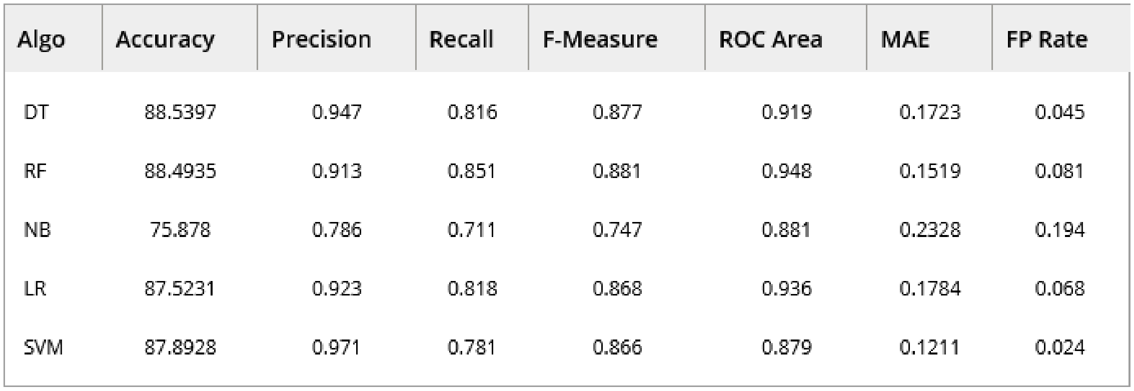
FIGURE 5. Distribution of Types of Vulnerability

It can be seen from the figure that vulnerabilities belonging to type gain privileges are more exploitable in case of vulnerabilities that occur in an operating system. However, an equal count of vulnerabilities also impacted the application software but a high severity index vulnerability is more exploitable. In case of gain information type, majority of vulnerabilities occurs in non-exploitable class that may suggest that vulnerability of this type might not get exploited but the vulnerabilities encountered in application software with medium severity level have an equal chance of exploitation and not. The cross site-scripting (XSS) type vulnerabilities impact application software more and it can be seen that vulnerabilities belonging to high severity index are more exploitable prone as compared to the low index vulnerabilities. A vulnerability of type bypass in application software with a high severity index has a higher chance of exploitation as compared to its counterpart, although in operating systems the number of counts of bypass vulnerability is more in non-exploited vulnerabilities as compared to exploited ones. The memory corruption flaws exist more in operating systems and a large share of vulnerabilities of high severity index are exploitable as compared to the application software. In case of buffer overflow type, application software of high severity index is more prone to exploits as compared to low and medium severity index. The vulnerabilities of type code execution occur more in application software and it can also be seen in Figure 5, high severe vulnerabilities of code execution type are more exploit-prone in both operating system and application software. Similarly, SQL injections also occur more in application software with a higher rate of exploitation for high severe vulnerabilities. But, in case of denial of service (DoS) type, the operating systems collect a higher count of vulnerabilities that are not exploitable. However, those vulnerabilities discovered in application software are more exploit-prone. Hence, we can advocate that vulnerability type and the software class in which the vulnerabilities have been occurred influences the exploitation process significantly. Also, the exploit count of vulnerabilities is more in application software as compared to the operating system. But a higher severe vulnerability discovered in an operating system is more exploit-prone than a low or medium level. In the next section, we examine the results obtained to predict the exploit-proneness of a vulnerability considering the CVSS metric, vulnerability types, and software classes.

6.4 Prediction performance of different classifiers

We have considered different machine learning algorithms – Naïve Bayes (NB), Support Vector machine (SVM), Logistic Regression (LR), Decision Trees (DT), and Random Forest (RF) to predict the exploit-proneness of a vulnerability and have compared their predictive performance. To obtain the results, we have considered 10-fold cross validation technique to test the classifiers in order to reduce the variability in the prediction. It is a technique used for accessing the predictive performance of a model by randomly spliting the dataset into 10-folds of equal size and each fold is used as testing at any giving iteration. Here, we have used stratified sampling such that an equal proportion of exploitable and non-exploitable vulnerabilities present in each fold during the random splitting. Finally, we calculate the performance metrics obtained using different technique and is presented in Table 4.

TABLE 4. Prediction performance of different techniques



Algo	Accuracy	Precision	Recall	F-Measure	ROC Area	MAE	FP Rate
DT	88.5397	0.947	0.816	0.877	0.919	0.1723	0.045
RF	88.4935	0.913	0.851	0.881	0.948	0.1519	0.081
NB	75.878	0.786	0.711	0.747	0.881	0.2328	0.194
LR	87.5231	0.923	0.818	0.868	0.936	0.1784	0.068
SVM	87.8928	0.971	0.781	0.866	0.879	0.1211	0.024

From Table 4, we would like to emphasize some points that: first, mostly all the techniques are predicting >75% accuracy. This validate the usefulness of using vulnerability types and the software classes in predicting the exploit-proneness of a vulnerability, irrespective of what learning methodology is considered. Second, we are able to predict 85% of exploit-prone vulnerabilities having an overall accuracy of 88% in case of Random Forest classification.

For a detailed analysis of different prediction models, various measures such as accuracy, recall, FP rate, and F_1-measure are compared. The performance measures of a prediction technique having a higher value is considered better in terms of accuracy, recall, and F_1-measure. And, as far as FP rate and MAE is considered smaller value is considered for a good predictor. From Table 4, it can be observed that Decision Tree and Random Forest performing best when overall accuracy is considered. The recall measure for the classifier Random Forest outperforms all other techniques, thus in our case Random Forest techniques can be considered as an efficient technique in predicting a maximum number of exploit-prone vulnerabilities. However, many exploit-prone vulnerabilities remain unnoticed if the prediction is done using Naïve Bayes technique. If we consider Precision attribute the SVM and Decision Technique can be considered as the best classifier. Similarly, the false positive rate criteria for the SVM and Decision Tree criteria will likely to raise fewer false results as compared to other techniques.

As mentioned earlier, there exists a trade-off between recall and precision. If we observe the F_1-measure score, Decision Tree and Random Forest maintains a higher balance between recall and precision as compared to other techniques. It is also important to observe how significantly different techniques predict the exploit-prone vulnerabilities while giving more emphasis to recall by examining the F_2-measure. Figure 6 presents a comparison of F_2-measures of different techniques, with Decision Tree and Random Forest showing the best performance. Hence, to predict a higher percentage of exploitable vulnerabilities, then both Decision Tree and Random Forest are likely to be preferred.

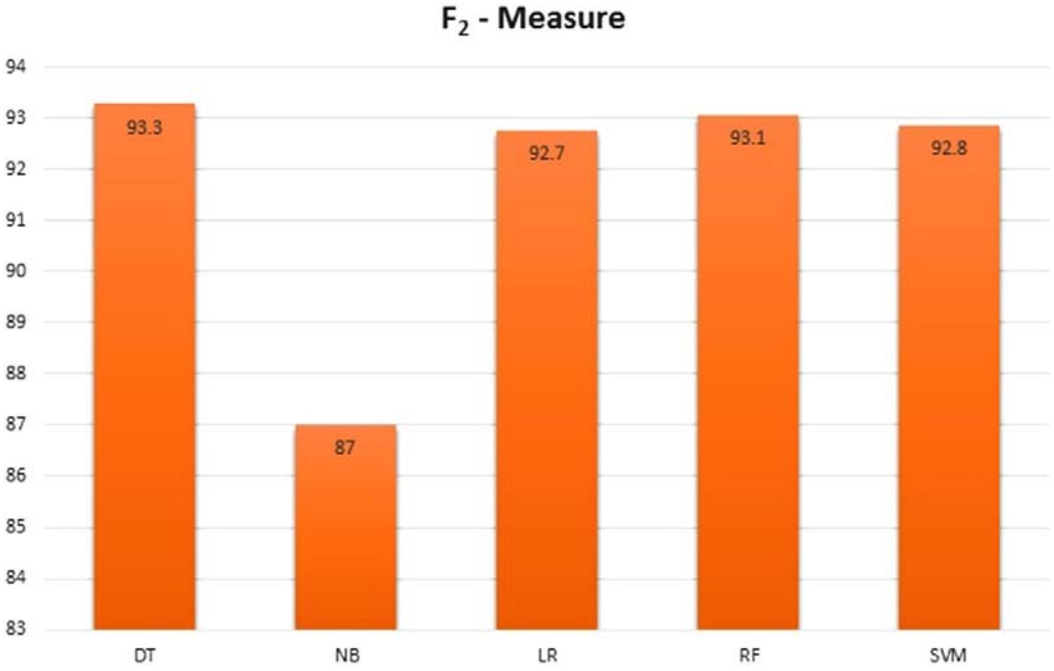


FIGURE 6. Comparison of F_2-measures of Different Techniques

Moreover, we investigate the trade-off between recall and false positive rate by examining the Receiver Operating Curve (ROC). ROC is often used to analyze the performance of a predictor in determining a positive label or a true class. Figure 7 presents the ROC curves of all different techniques considered for predicting the exploit-proneness of a vulnerability. A predictor achieving a higher true positive rate with a low false positive rate is considered as a good predictor. This can be interpreted as a technique perform better if its ROC curve lies above that of another technique. As per Figure 7, the Random Forest technique is performing best for predicting the exploit-prone vulnerabilities since its ROC value is coming out highest as compared to the other algorithms. Moreover, to choose a classifier based on only ROC value does not make sense as other comparison criterion and the value of F_2-measure also plays important role in selecting the best techniques. Hence, it is required to rank the techniques considering all different criterions in order to find out which technique is performing best for detecting the exploit-prone vulnerabilities.

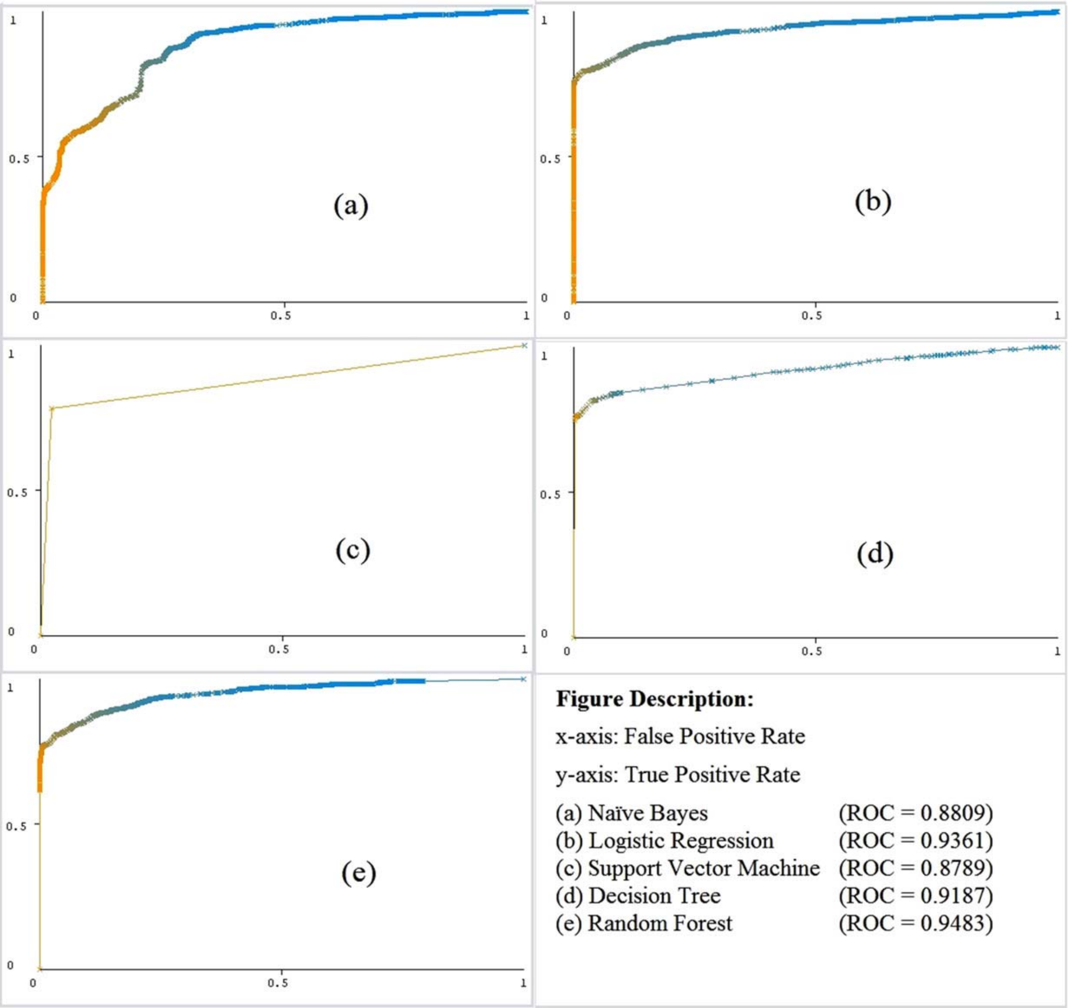
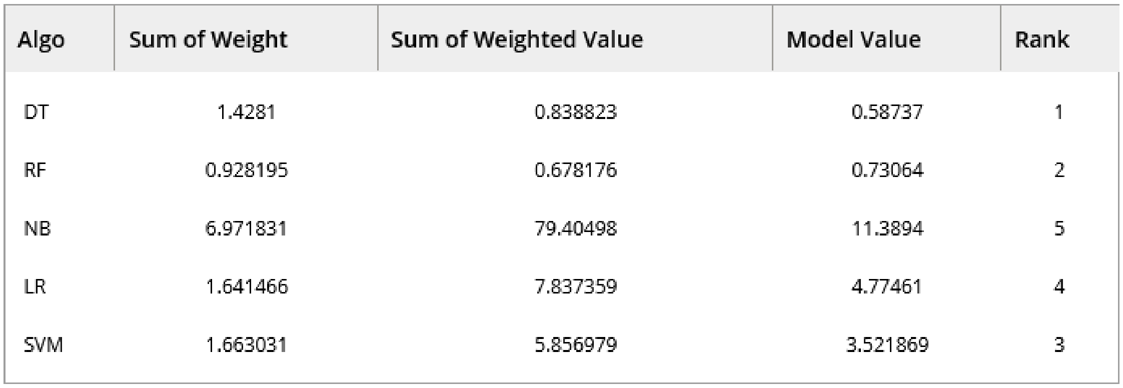


FIGURE 7. ROC curves of Different Techniques

6.5 Weighted criteria based ranking

In this section, we discuss the results obtained using the weighted criteria-based ranking. The steps involved in finding the weights and the ranks are provided in Anand et al.[16] In order to rank different techniques, Table 5 provides an overview of various weights computed using the weighted criteria based ranking method.

TABLE 5. Weighted criteria table



Algo	Sum of Weight	Sum of Weighted Value	Model Value	Rank
DT	1.4281	0.838823	0.58737	1
RF	0.928195	0.678176	0.73064	2
NB	6.971831	79.40498	11.3894	5
LR	1.641466	7.837359	4.77461	4
SVM	1.663031	5.856979	3.521869	3

After applying the weighted criteria value method in determining the ranking for different techniques used in predicting the exploit-prone vulnerabilities, it has been found that Decision Tree is performing best followed by the Random Forest techniques. Hence, considering all the criterions to evaluate the best classifier for the prediction; Decision Tree approach is coming out as the most efficient algorithm in predicting the exploit-prone vulnerabilities.

7 CONCLUSION

In this paper, we evaluate the effectiveness of applying different vulnerability characteristics to predict exploit-prone vulnerabilities discovered in a software system. To analyze, we consider five alternative machine learning techniques that learn from disclosed vulnerabilities. We conduct an empirical study on NVD database that aggregate the early signs a vulnerability poses in determining the likelihood of exploitation that helps in prioritizing the patches. Overall, we are able to correctly predict >75% of exploit-prone vulnerabilities, maintaining a false positive rate <20%. This study shows that the vulnerability types, CVSS parameters, and target software classes can be useful in predicting the exploit-prone nature of a vulnerability. Such predictions will allow software developer to take preventive action while developing a software against the potential threats evolve during the coding.

ACKNOWLEDGMENTS

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors. The authors would like to thank various reviewers who have helped in the proper articulation of the manuscript.

REFERENCES

1. Nayak K, Marino D, Efstathopoulos P, Dumitras T. Some vulnerabilities are different than others. In: A Stavrou, H Bos, G Portokalidis, eds. Research in Attacks, Intrusions and Defenses. Cham, Switzerland: Springer; 2014: 426- 446.
2. The IBM Website, http://www.ibm.com/services/us/iss/xforce/midyearreport. Accessed May 21, 2020.
3. Mell P, Scarfone K, Romanosky S. A Complete Guide to the Common Vulnerability Scoring System. https://www.first.org/cvss/cvss-v2-guide.pdf. Accessed May 21, 2020.

4. Alhazmi OH, Malaiya YK, Ray I. Measuring, analyzing and predicting security vulnerabilities in software systems. *Computers Security*. 2007; 26(3): 219- 228.
5. Anand A, Bhatt N. Vulnerability discovery modeling and weighted criteria based ranking. *J Ind Soc Probab Stat*. 2016; 17(1): 1- 10.
6. Anand A, Das S, Aggrawal D, Klochkov Y. Vulnerability discovery modelling for software with multi-versions. In: M Ram, JP Davim, eds. Advances in Reliability and System Engineering. Cham, Switzerland: Springer; 2016: 255- 265.
7. Bhatt N, Anand A, Yadavalli VSS, Kumar V. Modeling and characterizing software vulnerabilities. *Int J Math Eng Manag Sci*. 2017; 2(4): 288- 299.
8. Bozorgi M, Saul LK, Savage S, Voelker GM. Beyond heuristics: learning to classify vulnerabilities and predict exploits. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, NY: Association for Computing Machinery; 2010: 105- 114.
9. Allodi L, Massacci F. Comparing vulnerability severity and exploits using case-control studies. *ACM Trans Inform Syst Secur*. 2014; 17(1): 1- 20.
10. Edkrantz M, Said A. Predicting cyber vulnerability exploits with machine learning. In: S Nowaczyk, ed. Thirteenth Scandinavian Conference on Artificial Intelligence. Amsterdam, the Netherlands: IOS Press; 2015: 48- 57.
11. Sabottke C, Suciu O, Dumitras T. Vulnerability disclosure in the age of social media: exploiting Twitter for predicting real-world exploits. Proceedings of the 24th USENIX Security Symposium. Berkeley, CA: USENIX Association; 2015: 1041-1056.
12. Almukaynizi M, Nunes E, Dharaiya K. Patch before exploited: an approach to identify targeted software vulnerabilities. In: L Sikos, ed. AI in Cybersecurity. Cham, Switzerland: Springer; 2019: 81- 113.
13. Witten IH, Frank E, Hall MA, Pal CJ. Data Mining: Practical Machine Learning Tools and Techniques. 4th ed. San Francisco, CA: Morgan Kaufmann; 2016.
14. Arisholm E, Briand LC, Fuglerud M. Data mining techniques for building fault-proneness models in telecom java software. 18th IEEE International Symposium on Software Reliability. Trollhattan, Sweden: IEEE; 2007: 215- 224.
15. Kuang L, Zulkernine M. An anomaly intrusion detection method using the CSI-KNN algorithm. In *Proceedings of the 2008 ACM symposium on Applied computing*, Ceara, Brazil: ACM, Fortaleza; 2008: 921- 926.
16. Anand A, Kapur PK, Agarwal M, Aggrawal D. Generalized innovation diffusion modeling & weighted criteria based ranking. In *Proceedings of 3rd International Conference on Reliability, Infocom Technologies and Optimization*. Noida, India: IEEE; 2014: 1- 6.
17. Anand A, Bhatt N, Aggrawal D. Modeling software patch management based on vulnerabilities discovered. *Int J Reliab Qual Saf Eng*. 2020; 27(02):2040003.
18. Das S, Anand A, Agarwal M, Ram M. Release time problem incorporating the effect of imperfect debugging and fault generation: an analysis for multi-upgraded software system. *Int J Reliab Qual Saf Eng*. 2020; 27(02):2040004.
19. Anand A, Bhatt N, Alhazmi OH. Modeling software vulnerability discovery process inculcating the impact of reporters. *Inform Syst Front*. 2020: 1- 4.
20. Bhatt N, Anand A, Aggrawal D. Improving system reliability by optimal allocation of resources for discovering software vulnerabilities. *Int J Qual Reliab Manag*. 2019; (ahead-of-print). http://doi.org/10.1108/ijqrm-07-2019-0246.
21. Kaur J, Anand A, Singh O. Modeling software vulnerability correction/fixation process incorporating time lag. Recent Advancements in Software Reliability Assurance. Boca Raton, FL: CRC Press; 2019: 39- 58.

22. Anand A, Agrawal M, Bhatt N, Ram M. Software patch scheduling policy incorporating functional safety standards. Advances in System Reliability Engineering. Cambridge, MA: Academic Press; 2019: 267- 279.
23. Bhatt N, Anand A, Aggrawal D, Alhazmi OH. Categorization of vulnerabilities in a software. System Reliability Management: Solutions and Technologies. Boca Raton, FL: CRC Press; 2018: 121- 135.
24. A Anand, M Ram, eds. System Reliability Management: Solutions and Technologies. Boca Raton, FL: CRC Press; 2018 Sep 21.

Biographies

Navneet Bhatt received his BSc in computer science and MSc in applied operational research in 2011 and 2013 from University of Delhi, Delhi, India. He has submitted his PhD in the Department of Operational Research, University of Delhi, Delhi, India. He is a lifetime member of the Society for Reliability Engineering, Quality and Operations Management (SREQOM). His current research is focused on software vulnerability discovery modeling, software reliability, machine learning, and multi-criteria decision modeling.

Adarsh Anand did his doctorate in the area of software reliability assessment and innovation diffusion modeling in marketing. Presently, he is working as an assistant professor in the Department of Operational Research, University of Delhi, Delhi, India. He has been conferred with Young Promising Researcher in the field of Technology Management and Software Reliability by Society for Reliability Engineering, Quality and Operations Management (SREQOM) in 2012. He is a lifetime member of the Society for Reliability Engineering, Quality and Operations Management (SREQOM). He is also on the editorial board of International Journal of System Assurance and Engineering management (Springer). He has Guest edited several Special Issues for Journals of international repute. He has publications in journals of national and international repute. His research interest includes software reliability growth modelling, modelling innovation adoption and successive generations in marketing, and social network analysis. He has worked with CRC Press for two editorial projects; “System Reliability Management: Solutions and Technologies” and “Recent Advancements in Software Reliability Assurance.” He has also authored one text book with CRC group; “Market Assessment with OR Applications.”

V. S. S. Yadavalli is a professor and head of Department of Industrial; Systems Engineering, at University of Pretoria, South Africa. He obtained his PhD from Indian Institute of Technology, Chennai, India, in 1983. His main areas of interest include stochastic modeling toward reliability, queuing, inventory, manpower planning systems, and finance. He has published over 150 research papers in these areas. He published articles in various national and international journals including *IEEE Transactions on Reliability, Computers and Operations Research, Computers*; *Industrial Engineering, Stochastic Analysis and Applications*, *International Journal of Systems Science*, *International Journal of Production Economics*, *International Journal of Production Research*, *Applied Mathematics*; *Computation*, and so on. He was past vice-president and president of Operations Research Society of South Africa. He is in the editorial board of various national and international journals. He is a fellow of the South African Statistical Association. He received various awards from various professional bodies like IEOM (Industrial Engineering; Operations Management, USA), South African Institute of Industrial Engineers, South African Institute of Mining and Metallurgy, and so on for his contributions toward research.