Can We Use Software Bug Reports to Identify Vulnerability Discovery Strategies?

Farzana Ahamed Bhuiyan
Tennessee Technological University
Cookeville, Tennessee, USA
fbhuiyan42@students.tntech.edu

Raunak Shakya
Tennessee Technological University
Cookeville, Tennessee, USA
rshakya@students.tntech.edu

Akond Rahman
Tennessee Technological University
Cookeville, Tennessee, USA
arahman@tntech.edu

ABSTRACT

Daily horror stories related to software vulnerabilities necessitates the understanding of how vulnerabilities are discovered. Identification of data sources that can be leveraged to understand how vulnerabilities are discovered could aid cybersecurity researchers to characterize exploitation of vulnerabilities. *The goal of the paper is to help cybersecurity researchers in characterizing vulnerabilities by conducting an empirical study of software bug reports.* We apply qualitative analysis on 729, 908, and 5336 open source software (OSS) bug reports respectively, collected from Gentoo, LibreOffice, and Mozilla to investigate if bug reports include vulnerability discovery strategies i.e. sequences of computation and/or cognitive activities that an attacker performs to discover vulnerabilities, where the vulnerability is indexed by a credible source, such as the National Vulnerability Database (NVD). We evaluate two approaches namely, text feature-based approach and regular expression-based approach to automatically identify bug reports that include vulnerability discovery strategies.

We observe the Gentoo, LibreOffice, and Mozilla bug reports to include vulnerability discovery strategies. Using text feature-based prediction models, we observe the highest prediction performance for the Mozilla dataset with a recall of 0.78. Using the regular expression-based approach we observe recall to be 0.83 for the same dataset. Findings from our paper provide the groundwork for cybersecurity researchers to use OSS bug reports as a data source for advancing the science of vulnerabilities.

CCS CONCEPTS

• **Security and privacy → Software security engineering**.

KEYWORDS

bug report, empirical study, ethical hacking, strategy, vulnerability

ACM Reference Format:
Farzana Ahamed Bhuiyan, Raunak Shakya, and Akond Rahman. 2020. Can We Use Software Bug Reports to Identify Vulnerability Discovery Strategies?. In *Hot Topics in the Science of Security Symposium (HotSoS '20), April 7–8, 2020, Lawrence, KS, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3384217.3385618

HotSoS '20, April 7–8, 2020, Lawrence, KS, USA

ACM ISBN 978-1-4503-7561-0/20/04…$15.00
https://doi.org/10.1145/3384217.3385618

1 INTRODUCTION

According to the United States Department of Homeland Security, the information technology (IT) sector, which encompasses software and software-based services is a critical infrastructure and "*central to the nation's security, economy, and public health and safety*" [29]. However, software vulnerabilities can lead to large-scale consequences. For example, in July 2019, the 'DELL PC Doctor' vulnerability impacted millions of Dell computers [13, 14]. Characterizing how vulnerabilities are discovered can be helpful to devise mechanisms that will protect software and software-based services early at the development stage.

Creation of knowledge related to vulnerability discovery has gained interest in the research community. For example, King [30] emphasized on the understanding of attacker actions to characterize the nature of vulnerabilities. Munaiah et al. [28] also expressed similar views: "*The ability to understand the way in which these adversaries discover and exploit vulnerabilities to infiltrate systems can help bring the attacker mindset out of the wild and into the software development process*".

Identifying vulnerability discovery strategies can be useful to characterize vulnerabilities in software systems. A vulnerability discovery strategy is one or more computational and/or cognitive activities that an attacker performs to discover a vulnerability, where the vulnerability is indexed by a credible source, such as the National Vulnerability Database (NVD). However, the availability of data sources to study vulnerability discovery strategies is limited [28]. Researchers in prior work [28] have stated that even though identifying vulnerability discovery strategies is important, conducting such analysis could be challenging due to "*scarcity of such empirical information*".

One option to identify vulnerability discovery strategies could be the use of bug reports. A bug report is a description of bugs that are typically indexed by bug databases. In the software engineering domain, researchers have used open source software (OSS) bug reports successfully to characterize and quantify software bugs [35]. Based on evidence reported by prior research [5, 7, 8], we conjecture that bug reports from OSS can be used to identify vulnerability discovery strategies. Prior research [5, 7, 8] has reported that OSS bug reports include what sequence of actions were performed to discover and reproduce a certain bug in a software. Our hypothesis is that OSS bug reports will include sequence of actions that express how the vulnerability was discovered by the attacker. For example, if an OSS bug report includes steps of actions on how to reproduce a vulnerability, then the bug report can further be investigated to find out how the attacker discovered the vulnerability. OSS bug reports are retrievable through OSS APIs, which researchers can

use to construct datasets that can further be analyzed to identify vulnerability discovery strategies.

The goal of the paper is to help cybersecurity researchers in characterizing vulnerabilities by conducting an empirical study of software bug reports.

We answer the following research questions:

- ***RQ1**: How frequently do vulnerability discovery strategies appear in bug reports?*
- ***RQ2**: How can we automatically identify bug reports that include vulnerability discovery strategies?*

We conduct an empirical analysis with OSS bug reports collected from IT organizations who have open sourced the bug reports and source code of their software. We apply qualitative analysis on 729, 908, and 5336 OSS bug reports respectively, collected from Gentoo, LibreOffice, and Mozilla to investigate if those reports include vulnerability discovery strategies. We also use two automated approaches namely, text feature-based approach and regular expressions to identify bug reports that include vulnerability discovery strategies. For text feature-based approach, we construct n-grams [21] from the bug report content, and construct models with statistical learners, such as classification and regression tree [3], logistic regression [16], and deep neural networks [23]. We evaluate our models using two approaches: 10×10-fold validation and cross dataset evaluation.

We make the following contributions:

- An empirical study that shows evidence of vulnerability discovery strategies that are present in OSS bug reports; and
- A set of prediction models that automatically identifies presence of vulnerability discovery strategies in OSS bug reports.

We organize the rest of the paper as following: in Section 2 we provide necessary background and prior research related to our paper. We present the methodology of our paper in Section 3. We provide empirical findings in Section 4, and discuss results in Section 5. We describe the threats to validity for our paper in Section 6. Finally, we conclude the paper in Section 7.

2 BACKGROUND AND RELATED WORK

In this section we provide necessary background on bug reports and discuss relevant prior work.

2.1 Background on Bug Reports

Bug reports are software artifacts that are used by software development teams to track bugs and tasks. Development teams use both open source and proprietary tools to track their bugs. Examples of open source tools include Bugzilla [1], Fossil [2], and Trac [3]. Examples of proprietary bug tracking tools include Jira [4] and Team Foundation Server [5]. Bug reports are also referred to as issue trackers.

Typical entities of bug reports include but are not limited to:

- title that summarizes the bug;
- a bug ID that is unique across all listed bugs;
- timestamp on when the bug report was created and updated;
- affected product information e.g. operating system, product type, and platform;
- bug report category;
- bug report priority; and
- comments that discuss how to reproduce the bug.

We provide an annotated snapshot of a bug report [6] retrieved from the Mozilla organization in Figure 1 to illustrate entities of a bug report even further.

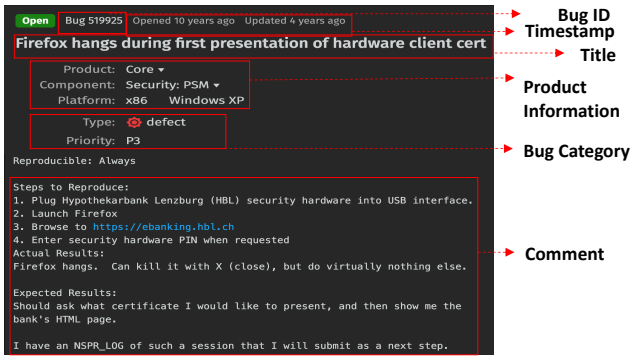

Figure 1: Annotation of a bug report retrieved from Mozilla.

2.2 Related Work

Our paper is closely related to prior work that have investigated bug localization, bug report identification, and bug report quality assessment. For example, Peters et al. [32] proposed 'FARSEC', a technique to reduce mislabelling of security bug reports using text-based prediction models. Peters et al. [32] developed FARSEC for automatic identification of security cross-words and for scoring bug reports according to how likely they are to be labeled as security bug reports.

Bug reports have also been leveraged for locating bugs in software. Dilshener et al. [12] proposed and evaluated an approach that uses a heuristic-based score for a file against a given report, without requiring historical dataset of code and bug reports. Ali et al. [1] proposed and evaluated 'LIBCROOS' that combines the results of information retrieval techniques with binary class relationships gathered through source code analyses.

Researchers have also investigated automated retrieval of information from bug reports. Chaparro et al. [7] investigated the effectiveness of query reduction strategies, based on the structure of bug descriptions available from bug reports. The authors [7] proposed an automated technique where developers issue an initial query from the full text of the bug report and inspect the top-N code candidates returned by text retrieval-based bug localization techniques. To reformulate the query, the authors [7] used 31 strategies using 5 text retrieval-based bug localization techniques. In another work, Chaparro et al. [8] proposed, implemented, and evaluated 'DEMIBUD', an automated technique to detect if bug information is missing in bug reports to reproduce a bug. The authors [8] identified patterns that captured the discourse and developed 3 versions

[1] https://www.bugzilla.org/
[2] https://www.fossil-scm.org/home/doc/trunk/www/index.wiki
[3] https://trac.edgewall.org/
[4] https://www.atlassian.com/software/jira
[5] https://azure.microsoft.com/en-us/services/devops/server/
[6] https://bugzilla.mozilla.org/show_bug.cgi?id=519925

of DEMIBUD, based on heuristics, natural language processing and linear support vector machine to detect the absence of expected software behavior (EB) and steps to reproduce (S2R) in bug reports. In another work, Chaparro et al. [5] proposed and evaluated EULER, an automated technique to identify and evaluate the quality of the steps to reproduce in a bug report and provide feedback to the reporters about ambiguous steps. Chaparro et al. [5] used sequence labeling in combination with discourse patterns and dependency parsing to identify steps to reproduce sentences. EULER was also capable to provide specific feedback to the reporter about missing steps to reproduce a bug. Chaparro et al. [6] conducted an empirical study to investigate to what extent duplicate bug report descriptions use the same vocabulary and the impact of the vocabulary agreement on duplicate detection of bug reports. Zhao et al. [40] proposed and evaluated 'S2RMiner', an automated technique that extracts the text description of steps to reproduce from bug reports using HTML parsing, natural language processing and machine learning techniques. The authors downloaded bug reports as HTML files and used HTML parsing to extract relevant text from the files of the bug reports. From the extracted content the authors used natural language processing to obtain text features of each sentence, and applied support vector machines in bug reports to predict and extract steps to reproduce.

The above-mentioned discussion shows a plethora of research related to bug reports e.g., bug information retrieval, bug localization, and bug report quality assessment. However, we notice lack of research that investigates if bug reports contain information on how vulnerabilities are discovered. We address this research gap in our paper.

3 METHODOLOGY

In this section, first we provide necessary definitions listed below:

- Vulnerability: A vulnerability is a weakness in computational logic existing in software and hardware components, which upon exploitation can result in a negative impact to confidentiality, integrity, or availability [11].
- Vulnerability discovery strategy: One or more computational and/or cognitive activities that an attacker performs to discover vulnerabilities, where the vulnerability is indexed by a credible source, such as the NVD.

Now, we describe our methodology to conduct the research study, which is summarized in Figure 2.

3.1 Bug Database Mining

In our paper we focus on identifying if vulnerability discovery strategies exist in software bug reports. The first step of performing such analysis is to mine bug databases and collect bug reports. We rely on OSS bug reports as they are accessible via APIs. We collect bug reports from three organizations: Gentoo Linux [7], LibreOffice [8], and Mozilla [9]. We select these organizations as they are different from one another with respect to the software product that they deliver to the end-users. Gentoo Linux is an OSS Linux-based operating system. LibreOffice is an OSS for word processing. Mozilla produces a wide range of OSS products including browsers (Mozilla Firefox and Camino), e-mail clients (Mozilla Thunderbird), bug tracking system (Bugzilla) and rendering engines (Gecko) [10]. Our assumption is that by collecting bug reports from a set of organizations that produce a wide range of software for end-users, we will be able to increase the generalizability of our findings. For all three organizations namely, Gentoo Linux, LibreOffice, and Mozilla we use the Bugzilla API to collect bug reports. We collect all bug reports used for our paper on July 25, 2019.

3.2 Bug Report Filtering

We include vulnerabilities that are indexed by the National Vulnerability Database (NVD). Vulnerabilities in the NVD are indexed by the keyword 'CVE' [11]. Upon collection of the bug reports from the three organizations, we apply filtering criteria to identify which bug reports are related to a vulnerability. The filtering criteria are listed below:

- *Step-1:* We search for the keyword 'CVE' in title, description, and comments for each collected bug report.
- *Step-2:* From the search results, we manually examine if a bug report actually is related to a vulnerability indexed by the NVD.

Upon completion of this step we will obtain a set of bug reports that are related to a vulnerability indexed by the NVD. Each collected bug report maps to a CVE. As bug reports can be duplicated, multiple bug reports can map to the same CVE.

3.3 Qualitative Rating to Construct Strategy Oracle Dataset

We construct an oracle dataset to identify if a bug report contains information on what strategy was adopted to discover a vulnerability. We use raters to identify if a bug report contains strategies to discover a vulnerability. In our paper, a rater is a person who is knowledgeable in software security, and performs qualitative analysis. Each rater, who are experienced in software security and bug report, individually looks at a bug report and determines if the bug report provides the strategy on how the reported vulnerability was discovered. A rater determines a bug report to contain a strategy to discover the vulnerability if each of the following criteria is satisfied:

- *Criteria-1:* The bug report includes text patterns that indicate whether the reporter is describing on how the vulnerability is discovered. The rater manually examines if the bug report includes any of the following text patterns: 'steps', 'reproduce', and 'observed behavior';
- *Criteria-2:* The bug report includes a sequence of steps on how the vulnerability was discovered; and
- *Criteria-3:* The bug report provides output of the conducted activities in the form of attachments, such as console output and screenshots.

We report the agreement rate by calculating Cohen's Kappa [9] to record the agreement level between raters. We follow Landis and Koch [22]'s interpretations to interpret Cohen's Kappa values.

[7] https://www.gentoo.org/
[8] https://www.libreoffice.org/
[9] https://www.mozilla.org/en-US/
[10] http://kb.mozillazine.org/Summary_of_Mozilla_products
[11] https://nvd.nist.gov/vuln

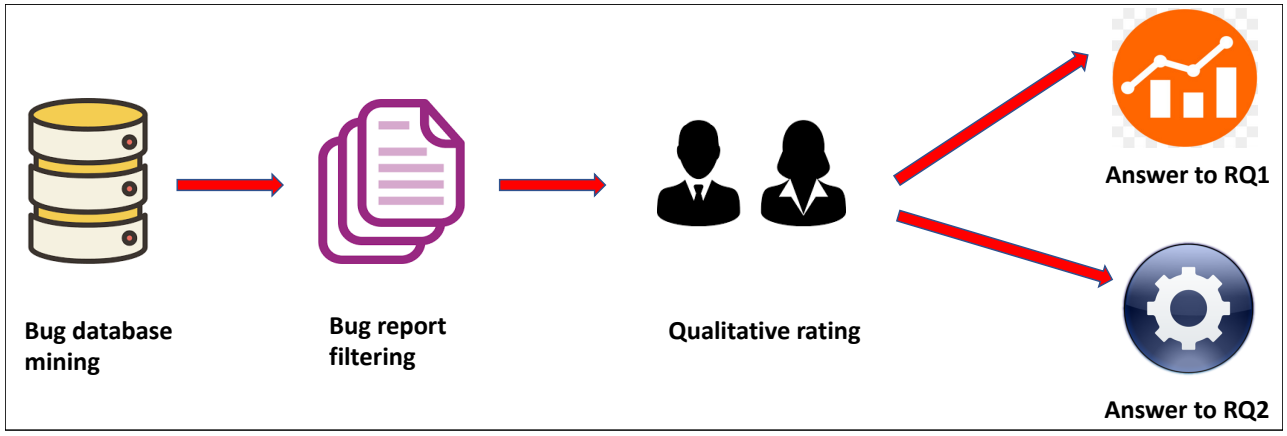

Figure 2: Steps to conduct our research study.

Table 1: Cohen's Kappa Interpretation by Landis and Koch [22]

Cohen's Kappa	Interpretation
< 00.0	'Poor agreement'
0.00~0.20	'Slight agreement'
0.21~0.40	'Fair agreement'
0.41~0.60	'Moderate agreement'
0.61~0.80	'Substantial agreement'
0.81~1.00	'Almost perfect agreement'

Interpretation of Landis and Koch [22] for Cohen's Kappa is listed in Table 1.

The raters may disagree. We allocate another rater who is experienced in bug reports and software security to resolve disagreements. In the case of disagreements, the resolver's decision is final.

3.4 Answer to RQ1: How frequently do vulnerability discovery strategies appear in bug reports?

Answer to RQ1 demonstrates evidence on whether or not bug reports contain strategies that are used to discover vulnerabilities. We answer RQ1 by using the oracle dataset constructed from Section 3.3. The oracle dataset provides a mapping on which bug reports include a strategy to discover a vulnerability. From the mapping we can quantify how many bug reports include a vulnerability discovery strategy. We use two metrics to quantify frequency:

- count of bug reports that include a strategy to discover a vulnerability; and
- percentage of bug reports that include vulnerability discovery strategies. We use Equation 1 to compute percentage of bug reports that include a vulnerability discovery strategy.

$$\text{Proportion of Bug Reports}(\%) = \frac{\text{\# of bug reports that include a vulnerability discovery strategy}}{\text{total \# of bug reports in the dataset}} \quad (1)$$

3.5 Answer to RQ2: How can we automatically identify bug reports that include vulnerability discovery strategies?

Using raters to identify bug reports that include vulnerability discovery strategies require manual effort and may not scale when analyzing large amount of bug reports. We hypothesize automated techniques that leverage text processing and machine learning can help in identifying bug reports with vulnerability discovery strategies. Similar to prior work [8] that has investigated reproducibility of bugs from bug report content, we implement two approaches that we describe below:

3.5.1 ***Regular Expression-based Approach.*** We hypothesize that bug report descriptions or comments will include certain text patterns that can be automatically analyzed to identify the presence of vulnerability discovery strategies. We use Figure 3 to illustrate our hypothesis even further. From Figure 3 we observe a bug report [4] comment to describe steps of actions to reproduce a vulnerability. The steps to reproduce follow a certain pattern: first, the 'Steps To Reproduce' keyword is used to express that the following steps were used to reproduce the vulnerability. Next, , we describe how the vulnerability is discovered by using an itemized list of actions ('1., 2., 3., 4.'). From the presented example in Figure 3, we observe that text patterns that are reflective of vulnerability discovery strategies exist in bug reports, and we can rely on regular expressions to capture these patterns.

We rely on regular expressions provided by Chaparro et al. [8] to derive the regular expressions needed to extract the text patterns related to vulnerability discovery strategies. We rely on Chaparro et al. [8]'s regular expressions as these regular expressions are systematically derived from bug reports that express steps of actions on how to reproduce a bug. We present the regular expression that we use to automatically detect vulnerability discovery strategy in Figure 4. Our assumption is that by using the same regular expression we can also effectively identify vulnerability discovery strategies in bug reports. We evaluate our assumptions further using three measures to determine the performance of our regular expression-based approach, which are described below:

- Precision: Precision measures the proportion of bug reports that include vulnerability discovery strategies given that the model predicts to include vulnerability discovery strategies. We use Equation 2 to calculate precision.

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

- Recall: Recall measures the proportion of bug reports that include vulnerability discovery strategies and also correctly predicted by the prediction model. We use Equation 3 to calculate recall.

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

- F-Measure: F-Measure is the harmonic mean of precision and recall. Increase in precision, often decreases recall, and vice-versa [25]. F-Measure provides a composite score of precision and recall, and is high when both precision and recall is high.

$$F - Measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4}$$

In Equations 2 and 3, TP, FP, and FN respectively, stands for true positive, false positive and false negative.

Steps To Reproduce:
1. open Nightly with e10s enabled
2. open the attached HTML
3. select "THIS" text in the file
4. drag and drop the text to empty area of tabs bar

Figure 3: Example text pattern in bug report that expresses a step of actions to discover a vulnerability ('CVE-2017-7812').

```
(step(s)?|how) to (reproduce|
recreate|create|replicate)|
(step|repro|repro step|
step to repro)|follow(ing)?
(scenario(s)?|step(s)?):|
^\d+\-+.+|^(\[\d+(\w+)?\]).+|
^(\(\d+(\w+)?\)).+|
^(\{\d+(\w+)?\}).+|
^step\d+ \:.* |^\d+(\.|\)) .*
```

Figure 4: Regular expression used to automatically identify bug reports with vulnerability discovery strategies.

3.5.2 ***Machine Learning-based Approach.*** We hypothesize that by extracting text features and applying statistical learners we can build models that will automatically predict what bug reports include vulnerability discovery strategies. We summarize the steps to construct prediction models in Figure 5. We describe the steps to construct prediction models below:

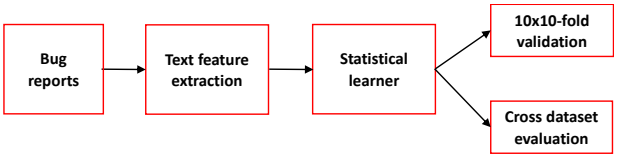

Figure 5: Steps to construct prediction models to automatically identify the presence of vulnerability discovery strategies in bug reports.

Index	Text feature	Frequency
1	open	1
2	nightly	1
3	with	1
4	e10s	1
5	enabled	1
6	open nightly	1
7	nightly with	1
8	with e10s	1
9	e10s enabled	1
10	open nightly with	1
11	nightly with e10s	1
12	with e10s enabled	1

Figure 6: A hypothetical example to demonstrate text feature extraction process described in Section 3.5.2. We extract uni-grams, bi-grams, and tri-grams from bug report content.

Step-1: Text feature extraction For our machine learning-based approach we extract text features from the bug report comments. For text feature extraction we apply uni-grams, bi-grams, and tri-grams [21]. Uni-grams, bi-grams, and tri-grams are special cases of n-grams i.e. a contiguous sequence of n tokens from a given sample of text [21]. For uni-gram, bi-gram, and tri-gram, the value

of *n* is respectively, one, two, and three. Our assumption is that by capturing these sequences of tokens we may capture text features that can be used to predict the presence of vulnerability discovery strategies. In the case of uni-grams, we assume to capture special keywords that could be used to predict the presence of vulnerability discovery strategies in a bug report.

We further demonstrate the process of text feature extraction using a hypothetical example: "*open nightly with e10s enabled*". The sentence is extracted from a Mozilla bug report, which maps to a vulnerability indexed by the NVD (CVE-2017-7812) [4]. With the application of our text feature extraction process we extract 12 features indexed as 1-12 in Figure 6. Text features indexed as 1-5 are uni-grams, 6-9 are bi-grams, and 10-12 are tri-grams. From the 12 text features we construct a feature matrix, which is used as input to the statistical learners.

Step-2: Apply statistical learners Researchers use statistical learners to build prediction models that learn from historic data and make prediction decisions on unseen data. We use Scikit Learn API [31] to construct prediction models using statistical learners. We use six statistical learners that we briefly describe, and reasons for selecting these learners, as following:

- *Classification and Regression Tree (CART)*: CART generates a tree based on the impurity measure, and uses that tree to provide decisions based on input features [3]. We select CART because this learner does not make any assumption on the distribution of features, and is robust to model overfitting [3, 36].
- *K Nearest Neighbor (KNN)*: The KNN classification technique stores all available prediction outcomes based on training data and classifies test data based on similarity measures. We select KNN because prior research has reported that defect prediction models that use KNN perform well [20].
- *Logistic Regression (LR)*: LR estimates the probability that a data point belongs to a certain class, given the values of features [16]. LR provides good performance for classification if the features are roughly linear [16]. We select LR because this learner performs well for classification problems [16], such as defect prediction [33] and fault prediction [20].
- *Support Vector Machine (SVM)*: Support vector machines predict labels by quantifying the amount of separation for features between multiple classes. We select SVM because prior research [10, 27] has reported SVMs to perform well for text-related classification.
- *Random Forest (RF)*: RF is an ensemble technique that creates multiple classification trees, each of which is generated by taking random subsets of the training data [2, 36]. Unlike LR, RF does not expect features to be linear for good classification performance. Researchers [19] recommended the use of statistical learners that uses ensemble techniques to build defect prediction models.
- *Deep Neural Network (DNN)*: Deep neural network is a variant of artificial neural network (ANN), where the count of hidden layers used within the ANN is multiple, and can vary from three to thousands of hidden layers [23]. Our implementation of DNN uses five parameters: first, we use a multi-layer perceptron with 5 hidden layers, where each layer is fully connected to the next one. Second, we use the backpropagation algorithm [39] for training. Third, we use cross-entropy loss function for classification. Fourth, we use the rectified linear unit function as our activation function, and fifth we train our model for 200 epochs.

Table 2: Selection of Bug Reports for Analysis

	Gentoo	Libre	Mozilla
Initial count	729	908	5,336
Criteria-1 (Keyword analysis)	566	89	1,151
Criteria-2 (Manual analysis)	9	32	536
Final bug report count	9	32	536

Prediction performance measures: As shown in Section 3.5.1, similar to our evaluation of regular expression-based approach, we use three measures to evaluate the prediction performance of the constructed models: precision, recall, and F-measure.

Step-3: Evaluation We use two approaches to evaluate our constructed prediction models, which we describe below:

- 10×10-fold validation: We use 10×10-fold validation to evaluate our prediction models. We use this approach by randomly partitioning the dataset into 10 equal sized subsamples or folds [36]. The performance of the constructed prediction models is tested by using 9 of the 10 folds as training data, and the remaining fold as test data. Similar to prior research [19], we repeat the 10-fold validation 10 times to avoid prediction errors. We report the median prediction performance score of the 10 runs.
- Cross Dataset Prediction: For cross dataset prediction, we build prediction models by training the statistical learners on one dataset and use another dataset for testing. Along with 10×10-fold validation we use cross dataset evaluation because, the cross dataset evaluation can provide evidence on how generalizable the machine learning-based approach is, and if there are similarities between datasets with respect to text features that are reflective of vulnerability discovery strategies. The three datasets yield a total of six train and test combinations. Similar to prior research [18], we repeat the cross dataset prediction procedure 10 times and report the median prediction performance score of the 10 runs.

4 EMPIRICAL FINDINGS

In this section we report our empirical findings.

4.1 Bug Database Mining

By using the Bugzilla API, we collect 15010, 30000, and 32170 bug reports respectively, for Gentoo Linux, LibreOffice, and Mozilla.

4.2 Bug Report Filtering

We collect 9, 32, and 536 bug reports that map to a CVE indexed in NVD. In Table 2, we report a complete breakdown of how many bug reports are filtered using our criteria mentioned in Section 3.2.

Table 3: Agreement Level between Raters to Construct Oracle Dataset

Property	Gentoo	LibreOffice	Mozilla
Agreement	99.7%	98.1%	99.7%
Cohen's κ	0.67	0.74	0.68
Interpretation	Substantial	Substantial	Substantial

Table 4: Answer to RQ1: Bug Reports with Strategies to Discover Vulnerabilities

Metric	Gentoo	Libre	Mozilla
Count	9	29	425
Proportion of Bug Reports(%)	1.2	3.2	7.9

4.3 Qualitative Rating to Construct Strategy Oracle Dataset

We allocate two raters, the first and second authors of the paper to conduct qualitative rating. Both raters individually, examined bug reports for all three datasets to identify which bug reports include strategies to discover vulnerabilities. We report summary statistics of the qualitative rating step in Table 3. The Cohen's Kappa is listed in the 'Cohen's κ' row. The 'Agreement' row reports the count of bug reports for which we observe agreements between the rating of the first and the second authors. For example, the two authors agreed on the rating for 99.7% bug reports for Gentoo, and the Cohen's κ is 0.67.

Our results show that the agreement between the raters varied from 98.1%~99.7%. According to Landis and Koch [22], the agreement rate is 'Substantial' indicating high agreement between the two raters. For the disagreements we use the last author as the resolver. Upon resolving disagreements we construct an oracle dataset that we use to answer RQ1 and RQ2.

4.4 Answer to RQ1: How frequently do vulnerability discovery strategies appear in bug reports?

We examine bug reports to see if the reports contain strategies used in vulnerability discovery. The number of bug reports that included vulnerability discovery strategies varied from 1.2%~7.9%. Proportion-wise the Mozilla dataset has the highest amount of bug reports that include vulnerability discovery strategies: 7.9% of the 5,336 bug reports downloaded from Mozilla maps to a vulnerability, and also include the vulnerability discovery strategy in the bug report. We provide detailed answers in Table 4, where the 'Proportion (%)' metric shows the proportion of bug reports that include strategies to discover vulnerabilities. Based on our answers to RQ1 we can conclude that bug reports can be used to identify strategies for vulnerability discovery.

4.5 Answer to RQ2: How can we automatically identify bug reports that include vulnerability discovery strategies?

We answer RQ2 in this section. We present the results using regular expression in Table 5, where we report the precision, recall, and F-measure for three datasets: Gentoo, Libre, and Mozilla. Considering recall, we observe the regular expression approach to perform the best for the Mozilla dataset with a recall of 0.83. Considering F-measure, LibreOffice performs best: for LibreOffice we observe a F-measure of 0.31. We do not observe precision to be >= 0.19 for any of the datasets, which suggests regular expression-based approaches can generate a lot of false positives.

Table 5: Answer to RQ2: Results for the Regular Expression Approach

Metric	Gentoo	Libre	Mozilla
Precision	0.14	0.19	0.16
Recall	0.37	0.79	0.83
F-measure	0.20	0.31	0.27

We present the findings related to machine learning approach with 10×10-fold validation in Table 6. The 'Learner' lists the names of the learner for which we report precision, recall, and F-measure for the three datasets: Gentoo (GENT), LibreOffice (LIBR), and Mozilla (MOZI). We observe LR to provide the highest F-measure for Mozilla. Precision is the highest for Mozilla when CART is applied. The performance is the worst for the Gentoo dataset: no learner is able to detect the presence of vulnerability discovery strategy for the dataset. For the Gentoo dataset, precision, recall, and F-measure are respectively, 0.0, 0.0, and 0.0 for all six learners.

We report the cross dataset prediction results in Tables 7, 8, and 9. As described in Section 3.5.2, for cross dataset evaluation, we construct a model with one dataset, and test the performance of the model using another dataset. Tables 7, 8, and 9 respectively, describe the prediction results, when we train models with the Gentoo, LibreOffice, and Mozilla datasets. We observe the prediction performance to be lowest when the model is trained with the Gentoo dataset. When trained with the Mozilla dataset, the precision, recall, and F-measure of the Gentoo dataset are respectively, 1.00, 0.71, and 0,74, which are higher than that of the 10×10-fold validation. Our findings show that when trained with the Mozilla dataset the prediction performance is higher for the other datasets implying that the text features extracted from the Mozilla dataset can be used to predict the presence of vulnerability discovery strategies in other datasets.

5 DISCUSSION

We discuss the findings of our paper in this section:

5.1 On the Value of Bug Reports to Advance Understanding of Vulnerabilities

Results from Table 4 show that OSS bug reports contain vulnerability discovery strategies. Our reported frequency of vulnerability discovery strategy is lower than that of bug discovery strategy as reported in prior research [8]. Chaparro et al. [8] studied 2,912 bug reports and observed 51.9% of the studied bug reports to include strategies on how a bug was discovered.

5.2 Reporting of Vulnerability Discovery Strategies in Bug Reports

As shown in Table 4, the proportion of bug reports that include vulnerability discovery strategies vary from 1.2%~7.9%. One possible explanation can be attributed to lack of context on how a rater

Table 6: Answer to RQ2: Results for Machine Learning-based Approach with 10×10-fold Validation

Learner	Precision			Recall			F-measure		
	GENT	LIBR	MOZI	GENT	LIBR	MOZI	GENT	LIBR	MOZI
CART	0.00	0.67	0.90	0.00	0.67	0.71	0.00	0.67	0.79
DNN	0.00	0.00	0.82	0.00	0.00	0.77	0.00	0.00	0.79
KNN	0.00	0.00	0.64	0.00	0.00	0.11	0.00	0.00	0.18
LR	0.00	0.50	0.87	0.00	0.33	0.78	0.00	0.40	0.82
RF	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
SVM	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 7: Answer to RQ2: Results for Machine Learning-based Approach with Cross-dataset Evaluation. Training dataset is Gentoo.

	Precision		Recall		F-measure	
	LIBR	MOZI	LIBR	MOZI	LIBR	MOZI
CART	0.00	0.00	0.00	0.00	0.00	0.00
DNN	0.00	0.26	0.00	0.11	0.00	0.15
KNN	0.00	0.00	0.00	0.00	0.00	0.00
LR	0.00	0.12	0.00	0.00	0.00	0.00
RF	0.00	0.00	0.00	0.00	0.00	0.00
SVM	0.00	0.00	0.00	0.00	0.00	0.00

Table 8: Answer to RQ2: Results for Machine Learning-based Approach with Cross-dataset Evaluation. Training dataset is LibreOffice.

	Precision		Recall		F-measure	
	GENT	MOZI	GENT	MOZI	GENT	MOZI
CART	0.00	0.53	0.00	0.65	0.00	0.59
DNN	0.00	0.59	0.00	0.09	0.00	0.15
KNN	0.00	0.00	0.00	0.00	0.00	0.00
LR	0.00	0.50	0.00	0.04	0.00	0.07
RF	0.00	0.00	0.00	0.00	0.00	0.00
SVM	0.00	0.00	0.00	0.00	0.00	0.00

Table 9: Answer to RQ2: Results for Machine Learning-based Approach with Cross-dataset Evaluation. Training dataset is Mozilla.

	Precision		Recall		F-measure	
	GENT	LIBR	GENT	LIBR	GENT	LIBR
CART	0.77	0.67	0.71	0.07	0.74	0.12
DNN	0.57	0.71	0.71	0.17	0.63	0.27
KNN	1.00	0.00	0.25	0.00	0.40	0.00
LR	0.83	0.70	0.64	0.24	0.36	0.73
RF	0.00	0.00	0.02	0.00	0.04	0.00
SVM	1.00	0.00	0.00	0.00	0.00	0.00

perceives the content of the bug report. Description of the discovery strategy might make more sense to the person who performed the discovery but due to lack of context, a rater, who is well-versed in software security may not understand the steps of actions on how to perform the discovery. A set of guidelines on proper reporting of vulnerability discovery strategies can be helpful for security researchers who do not actively contribute to OSS projects, but need to report vulnerabilities.

Lack of context to reproduce a bug is common in software development. In 2016, developers who host their projects on GitHub, signed and sent a petition to GitHub, where they stated that bug reports "... are often filed missing crucial information like reproduction steps" [12]. Researchers [15, 41, 42] have reported the negative impact of incomplete bug reports. For example, non-reproducible bugs [15], unfixed bugs [41], and delayed bug resolution [42] are often attributed to incomplete information related to bug descriptions.

5.3 Automation

Our findings reported in Section 4.5 provide evidence that the two automation strategies we used have limitations. As shown in Table 5, using regular expressions, the highest F-measure is 0.31, which is subject to improvement. With 10×10-fold validation the highest F-measure is 0.82, but 0.00 for Gentoo. Furthermore, from Section 4.5 we observe that when building models with the Mozilla dataset, the prediction performance measure is relatively higher for cross-dataset prediction results. One possible explanation can be the text patterns that appear in the Mozilla dataset are capable of separating out bug reports that include vulnerability discovery strategies and bug reports that do not. Our findings suggest that automated detection of vulnerability discovery strategy in bug reports is related to the textual content presented in the bug reports.

5.4 Future Research

Findings from our paper provide opportunities to pursue the following research directions:

- **Strategy Mining**: Researchers can use bug reports to identify what strategies are used to discover vulnerabilities. A synthesis of vulnerability discovery strategies can be helpful for IT organizations who do not have security experts in their team, but are interested in discovery and repair of software vulnerabilities before they are released to the end-users. Munaiah et al. [28] stressed the importance of identifying vulnerability discovery strategies from available software artifacts: "*As the complexity of the software system increases, speculating the different ways in which an attacker could exploit a vulnerability becomes a daunting task. The key, therefore, is to leverage empirical information to characterize the typical ways in which attackers tend to discover and exploit vulnerabilities.*".
  While pursuing this research project, researchers may benefit from applying qualitative analysis with multiple raters, as automated techniques, such as the use of regular expressions and text feature-based prediction models are limited with respect to detection accuracy. In addition, researchers can compare and contrast the findings obtained from bug reports to those obtained

[12] https://github.com/dear-github/dear-github

from other data sources, such as capture the flag (CTF) competitions. Recently, researchers [28] have mined CTF data and observed that while discovering vulnerabilities, CTF participants discover vulnerabilities in a sequential manner, which is similar to our definition of vulnerability discovery strategy. Also, researchers can investigate who performs the discovery strategies, their experience in vulnerability discovery, and how difficult it is to perform the discovery. Our findings provide the foundation for further research in the domain of vulnerability understanding and characterization.

- **Improving Automated Detection Strategies**: We have discussed how our automated approaches to detect vulnerability discovery strategy is limited with respect to detection accuracy. We advocate for future research that will investigate if other text mining techniques, such as topic modeling [38], word2vec [26], and sequential text pattern mining [24] can effectively detect presence of vulnerability discovery strategies.
  Currently, our automated approach does not show prediction performance that is consistent across all datasets. For example, for the Gentoo dataset, both the machine learning approach and regular expression-based approach showed lower detection accuracy results compared to those of Mozilla and LibreOffice. Apart from text mining and regular expressions, researchers can investigate what other techniques can be used to build a universal classifier that is capable of detecting the presence of vulnerability discovery strategies for multiple datasets collected from a wide range of domains, such as browsers (Mozilla) and operating systems (Gentoo).
- **Towards Better Reporting of Vulnerability Discovery**: Our findings show evidence that vulnerability discovery strategies may not be adequately reported in bug reports. For OSS projects, practitioners who report vulnerability discovery strategies might be peripheral contributors [34] i.e. contributors who do not regularly engage in regular software development and maintenance tasks, and may not be aware of the practices on how to report vulnerability discovery strategies. Concrete guidelines could be helpful in proper reporting of vulnerability discovery strategies.

6 THREATS TO VALIDITY

We discuss the limitations of our paper as following:

- **Conclusion Validity**: All of our findings are derived from three datasets, which are constructed by the two raters. The data construction process is susceptible to rater judgment. The raters' experience can bias the process of identifying what bug reports include strategies to discover vulnerabilities. We mitigate this limitation by assigning at least two raters. We also use a resolver to resolve the disagreements between the two raters.
  Prior work [17, 37] has underlined the importance of tuning parameters of statistical learners to get better prediction performance. While building prediction models we didn't tune the statistical learners. For CART, KNN, LR, RF, and SVM we rely on the default parameters provided by the Scikit Learn API [31]. For DNN, the parameters are determined by the first author's judgment. We acknowledge that not tuning parameters of statistical learners might influence the prediction performance of the constructed models.
- **Construct Validity**: We use raters to construct the oracle dataset. Our process is susceptible to mono-method bias where subjective judgment of raters can influence the findings. We mitigate this threat by using two raters and one resolver.
- **External Validity**: Our results are subject to external validity as we used three datasets collected from the OSS domain. Findings from our paper may be limited and may not generalize to other datasets. We mitigate this limitation by using datasets from three organizations who deliver a variety of software products to the end-users.
- **Internal Validity**: From the bug reports we extract comments and descriptions to collect necessary text to determine if a bug report includes strategies to discover vulnerabilities. The descriptions and comments available in the bug reports may not be comprehensive to provide enough context to determine if a strategy is present. We rely on CVEs reported in the bug reports to identify those that are related to vulnerabilities. The bug reports may include discussion on vulnerabilities that are not confirmed and indexed in NVD. Our analysis may miss latent vulnerabilities and unconfirmed vulnerabilities, which could influence the results.

7 CONCLUSION

Identifying and characterizing the nature of vulnerabilities have gained a lot of interest amongst researchers. However, data sources to study vulnerabilities can be limiting. One approach to mitigate this limitation is to investigate if bug reports include descriptions of how vulnerabilities are discovered. We hypothesized that bug reports could include descriptions of how vulnerabilities are discovered, which could help researchers to further identify what strategies practitioners execute to discover vulnerabilities for OSS. We evaluated our hypothesis by conducting an empirical analysis with 729, 908, and 5336 bug reports respectively, collected from Gentoo, LibreOffice, and Mozilla.

We observe OSS bug reports to include descriptions on how to discover vulnerability strategies: the proportion of bug reports that included vulnerability discovery strategies varied from 1.2%∼7.9%. Based on our answers to RQ1 we can conclude that software bug reports include information that can be used to identify strategies for vulnerability discovery. Our findings also suggest that automated detection of vulnerability discovery strategy is dependent on the text features that we mine from bug reports, and is sensitive to what dataset is being used. For example, we observe the highest detection performance for Mozilla using the machine learning approach. Using the regular expression-based approach we observe the highest detection performance for LibreOffice. Based on our empirical study we recommend researchers to apply qualitative analysis to identify vulnerability discovery strategies, as automated approaches may be limiting to detect the presence of vulnerability discovery strategies in OSS bug reports. We hope our paper will advance research in the area focused on characterizing vulnerabilities in OSS projects.

ACKNOWLEDGMENTS

We thank members of the PASER group at Tennessee Technological University for their valuable feedback on the paper.

REFERENCES

[1] N. Ali, A. Sabané, Y. Guéhéneuc, and G. Antoniol. 2012. Improving Bug Location Using Binary Class Relationships. In *2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation*. 174–183. https://doi.org/10.1109/SCAM.2012.26

[2] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32. https://doi.org/10.1023/A:1010933404324

[3] Leo Breiman et al. 1984. *Classification and Regression Trees* (1st ed.). Chapman & Hall, New York. 358 pages. http://www.crcpress.com/catalog/C4841.htm

[4] Bugzilla. 2017. Web content can open local files by hooking drag and drop to outside of content. https://bugzilla.mozilla.org/show_bug.cgi?id=1379842. [Online; accessed 21-December-2019].

[5] Oscar Chaparro, Carlos Bernal-Cardenas, Jing Lu, Kevin Moran, Andrian Marcus, Massimiliano Di Penta, Denys Poshyvanyk, and Vincent Ng. 2019. Assessing the Quality of the Steps to Reproduce in Bug Reports. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019)*. ACM, New York, NY, USA, 86–96. https://doi.org/10.1145/3338906.3338947

[6] O. Chaparro, J. M. Florez, and A. Marcus. 2016. On the Vocabulary Agreement in Software Issue Descriptions. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 448–452. https://doi.org/10.1109/ICSME.2016.44

[7] Oscar Chaparro, Juan Manuel Florez, and Andrian Marcus. 2019. Using bug descriptions to reformulate queries during text-retrieval-based bug localization. *Empirical Software Engineering* 24, 5 (01 Oct 2019), 2947–3007. https://doi.org/10.1007/s10664-018-9672-z

[8] Oscar Chaparro, Jing Lu, Fiorella Zampetti, Laura Moreno, Massimiliano Di Penta, Andrian Marcus, Gabriele Bavota, and Vincent Ng. 2017. Detecting Missing Information in Bug Descriptions. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017)*. ACM, New York, NY, USA, 396–407. https://doi.org/10.1145/3106237.3106285

[9] Jacob Cohen. 1960. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement* 20, 1 (1960), 37–46. https://doi.org/10.1177/001316446002000104 arXiv:http://dx.doi.org/10.1177/001316446002000104

[10] Fabrice Colas and Pavel Brazdil. 2006. Comparison of SVM and Some Older Classification Algorithms in Text Classification Tasks. In *Artificial Intelligence in Theory and Practice*, Max Bramer (Ed.). Springer US, Boston, MA, 169–178.

[11] National Vulnerability Database. 2019. NVD-Vulnerabiliites. https://nvd.nist.gov/vuln. [Online; accessed 22-August-2019].

[12] Tezcan Dilshener, Michel Wermelinger, and Yijun Yu. 2016. Locating Bugs Without Looking Back. In *Proceedings of the 13th International Conference on Mining Software Repositories (MSR '16)*. ACM, New York, NY, USA, 286–290. https://doi.org/10.1145/2901739.2901775

[13] PC Doctor Inc. 2019. PC Diagnostic & System Information Solutions Pre-installed on PC/Android Systems. https://www.pc-doctor.com/solutions/oems. [Online; accessed 14-Nov-2019].

[14] DZone. 2019. Millions of Dell PCs Vulnerable to Flaw in Third-Party Component. https://threatpost.com/millions-of-dell-pcs-vulnerable-to-flaw-in-third-party-component/145833/. [Online; accessed 08-Aug-2019].

[15] Mona Erfani Joorabchi, Mehdi Mirzaaghaei, and Ali Mesbah. 2014. Works for Me! Characterizing Non-reproducible Bug Reports. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014)*. ACM, New York, NY, USA, 62–71. https://doi.org/10.1145/2597073.2597098

[16] David Freedman. 2005. *Statistical Models : Theory and Practice*. Cambridge University Press.

[17] Wei Fu, Tim Menzies, and Xipeng Shen. 2016. Tuning for software analytics: Is it really necessary? *Information and Software Technology* 76 (2016), 135 – 146. https://doi.org/10.1016/j.infsof.2016.04.017

[18] Takafumi Fukushima, Yasutaka Kamei, Shane McIntosh, Kazuhiro Yamashita, and Naoyasu Ubayashi. 2014. An Empirical Study of Just-in-time Defect Prediction Using Cross-project Models. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014)*. ACM, New York, NY, USA, 172–181. https://doi.org/10.1145/2597073.2597075

[19] Baljinder Ghotra, Shane McIntosh, and Ahmed E. Hassan. 2015. Revisiting the Impact of Classification Techniques on the Performance of Defect Prediction Models. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1 (ICSE '15)*. IEEE Press, Piscataway, NJ, USA, 789–800. http://dl.acm.org/citation.cfm?id=2818754.2818850

[20] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. 2012. A Systematic Literature Review on Fault Prediction Performance in Software Engineering. *IEEE Transactions on Software Engineering* 38, 6 (Nov 2012), 1276–1304. https://doi.org/10.1109/TSE.2011.103

[21] Daniel Jurafsky and James H. Martin. 2009. *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[22] J. Richard Landis and Gary G. Koch. 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics* 33, 1 (1977), 159–174. http://www.jstor.org/stable/2529310

[23] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[24] David D Lewis and William A Gale. 1994. A sequential algorithm for training text classifiers. In *SIGIR'94*. Springer, 3–12.

[25] Tim Menzies, Alex Dekhtyar, Justin Distefano, and Jeremy Greenwald. 2007. Problems with Precision: A Response to "Comments on 'Data Mining Static Code Attributes to Learn Defect Predictors'". *IEEE Trans. Softw. Eng.* 33, 9 (Sept. 2007), 637–640. https://doi.org/10.1109/TSE.2007.70721

[26] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 3111–3119. http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

[27] Vikramjit Mitra, Chia-Jiu Wang, and Satarupa Banerjee. 2007. Text Classification: A Least Square Support Vector Machine Approach. *Appl. Soft Comput.* 7, 3 (June 2007), 908–914. https://doi.org/10.1016/j.asoc.2006.04.002

[28] N. Munaiah, A. Rahman, J. Pelletier, L. Williams, and A. Meneely. 2019. Characterizing Attacker Behavior in a Cybersecurity Penetration Testing Competition. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–6. https://doi.org/10.1109/ESEM.2019.8870147

[29] Department of Homeland Security. 2019. Information Technology Sector. https://www.dhs.gov/cisa/information-technology-sector. [Online; accessed 15-Nov-2019].

[30] Science of Security and Privacy. 2010. Cyber Security – Is Science Possible? https://cps-vo.org/node/624. [Online; accessed 24-December-2019].

[31] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* 12 (Nov. 2011), 2825–2830. http://dl.acm.org/citation.cfm?id=1953048.2078195

[32] F. Peters, T. T. Tun, Y. Yu, and B. Nuseibeh. 2019. Text Filtering and Ranking for Security Bug Report Prediction. *IEEE Transactions on Software Engineering* 45, 6 (June 2019), 615–631. https://doi.org/10.1109/TSE.2017.2787653

[33] Foyzur Rahman and Premkumar Devanbu. 2013. How, and Why, Process Metrics Are Better. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*. IEEE Press, Piscataway, NJ, USA, 432–441. http://dl.acm.org/citation.cfm?id=2486788.2486846

[34] Pankaj Setia, Balaji Rajagopalan, Vallabh Sambamurthy, and Roger Calantone. 2012. How peripheral developers contribute to open-source software development. *Information Systems Research* 23, 1 (2012), 144–163.

[35] J. D. Strate and P. A. Laplante. 2013. A Literature Review of Research in Software Defect Reporting. *IEEE Transactions on Reliability* 62, 2 (June 2013), 444–454. https://doi.org/10.1109/TR.2013.2259204

[36] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. 2005. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[37] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. 2016. Automated Parameter Optimization of Classification Techniques for Defect Prediction Models. In *Proceedings of the 38th International Conference on Software Engineering (ICSE '16)*. ACM, New York, NY, USA, 321–332. https://doi.org/10.1145/2884781.2884857

[38] Hanna M Wallach. 2006. Topic modeling: beyond bag-of-words. In *Proceedings of the 23rd international conference on Machine learning*. ACM, 977–984.

[39] Bernard Widrow and Michael A Lehr. 1990. 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proc. IEEE* 78, 9 (1990), 1415–1442.

[40] Yu Zhao, Kye Miller, Tingting Yu, Wei Zheng, and Minchao Pu. 2019. Automatically Extracting Bug Reproducing Steps from Android Bug Reports. In *Reuse in the Big Data Era*, Xin Peng, Apostolos Ampatzoglou, and Tanmay Bhowmik (Eds.). Springer International Publishing, Cham, 100–111.

[41] Thomas Zimmermann, Nachiappan Nagappan, Philip J. Guo, and Brendan Murphy. 2012. Characterizing and Predicting Which Bugs Get Reopened. In *Proceedings of the 34th International Conference on Software Engineering (ICSE '12)*. IEEE Press, Piscataway, NJ, USA, 1074–1083. http://dl.acm.org/citation.cfm?id=2337223.2337363

[42] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss. 2010. What Makes a Good Bug Report? *IEEE Transactions on Software Engineering* 36, 5 (Sep. 2010), 618–643. https://doi.org/10.1109/TSE.2010.63