To Fear or Not to Fear That is the Question: Code Characteristics of a Vulnerable Function with an Existing Exploit

[1]Awad Younis, [1]Yashwant K. Malaiya, [1]Charles Anderson, and [1]Indrajit Ray
[1]Computer Science Department, Colorado State University, Fort Collins, CO 80523, USA
{younis,malaiya,anderson,indrajit}@cs.colostate.edu

ABSTRACT
Not all vulnerabilities are equal. Some recent studies have shown that only a small fraction of vulnerabilities that have been reported has actually been exploited. Since finding and addressing potential vulnerabilities in a program can take considerable time and effort, recently effort has been made to identify code that is more likely to be vulnerable. This paper tries to identify the attributes of the code containing a vulnerability that makes the code more likely to be exploited. We examine 183 vulnerabilities from the National Vulnerability Database for Linux Kernel and Apache HTTP server. These include eighty-two vulnerabilities that have been found to have an exploit according to the Exploit Database. We characterize the vulnerable functions that have no exploit and the ones that have an exploit using eight metrics. The results show that the difference between a vulnerability that has no exploit and the one that has an exploit can potentially be characterized using the chosen software metrics. However, predicting exploitation of vulnerabilities is more complex than predicting just the presence of vulnerabilities and further research is needed using metrics that consider security domain knowledge for enhancing the predictability of vulnerability exploits.

Keywords
Vulnerabilities Severity; Exploitability; Software security; Exploits; Data mining and machine learning; Feature selection; Prediction; Software metrics.

1. INTRODUCTION
Identifying and addressing software vulnerabilities is important before software release because a single software vulnerability can lead to a breach with a high impact to an organization. However, identifying and addressing potential vulnerabilities can take considerable expertise and effort. Recently, researchers [1], [2], [3], [4] have started investigating ways to predict code areas which are more likely to be vulnerable so security testers can focus on them.

Software vulnerabilities, pose different levels of potential risk. A vulnerability with an exploit written for it presents more risk than the one without an exploit because the existence of an exploit allows an attacker to take advantage of a vulnerability and potentially compromise the affected systems. Allodi and Massacci in [5] have shown that out of the 49599 vulnerabilities reported by the National Vulnerability Database, only 2.10% are in fact exploited. Younis and Malaiya in [6] have also found that only 6.8% out of 486 vulnerabilities of Microsoft Internet Explorer have reported exploits. K. Nayak et al. [7] have reported that combining all of the products they have studied only 15% of disclosed vulnerabilities are ever exploited. Thus, identifying *what characterizes a vulnerability having an exploit* is needed; it can identify code that are more likely than others to have exploits and help security testers focus on areas of highest risk, thus saving limited resources and time. It should be noted that having a reported exploit does not necessarily mean some company or individuals have suffered a real attack. It means that a proof for exploiting a vulnerability exists. Obtaining data on real attacks is challenging because such data is generally kept confidential. Therefore, we will use the presence of an exploit as the ground truth for characterizing exploited vulnerabilities.

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

The objective of this research is to investigate what could characterize a code containing a vulnerability with an exploit. To address this objective, we have studied 183 vulnerabilities from the National Vulnerability Database [8] for the Linux Kernel and Apache HTTP server. The two software systems have been selected because of their rich history of publicly available vulnerabilities, availability of reported exploits, the existence of an integrated repository, availability of the source code, and their diversity in size, functionalities, and domain. For every selected vulnerability, we verify whether it has an exploit reported in the Exploit Database or not [9]. Eighty-two vulnerabilities have been found to have an exploit. Ten of them are for Apache HTTP server and 72 for Linux Kernel. We then mapped these vulnerabilities to their locations at the function granularity level.

After that, we characterize the vulnerable functions with and without an exploit using the selected eight software metrics: Source Line of Code, Cyclomatic complexity, CountPath, Nesting Degree, Information Flow, Calling functions, Called by functions, and Number of Invocations. The reasons why these metrics have been selected are discussed in section 3.1. Based on the metrics values of the vulnerable functions with and without an exploit, we first test the individual selected metrics discriminative power using Welch t-test [10]. Next, we select a combination of these metric using three feature selection methods: correlation-based, wrapper, and principal component analysis. Then, we test the predictive power of the selected subset of metrics using four classifiers: Logistic Regression, Naïve Base, Random Forest, and Support Vector machine.

CODASPY'16, March 09-11, 2016, New Orleans, LA, USA
 ISBN 978-1-4503-3935-3/16/03…$15.00
DOI: http://dx.doi.org/10.1145/2857705.2857750

Table 1. Software Metrics	
Metrics	**Description**
Source Line of Code	**SLOC** measures the size of a code [13]. A higher value of SLOC indicates that an entity is to be difficult to test.
Cyclomatic complexity	**CYC** measures the number of independent paths through a program unit [14]. The higher this metric the more likely an entity is to be difficult
CountPath	**CountPath** measures the number of unique decision paths. A higher value of the CountPath metric represents a more complex code structure [13].
Nesting Degree	**ND** measures the maximum nesting level of control structures in a function. The higher this metric the more likely an entity is to be difficult to test [15].
Information Flow	**Fan-In** measures information flow, which represents the number of inputs a function uses. [16]. The more inputs from external sources the harder to trace where they came from.
Calling Functions	**In-Degree** measures the number of functions that call the function corresponding to the node [17]. The more dependent upon a peace of code, the higher the chance it has a defect.
Called by Functions	**Out-Degree** measures the number of functions that the function corresponding to the node calls [17]. The more depends upon other code, the higher the chance to have a defect.
Number of Invocations	It measures the number of functions that needed to be called before invoking the vulnerable function [18]. The higher this metric the more difficult to reach the vulnerable code.

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

This paper is organized as follows. In section 2, the background of the vulnerabilities, vulnerability databases, exploit database, software metrics, confusion matrix, and feature subset selection methods are discussed. In the next section, the hypotheses are examined and the methodology of testing them is presented. In sections 4, the case studies along with the results are introduced. Section 5 presents the discussion whereas section 6 presents the related work. Finally, concluding comments is given along with the issues that need further research.

2. BACKGROUND

2.1 Vulnerability and Exploit Databases

A software vulnerability is defined as "an instance of [a mistake] in the specification, development, or configuration of software such that the execution can violate the [explicit or implicit] security policy" [11]. Vulnerability' databases are maintained by several organizations such as National Vulnerability Database (NVD) [8] as well as the vendors of the software. An exploit is a method: piece of software, a chunk of data, or a sequence of commands, that identifies and takes advantage of vulnerability [20]. EDB is an exploit database that records exploits and vulnerable software [9].

2.2 Software Metrics

A software metric is a measure of some property of a piece of software. Table 1 summarizes the eight selected metrics. We have selected these metrics based on prior research on fault and vulnerability prediction.

2.4 Confusion Matrix

The confusion matrix table shows the actual vs. the predicted results. For the two class problem a vulnerability is either *has an exploit* or *has no exploit*. The following terms are defined based on Table 2.

Table 2. Confusion matrix			
		Prediction	
		Has an Exploit	**Has no Exploit**
Actual	**Has an Exploit**	TP= True Positive	FN=False Negative
	Has no Exploit	FP= False Positive	TN= True Negative

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

2.5 Feature Subset Selection

Two commonly known types of feature subset selection methods are the filter and the wrapper approach. In the filter approach, the feature selection is performed independently of the learning algorithm and it selects a subset based only on the data characteristics. The wrapper approach, however, conducts a search for a good subset using the learning algorithm as part of the evaluation function [19].

2.5.1 Correlation-based feature selection

Correlation-based feature selection (CFS) evaluates subsets of attributes instead of individual attributes [20]. This technique uses a heuristic to evaluate subset of attributes. The heuristic balances how predictive a group of features are and how much redundancy is among them. In this study, CFS is used with the Greedy stepwise forward search through the space of attribute subsets.

2.5.2 Wrapper Subset Evaluation

The wrapper feature subset evaluation conducts a search for a good subset using a learning algorithm (classifier) as part of the evaluation function. In this study, repeated five-fold cross-validation is used as an estimate for the accuracy of the classifier while a greedy stepwise forward search is used to produce a list of attributes, which are ranked according to their overall contribution to the accuracy of the attribute set with respect to the target learning algorithm [21].

2.5.3 Principal Component Analysis

Principal Component Analysis (PCA) is a statistical technique that transforms a set of possibly correlated variables into a set of linearly uncorrelated variables [22]. These linearly uncorrelated variables are called principal components. The transformation is accomplished by first computing the covariance matrix of the original variables and after that finding its Eigen vectors, principal components. The principal components have the property that most of their information content is stored in the first few features so that the remainder can be discarded. It should be noted that in this paper, PCA is used with the ranker search method that ranks attributes by their individual evaluations.

3. HYPOTHESES AND METHODOLOGY

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

3.1 Hypotheses

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

In the software security field, experts argued that complexity is the enemy of security [1]. It is believed that complexity can be the cause of subtle vulnerabilities that are hard to test and analyze [23] and hence providing a chance to attackers to exploit them. However, we consider the fact that predicting a presence of a vulnerability is different from predicting its exploitation because the latter involves the attacker behavior. As the measures of the complexity vary, we consider the possibility that the potential exploit writers would prefer to exploit less complex code [24]. Besides, in [25], the researchers have observed that the complexity measures for Windows 7 and 8 were negative and they argued that the reason could be that attackers favor simpler vulnerable targets. Based on these reasons, we set up the following research hypotheses related to code complexity (size and structure metrics):

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

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

Sparks et al. [26] studied the penetration depth of reaching a node in a control flow graph. They found that the nodes at greater depths (>10 edges) become increasingly difficult to reach. If crafting an input that can reach a vulnerable statement for a single method is difficult, we believe that crafting an input to call a method containing a vulnerable statement from other methods could be even harder. If we further assume the target system is a closed system, it gets even harder for the attackers to figure out the sequences of calls and inputs that are needed to trigger them. Younis et al. [18] observed that the degree of a call depth of vulnerable functions varies among vulnerabilities. Some of the vulnerabilities have only one degree of depth, while others have 13. Thus, we set up the following research hypothesis for the ease of access (Number of Invocations metric):

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

On the other hand, Younis et al. [18] argued that the more functions are called by the vulnerable function, the higher the effect if the vulnerable function is exploited. They have shown that some vulnerable functions call more than 10 functions while other functions call only one or two functions. Based on the attack surface concept [27], however, the more a function is exposed to the outside environment the larger attack surface. Thus, we argue that the higher the communication a function has, the larger its attack surface gets. From this reasoning, we set up the following research hypothesis for the communication (Fan-In, In-Degree, Out-degree metrics):

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

As argued by Manadhata and Wing [28], no single metric can be an indicator of software quality for all types of software. Besides, according to [19], a feature (or two features) that is completely useless by itself (themselves) can be useful when taken with others (together). Therefore, determining the combination of multiple features (metrics) is important. Thus, we set up the following research hypothesis:

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

3.2 Evaluation Strategy for the Hypotheses

We first investigate the discriminative power of the proposed individual metrics and then test their predictive capability when they are combined. A metric has a discriminative power if it can "discriminate between high-quality software components and low-quality software components" [29]. In this paper, a vulnerable function is classified as exploited if there exists a reported exploit for it and as not exploited if there exists no reported exploit for it at the time of our study. To evaluate the discriminative power of the proposed hypotheses, H1, H2, H3, and H4, we test their null hypotheses. Because our data has unequal sample size, unequal variances and is skewed, we used the Welch t-Test [10]. The Welch t-test is an adaptation of t-test to compare the means of two samples when the two samples have unequal variances and unequal sample sizes and it also is known to provide a good performance for skewed distributions. The difference between the means of the two group is considered discriminative when the result from the Welch t-test are statistically significant at the $p < 0.05$ level.

However, to evaluate the predictive power of the hypothesis H5 (combined metrics), on the other hand, two challenges have to be addressed. First, how can we select the subset metrics? Second, how can we evaluate the predictive power of the selected subset? To address the first challenge, we use feature subset selection methods. There are two commonly used feature subset selection methods: the filter and the wrapper approach. In this paper, we use correlation-based feature selection (CFS) and wrapper subset evaluation (WRP) methods. The first method has been selected because it selects the subset features based on only the characteristics of the data while the second method selects the features based on the learning algorithm and this can help us observe an advantage of one method over the other. However, according to [17], combining several metrics can be affected by the multicollinearity and that is due to the inter-correlation among metrics. To account for the multicollinearity problem, we use the Principal Component Analysis (PCA) and compare its result with two selected methods.

To address the second challenge, we evaluate the performance of a binary classification technique to assess the predictive power of the selected subset metrics. A binary classifier can make two possible types of errors: false positives and false negatives. These two types of errors are defined in section 2.4. To measure the performance of a classifier, we mainly use recall, precision and false positive ratio. In addition, we also report the harmonic mean of the precision and recall, using the F-measure, and the accuracy. It should be noted that a high recall value is required even at the cost of the precision and that is because of the significant impact of one exploited vulnerability. It is also desirable to have a low false positive rate because that helps avoiding a waste of an inspection effort. P. Morrison et al. in [25] suggested that the value 0.7 of the recall and precision measures are considered reasonable for the prediction models in the realm of software quality.

Recall is defined as the ratio of the number of vulnerabilities correctly predicted as having an exploit to the number of

Table 4. Apache HTTP Server Vulnerabilities' Measures									
Vulnerability	**In-Degree**	**Out-Degree**	**CountPath**	**ND**	**CYC**	**Fan-In**	**No of Invocation**	**SLOC**	**Exploit Existence**
CVE-2009-1891	1	9	9000	6	68	45	2	211	**NEE**
CVE-2010-0010	4	9	145	4	11	16	4	38	**EE**
CVE-2013-1896	26	5	8	1	5	37	3	29	**EE**

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

We have considered four classifiers for this study and they are namely: Logistic Regression (LR), Naïve Bayes (NB), Random Forests (RF), and Support Vector Machine (SVM). LR has been selected because it is a standard statistical classification technique whereas NB has been selected because it is a simple classifier and it has often outperformed more sophisticated classifiers [4]. Besides, RF has been selected because it is more robust to noise such as inter-correlated features while SVM has been selected because it is less prone to overfitting.

4. EXPERIMENTATION

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

4.1 Data Collection

In this study, the data about vulnerabilities and exploits of Linux kernel and Apache HTTP Server were collected from NVD [8] and the EDB [9] respectively from the period 2002 to 2014. Table 3 shows the number of the selected vulnerabilities and their exploits. It should be noted that we have considered all reported vulnerabilities that have an exploit for the two selected software system. We only considered some of the vulnerabilities that do not have an exploit. These vulnerabilities have been mainly selected based on their age and information about their locations. We tried to select the vulnerabilities that are at least 3 or 4 years old, so that their lack of exploit is not due to their recent discovery.

Table 3. Vulnerabilities and Exploits			
Software	**EE**	**NEE**	**Total Each**
Linux Kernel	72	81	**153**
Apache	10	20	**30**
Total All	**82**	**101**	**183**

4.2 Computing the Metrics

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

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

Once the vulnerable version and the vulnerable function have been identified, we can now compute the metrics at the function level from the source code [30] and [31].There are different tools that can be used to compute these metrics. We have chosen the commercial tool Understand [32]. This tool has been chosen because it is user-friendly and it has a good set of APIs that allows interaction with programming languages such as Python, Perl, and C++. For the selected vulnerable version we have performed the following using our own python script:

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

Showing the whole measures for the selected software is limited by the number of pages, thus Table 4 shows the measures of the selected metrics for some of Apache HTTP Server vulnerabilities. As can be seen, the measures of the vulnerabilities are distinguished by the availability of exploit as either an exploit exist (EE) or no exploit exist (NEE).

4.3 Discriminative Power Test

4.3.1 Apache Dataset

Table 5 shows the results of testing the hypotheses H1, H2, H3, and H4 for the discriminative power of the individual metrics for the Apache HTTP Server dataset. It should be noted that we have performed an outliers test for all computed metrics to avoid their effect on the mean. Only one metric, calling functions, has been found to have outliers.

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

H2: Looking at the structure metrics values, however, only the CountPath metric has shown statistical significant difference, the p-value is 0.011, for vulnerabilities with exploits compared to those without an exploit. Thus, the Welch t-test result implies that the vulnerabilities with an exploit tend to have smaller CountPath

Table 5. Result of Discriminative Power Test for Apache HTTP Server Dataset								
			EE (Observations = 10)		NEE (Observations = 20)			
Class of Metrics	**Metrics**		**Mean**	**Variance**	**Mean**	**Variance**	**t-value**	**P-Value**
Size	SLOC		54.2	134.4	698.4	18148.2	-2.49	**0.021**
Structure	Cyclomatic complexity		13.6	27.3	83.8	766.6	-1.95	0.063
	Nesting Degree		3.7	2.5	3.4	3.7	0.43	0.669
	CountPath		45.4	2462.3	3072	13840876.5	-2.83	**0.011**
Ease of Access	Number of Invocations		2.9	0.4	2.5	0.6	1.55	0.136
Communication	Information Flow		14.69	93.5	17.5	150.6	-0.73	0.472
	Calling functions	**Outliers**	5.7	70.3	1.4	0.7	1.53	0.163
		No outliers	5.2	64.6	1.2	0.3	1.6	0.151
	Called by functions		4.9	9.1	8.4	37.5	-2.01	**0.054**

than the vulnerabilities without an exploit. On the other hand, while the Cyclomatic complexity values are smaller for vulnerabilities with an exploit, though the difference is not significant, Nesting Degree values are higher for vulnerabilities with an exploit and this is a contrary to what we anticipated. However, according to [33], the recommended maximum for Nesting Degree is 5 and those two values are less than this number. This suggests that Nesting Degree is smaller for both vulnerabilities with an exploit and those without an exploit.

H3: The ease of access metric has shown slightly higher values for vulnerabilities with exploits which is not what we anticipated. The median for the vulnerabilities with and exploits and without an exploit is 3.0 and 2.5 respectively. Therefore, the Welch t-test result suggests that the vulnerabilities with an exploit tend to have a slightly more number of invocations than those without an exploit but this difference is not significant. However, Sparks et al. in [26] found that the nodes at greater depths (>10 edges) become increasingly difficult to reach. Therefore, we conclude that the degree of depth (around 3.0) for both groups is smaller.

H4: As can be seen from the communication metrics values, only calling functions metric has shown to have higher values for vulnerabilities with exploits than for the vulnerabilities without an exploit. However, this difference is not statistically significant. On the other hand, information flow and called by functions metrics have shown a smaller values for vulnerabilities with an exploit and that is not what we anticipated.

4.3.2 Linux Kernel Dataset

We also obtained the results of testing the hypotheses H1, H2, H3, and H4 for the discriminative power of the individual metrics for the Linux Kernel dataset. We have performed an outlier test for all computed metrics so that to avoid its effect on the mean. All metrics' values have been found to have outliers except for one metric, number of invocations. We have run the Welch t-test on both data with and without outliers.

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

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

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

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

4.4 Predictive Power Test

To test predictive power of the metrics, we need to select a subset of the proposed metrics. To do that, we implemented CFS, WRP, and PCA feature selections techniques using Waikato Environment for Knowledge Analysis (WEKA) [34]. WEKA is a popular open source toolkit implemented in Java for machine learning and data mining tasks. Once the metrics subsets have been selected, we implemented the four selected classifiers: LR, NB, RF, and SVM using WEKA. It should be noted that the parameters for the chosen feature selection techniques and classifiers are initialized with the default settings of the WEKA toolkit. It should be also noted that the results are obtained by performing 10-fold cross-validation so that the variability in prediction are reduced. Cross-validation is a technique for assessing how accurately a predictive model will perform in practice [35]. However, it is more important to identify exploited vulnerabilities even at the expense of incorrectly predicting some not exploited vulnerabilities as exploited vulnerabilities. This is because a single exploited vulnerability may lead to serious security failures. Thus, we use recall as our main performance measure to compare among the classifiers' performance. The results of testing metrics predictive power are shown in the following two subsections.

4.4.1 Apache Dataset

Table 6 shows predictive power results for the Apache dataset. Column one and two contain the classifiers and their performance measures respectively. We first start with testing every classifier using the whole selected metrics and collect the performance measures provided by WEKA, as shown in column three. For

Table 6. Result of Predictive Power Test for Apache HTTP Server Dataset						
Model	**Performance Measures**	**All metrics (%)**	**Correlation-Based (CFS) (%)**	**Wrapper Subset (WRP) (%)**		**PCA (%)**
				BN	**RF**	
Logistic Regression	Recall	60	**73**	73	67	**74**
	Precision	60	**73**	72	64	**75**
	F-Measure	53	**73**	73	64	**74**
	Accuracy	60	**73.3**	73.3	66.6	**73.3**
	FPR	50	**33**	38	52	**28**
Naïve Bayes	Recall	70	70	**77**	80	63
	Precision	84	84	**79**	81	72
	F-Measure	70	70	**77**	80	64
	Accuracy	70	70	**76.7**	80	63.3
	FPR	15	15	**22**	20	28
Random Forest	Recall	**77**	70	73	**83**	73
	Precision	**76**	69	72	**84**	73
	F-Measure	**76**	70	73	**84**	73
	Accuracy	**76.6**	70	73.3	**83.3**	73.3
	FPR	**32**	40	38	**18**	33
Support Vector Machine	Recall	67	67	73	73	67
	Precision	44	44	81	81	44
	F-Measure	53	53	67	67	53
	Accuracy	66.7	66.7	73.3	73.3	66.7
	FPR	67	67	53	53	67

convenient interpretation, we express the performance measures in terms of percentage, where a 100 % is the best value and 0 %is the worst value. Then, we select a subset of those metrics using CFS, WRP, and PCA feature selection techniques and test the chosen classifier using the selected subsets and provide the performance measures as shown in column four, five, and six respectively.

Considering the value 0.7 (70%) of recall and precision measures as reasonable [25], we show for every metrics subset the highest recall for a classifier in bold. Now, we will compare the classifiers performance measures using different subset metrics. We will first start with using the whole metrics. As can be seen, only NB and RF report the best recall and precision value. Moreover their accuracy and precision performance measures are either 70 or more. Even though NB reports the best precision, the reported harmonic mean, F-measure, by RF is better. It should be also notated that FPR reported by NB is lower than the one reported by RF. However, when the subset metrics selected by the correlation-based feature selection, we see that not only LR has improved but it has done better than the other classifiers. Neither the BN nor the SVM has shown any improvement.

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

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

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

4.4.2 Linux Kernel Dataset

The metrics predictive power results for the Linux Kernel dataset show that none of the classifiers has a 70 % recall value. However, let us investigate if any of the classifiers has a recall score of at least 50%. When the whole metrics have been used, none of the classifiers has a recall score of 50%. However, when the CFS feature selection technique has been used, RF has a 50% recall score. It should be noted that while the NB and LR have improved when the CFS was used, compared when using the whole metrics, SVM performed worse than when it used the whole metrics. On the other hand, when the WRP has been used, only NB and RF has their recall score improved. When the PCA technique has been used, however, only NB and SVM have shown a recall score above 50%. Though using the feature selection techniques has slightly improve the performance of the chosen classifiers, their FPRs have a score close to or greater than 50%. This shows that the classifiers have difficulty to learn from this dataset and hence behaved almost randomly. It should be noted that the SVM, when the PCA was used, has performed the best compared to the other classifiers.

H5: We can conclude that there is a combination of metrics that significantly predicts vulnerabilities that have an exploit using Linux Kernel dataset but at low recall score.

4.5 Threats to Validity

In this paper we have considered the datasets for only two products, the Apache HTTP server and Linux Kernel. However, they are both very significant examples. The Apache HTTP server has more than 169 belonging to different categories. Besides, its line of code varies between 50,712 LOC to 358,633 LOC. We recognize that the number of vulnerabilities that have an exploit reported for them in Apache HTTP server is low. However, we are just scratching the surface based on what is available. On the other hand, Linux Kernel has larger number of vulnerabilities, more than 1200. Its size in line of code has ranged from 10,239 LOC to 15,803,499 LOC. It also has a greater variety of vulnerabilities. As there are other potential factors that can

influence the probability of development of an exploit for a vulnerability that have not been examined in this study. Finally, exploits that have not yet been publically reported were not considered in our study.

5. DISCUSSION

One of our main observations is that some metrics have a good discriminative and predictive power for Apache dataset. However, they do not have significant discriminative and predictive power for the Linux Kernel dataset. Moreover, we also observed that, unlike in Apache dataset, some metrics such as CountPath, values have been found to be higher for vulnerabilities with exploit in Linux dataset. One possible reason could be the value of the target. *The exploits developers may be willing to exploit vulnerabilities in an operating system even if it requires more effort because having a root access could be worth the effort.* Besides, when compared to Apache, Linux Kernel has more exploits. This shows that this might be because Linux Kernel is a more valuable target for the attackers. To verify this, we looked into the initial release dates of the both products and we found that the difference in age is not very significant (Linux is 23 years old and Apache 20 years old), and also looked into the usage statistics and market share data and we found Apache market share is around 57% [36] whereas Linux Kernel is about 52.4 % [37].

Based on the Apache dataset, we have also observed that when a function is vulnerable and has an exploit, its SLOC, CYC, and CountPath values have been found to be lower than the vulnerable functions without an exploit. A similar result has been observed by [25] when they try to predict the existence of a vulnerability. *It seems that the attackers favoring simpler vulnerable targets,* especially when the goal is to deny a service, such as the one provided by Apache, using the least effort. Using the WRP as a method to select a combination of metrics has the best impact on the classifiers performance. However, in [25], security domain knowledge metrics have to be considered in order for the vulnerability prediction performance to be enhanced.

6. RELATED WORK

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

***CVSS Metrics*:** CVSS metrics are the de facto standard for measuring the severity of vulnerabilities [38]. CVSS Base Score measures severity based on exploitability (the ease of exploiting vulnerability) and impact (the effect of exploitation). However, CVSS exploitability measures have some limitations. First, they assign static subjective numbers to the metrics based on expert knowledge. In contrast, we focus on reducing subjectivity in assessing vulnerability exploitation by basing our analysis on software attributes that can be objectively derived from the source code. Second, two of CVSS's factors (Access Vector and Authentication) have the same value for almost all vulnerabilities [5]. Third, there is no formal procedure for evaluating the third factor (Access Complexity) [38]. Consequently, it is unclear if CVSS considers the software structure and properties as a factor.

***Assessing vulnerability exploitability*:** Gegick et al. [39] argued that vulnerabilities that are located in low risk areas of the code should be prioritized differently from the ones that are located in high risk areas of the code. Their results show that the combined usage of the internal metrics has predicted the attack-prone components with a high accuracy and zero false negative rates. While the authors granularity analysis was at the component level ours is at the function level, which might reveal some more important information [40]. Moreover, we used different internal code-level metrics. In addition, they used reported security failures to identify a component as an attack prone and a non-attack-prone. In contrast, we used the availability of exploit to identify a vulnerable function as either exploited or not exploited function.

Bozorgi et al. [41] proposed a Machine Learning and Data mining technique that uses features mined from known vulnerability reports to predict the possibility of vulnerability exploitability. They compare their results with the CVSS Exploitability metrics and found that their approach can classify vulnerability exploitability better than the CVSS. We consider the relationship between some software internal metrics, which are extracted from the source code, and the availability of exploits. This is particularly important for newly released software where vulnerability reports are not available.

Allodi and Massacci in [42] and [43] have proposed the black market as an index of risk of vulnerability exploitation. Their approach assesses the risk of vulnerability exploitation based on the volumes of the attacks due to the vulnerability exploits sold in the black market. In contrast, in this paper we try to investigate the relationship between software metrics and the availability of exploits, as data about vulnerabilities attacked in the wild is not always available. This could help software developers predict vulnerabilities exploitation on the development side instead of the deployment side.

***Using software metrics*:** Several researchers have studied the possibility of using software metrics to predict vulnerable entities (components, classes, modules, files, and functions/methods) or the existence of vulnerabilities. Shin and Williams [1] and [2] investigated the possibility of using complexity metrics as predictors for the location of security problems. Chowdhury and Zulkernine [3] investigated the usability of complexity, coupling and cohesion metrics as predictors of vulnerabilities' location. However, in [4], Zimmermann et al. studied several software metrics including code complexity and dependency as predictors for the existence of vulnerabilities. Our work, on the other hand, focuses on predicting the exploitability of a vulnerability.

***Using Graph-based Metrics*:** Bhattacharya et al. [44] studied the possibility of applying graph-based approaches to software engineering tasks. Using the source code, they construct a graph model and use graph metrics to measure some properties. Their results show that graph metrics can detect significant structural changes, and can help estimate bug severity, prioritize debugging efforts, and predict defect-prone releases. In this paper, we use different graph metrics and investigate whether they are correlated with a vulnerability severity rather than a bug severity. Besides, we use the availability of an exploit to identify the severity of a vulnerability.

7. CONCLUSIONS AND FUTURE WORK

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

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

Even though the two selected applications have a rich history of reported vulnerabilities, considering software with a different domain, such as an open source browser like Firefox, increases the size of the dataset and that might reveal significant information. Improving the classifiers performance and capturing vulnerabilities exploitability may require further empirical investigations of software metrics specifically applicable to the security realm. Thus, further research is needed which considers the metrics related to attack surface [28], reachability and dangerous system calls metrics [18, 45], graph-based metrics [44], and static analysis tool warnings metrics [39]. Moreover, using an alternative approach such as a text mining technique [46] to predict vulnerability exploitability might lead to an interesting results.

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

8. REFERENCES

[1] Shin, Y. and Williams, L. "Is complexity really the enemy of software security?" *in Proc. ACM Workshop Quality Protection*, 2008, pp. 47–50.

[2] Shin, Y. and Williams, L. "An empirical model to predict security vulnerabilities using code complexity metrics," *in Proc. ACM-IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2008, pp. 315–317.

[3] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," *J. Syst. Archit.*, vol. 57, no. 3, pp. 294–313, 2011.

[4] T. Zimmermann, N. Nagappan, and L. Williams, "Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista," *in Proc. Int. Conf. Softw. Testing, Verification Validation*, 2010, pp. 421–428.

[5] L. Allodi and F. Massacci, "My Software has a Vulnerability, should I worry?," *arXiv preprint arXiv*:1301.1275, 2013.

[6] A. Younis and Y.K. Malaiya. "Comparing and Evaluating CVSS Base Metrics and Microsoft Rating System". *The 2015 IEEE International Conference on Software Quality, Reliability and Security,* 2015, pp. 252-261.

[7] K. Nayak, D. Marino, P. Efstathopoulos, T. Dumitra, "Some vulnerabilities are different than others". *In: Proceedings of the 17th International Symposium on Research in Attacks, Intrusions and Defenses,* 2014, pp. 426–446.

[8] "National Vulnerability Database Home.". Available: http://nvd.nist.gov/. [Accessed: 24-May-2015].

[9] EDB: Exploits Database by Offensive Security. Available: http://www.exploit-db.com/. [Accessed: 24-May-2015].

[10] M. Fagerland and L. Sandvik. "Performance of five two-sample location tests for skewed distributions with unequal variances." *Contemporary clinical trials*, vol. 30, pp.490-496, 2009.

[11] A. Ozment, "Improving vulnerability discovery models," *in Proceedings of the 2007 ACM workshop on Quality of protection*, New York, NY, USA, 2007, pp. 6–11.

[12] S. Frei, D. Schatzmann, B. Plattner, and B. Trammell, "Modeling the Security Ecosystem - The Dynamics of (In)Security," in Economics of Information Security and Privacy. Springer US, 2010, pp. 79–106.

[13] N.E. Fenton, S.L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, PWS Publishing Co., Boston, MA, USA, 1997.

[14] T.J. McCabe, A complexity measure, *IEEE Transactions on Software Engineering* 2 (4) (1976) 308–320.

[15] W.A. Harrison, K.I. Magel, A complexity measure based on nesting level, *ACM Sigplan Notices* 16 (3) (1981) 63–74.

[16] S. Henry, D. Kafura, Software structure metrics based on information flow, *IEEE Transactions on Software Engineering* (1981) 510–518.

[17] N. Nagappan, T. Ball, A. Zeller, Mining metrics to predict component failures, *in Proceedings of the 28th International Conference on Software Engineering*, Shanghai, China, May 2006, pp. 452–461.

[18] A. Younis, Y.K. Malaiya and I. Ray, "Assessing Vulnerability Exploitability Risk Using Software Proprieties", *Software Quality Journal*: 1-44, Mar 2015.

[19] G. Forman, "An extensive empirical study of feature selection metrics for text classification." The Journal of machine learning research, 3, p.1289-1305, 2003.

[20] M. Hall and L. Smith. Practical feature subset selection for machine learning. *In Proceedings 21st Australasian Computer Science Conference*, University of Western Australia, Perth, Australia, February 1996.

[21] R. Kohavi, G.H. John, "Wrappers for feature subset selection" *Artificial Intelligence*, 97(1-2), p. 273-324, 1997.

[22] I. Jolliffe, Principal component analysis. John Wiley & Sons, Ltd, 2002.

[23] B. Schneier, Beyond Fear: Thinking Sensibly about Security in an Uncertain World. Springer-Verlag, 2003.

[24] E. Alata1, V. Nicomette1, M. Kaâniche1, M. Dacier, and M. Herrb, "Lessons Learned from the Deployment of a High-Interaction Honeypot", EDCC'06: *in Proc. 6th European Dependable Computing Conf. Coimbra*, Portugal, 2006, pp. 39–46.

[25] P. Morrison, K. Herzig, B. Murphy, and L. Williams, "Challenges with Applying Vulnerability Prediction Models", Proceedings of the 2015 Symposium and Bootcamp on the Science of Security, 2015. Microsoft Research: http://research.microsoft.com/apps/pubs/default.aspx?id=240601. [Accessed: 24-March-2015].

[26] S. Sparks, S. Embleton, R. Cunningham, and C. Zou, "Automated vulnerability analysis: Leveraging control flow for evolutionary input crafting*," in Computer Security Applications Conference*, 2007. ACSAC 2007. Twenty-Third Annual, 2007, pp. 477–486.

[27] M. Howard, J. Pincus, and J. Wing, "Measuring Relative Attack Surfaces," *in Computer Security in the 21st Century*, D. T. Lee, S. P. Shieh, and J. D. Tygar, Eds. Springer US, 2005, pp. 109–137.

[28] P. K. Manadhata and J. M. Wing, "An Attack Surface Metric," *Software Engineering, IEEE Transactions* on, vol. 37, no. 3, pp. 371 –386, Jun. 2011.

[29] IEEE, "IEEE Standard for a Software Quality Metrics Methodology," *IEEE Std* 1061-1998 (R2004), IEEE CS, 2005.

[30] Apache-SVN. The apache software foundation. Available: http://www.svn.apache.org/viewvc/. [Accessed: 24-May-2015].

[31] Linux Kernel Archive. Available: https://www.kernel.org/ [Accessed: 24-May-2015].

[32] Scientific Toolworks Understand. Available: http://www.scitools.com/. [Accessed: 24-May-2015].

[33] LocMetrics. Available: http://www.locmetrics.com/index.html. [Accessed: 24-May-2015].

[34] WEKA Toolkit. Available: http://www.cs.waikato.ac.nz/ml/weka. [Accessed: 24-May-2015].

[35] I.H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques (2nd ed.), Morgan Kaufmann, San Francisco, 2005.

[36] Usage Statistics and Market Share of Web Servers for Websites. Available: http://www.w3techs.com/technologies/overview/web_server/all. [Accessed: 24-May-2015].

[37] Usage Statistics and Market Share of Web Servers for Websites. Available: http://w3techs.com/technologies/details/os-unix/all/all. [Accessed: 24-May-2015].

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

[39] M. Gegick, L. Williams, J. Osborne, and M. Vouk. "Prioritizing software security fortification through code-level metrics." *In Proceedings of the 4th ACM workshop on Quality of protection*, 2008, pp. 31-38.

[40] T. Zimmermann, R. Premraj, A. Zeller, "Predicting defects for eclipse". *In Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, 2007, pp. 9–15.

[41] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond heuristics: learning to classify vulnerabilities and predict exploits," *in Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, NY, USA, 2010, pp. 105–114.

[42] L. Allodi and F. Massacci, "A preliminary analysis of vulnerability scores for attacks in wild," *ACM Proc. of CCS BADGERS*, 2012, pp.17-24.

[43] L. Allodi and F. Massacci, "My Software has a Vulnerability, should I worry?,", 2013 , arXiv preprint arXiv:1301.1275.

[44] P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos, "Graph-based analysis and prediction for software evolution*," in Proc. Intl. Conf. on Softw. Eng. (ICSE). ACM*, 2012, pp. 419–429.

[45] A. Younis and Y.K. Malaiya," Using Software Structure to Predict Vulnerability Exploitation Potential," The 8th IEEE International Conference on Software Security and Reliability, 2014, pp. 13-18.

[46] R. Scandariato, J. Walden, A. Hovsepyan, W. Joosen. Predicting vulnerable software components via text mining. *IEEE Trans Softw Eng*, 40 (10) (2014), pp. 993–1006.