Digital Object Identifier

Security Bug Report Usage for Software Vulnerability Research: A Systematic Mapping Study

FARZANA AHAMED BHUIYAN[1], MD BULBUL SHARIF[2], AND AKOND RAHMAN.[3]
Department of Computer Science, Tennessee Technological University, Cookeville, Tennessee, USA
[1] fbhuiyan42@tntech.edu, [2] msharif42@tntech.edu, [3] arahman@tntech.edu

"This work was supported in part by the the National Science Foundation (NSF) award # 2026869 and the Cybersecurity Education, Research and Outreach Center (CEROC) at Tennessee Technological University."

ABSTRACT *Context:* Security bug reports are reports from bug tracking systems that include descriptions and resolutions of security vulnerabilities that occur in software projects. Researchers use security bug reports to conduct research related to software vulnerabilities. A mapping study of publications that use security bug reports can inform researchers on (i) the research topics that have been investigated, and (ii) potential research avenues in the field of software vulnerabilities.
Objective: The objective of this paper is to help researchers identify research gaps related to software vulnerabilities by conducting a systematic mapping study of research publications that use security bug reports.
Method: We perform a systematic mapping study of research that use security bug reports for software vulnerability research by searching five scholar databases: (i) IEEE Xplore, (ii) ACM Digital Library, (iii) ScienceDirect, (iv) Wiley Online Library, and (v) Springer Link. From the five scholar databases, we select 46 publications that use security bug reports by systematically applying inclusion and exclusion criteria. Using qualitative analysis, we identify research topics investigated in our collected set of publications.
Results: We identify three research topics that are investigated in our set of 46 publications. The three topics are: (i) vulnerability classification; (ii) vulnerability report summarization; and (iii) vulnerability dataset construction. Of the studied 46 publications, 42 publications focus on vulnerability classification.
Conclusion: Findings from our mapping study can be leveraged to identify research opportunities in the domains of software vulnerability classification and automated vulnerability repair techniques.

INDEX TERMS bug report, software security, survey, systematic mapping study, vulnerability

I. INTRODUCTION

Daily news reports reveal the serious consequences of security vulnerabilities in software projects. For example, in 2019, the 'DELL PC Doctor' vulnerability impacted millions of Dell computers [1]. As another example, Experian, a consumer credit reporting service, experienced a security attack due to a vulnerability, affecting nearly 24 million South African customers and approximately 793,749 business organizations in August 2020 [1]. Since the beginning of 2020, more than 445 million cyberattacks have been reported, all of which exploited latent vulnerabilities [2].

The prevalence of vulnerabilities has motivated researchers [2]–[6] to systematically study vulnerabilities in software by mining security bug reports used in open source software (OSS) projects. Security bug reports are reports from bug tracking systems that include descriptions and resolutions of security vulnerabilities that occur in software projects. Information contained in security bug reports enables researchers to conduct an in-depth analysis of reported vulnerabilities and derive insights on how to pro-actively mitigate vulnerabilities.

Despite their use in software vulnerability research, a systematic synthesis of publications that use security bug reports remains under-explored. A systematic mapping study (SMS) of publications that use security bug reports for software vulnerabilities might be beneficial for researchers in the following manner: (i) provide a synthesis of the research topics that already have been addressed, (ii) provide an understand-

[1]https://www.znetlive.com/blog/top-10-cybersecurity-incidents-in-2020/
[2]https://www.helpnetsecurity.com/2020/04/29/2020-attack-rate/

ing of what properties of security bug reports can be mined to conduct vulnerability-related research, and (iii) identify research topics that might be of interest to researchers.

The objective of this paper is to help researchers identify research gaps related to software vulnerabilities by conducting a systematic mapping study of research publications that use security bug reports.

We answer the following research questions:

- **RQ1**: What research topics have been investigated in publications that use security bug reports for software vulnerability research?
- **RQ2**: How frequently do identified research topics appear in publications that use security bug reports for software vulnerability research?
- **RQ3**: Which properties of security bug reports are used in publications for software vulnerability research?
- **RQ4**: What automated techniques are used in publications that use security bug reports for software vulnerability research?

We perform an SMS following the guidelines of Petersen et al. [7] to synthesize publications that use security bug reports for software vulnerability research. First, we search five scholar databases namely, IEEE Xplore [3], ACM Digital Library [4], ScienceDirect [5], Wiley Online Library [6], and Springer Link [7]. Using two search strings, we obtain a set of 45,077 publications from the software engineering (SE) domain that were published from 2000 to August 2020. By systematically applying inclusion and exclusion criteria [8], we obtain 46 publications. Following Kitchenham's guidelines [9], we assess the quality of our set of 46 publications. We apply a qualitative analysis technique called open coding [10] to identify topics discussed in the collected publications. We also investigate temporal trends and techniques applied in publications that have used security bug reports.

Contributions: We list our contributions as follows:

- A list of research topics studied in publications that use security bug reports (Section V-A);
- An evaluation of the temporal trends for publications that use security bug reports (Section V-B);
- An evaluation of the quality of publications that use security bug reports (Section V); and
- A list of future research directions to share the vision and expand the horizon of software vulnerability research (Section VI-B).

We organize the rest of the paper as follows: in Section II we describe the necessary background and in Section III we describe related publications. We provide our research methodology in Section IV. We present our results in Section V and discuss possible implications of findings in Section VI. We list the limitations of our SMS in Section VII. Finally, we conclude our paper in Section VIII.

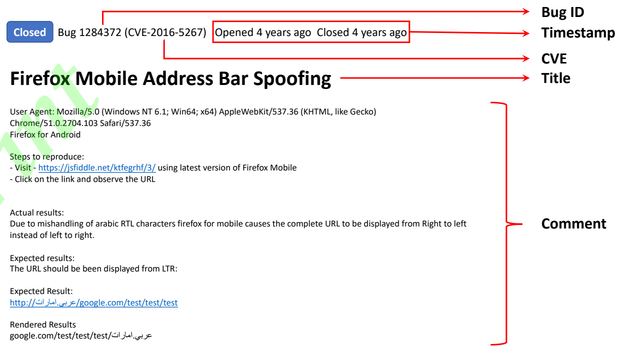
[3]http://ieeexplore.ieee.org/Xplore/home.jsp
[4]https://dl.acm.org/
[5]https://www.sciencedirect.com/
[6]https://onlinelibrary.wiley.com/
[7]https://link.springer.com/

II. BACKGROUND

We provide background information on security bug reports and SMS in this section.

A. BACKGROUND ON SECURITY BUG REPORTS

Security bug reports are reports from bug tracking systems that include descriptions and resolutions of security vulnerabilities that occur in software projects. Typical entities of bug reports include but are not limited to: (i) a title that summarizes the bug, (ii) a bug ID that is unique across all listed bugs, (iii) timestamp information that shows when the bug report was opened and closed; and (iv) comments that discuss how to reproduce the bug. We provide an annotated snapshot of a security bug report [11] retrieved from the Mozilla organization in Figure 1. The security bug report mentions a vulnerability, labeled as 'Common Vulnerabilities and Exposures (CVE)-2016-5267' for Mozilla Firefox. A vulnerability is defined as a weakness in a software system, system security procedures, internal controls, or implementation that could be exploited by a malicious user [8].



Bug 1284372 (CVE-2016-5267) Opened 4 years ago Closed 4 years ago

Firefox Mobile Address Bar Spoofing

User Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36
Firefox for Android

Steps to reproduce:
- Visit - https://jsfiddle.net/ktfegrhf/3/ using latest version of Firefox Mobile
- Click on the link and observe the URL

Actual results:
Due to mishandling of arabic RTL characters firefox for mobile causes the complete URL to be displayed from Right to left instead of left to right.

Expected results:
The URL should be been displayed from LTR:

Expected Result:
http://عربي.امارات/google.com/test/test/test

Rendered Results
google.com/test/test/test/عربي.امارات

FIGURE 1: Annotation of a security bug report retrieved from the Mozilla organization.

B. BACKGROUND ON SYSTEMATIC MAPPING STUDY

Systematic mapping is a technique that is used in medical research and recently in SE [12]–[15]. An SMS offers a 'map' of the research fields by classifying papers on the basis of the relevant categories and counting the work in each of those categories. An SMS offers a summary of the research domain to support researchers to identify topics that are well studied and to identify gaps that need further analysis. SMSs use the same basic approach as the systematic literature reviews (SLRs) but provide a framework for future research on a wide domain of SE rather than finding answers to a particular research issue [16].

III. RELATED WORK

Our SMS is related to prior research studies that have conducted SMSs in SE:

[8]https://csrc.nist.gov/glossary/term/vulnerability

A. PRIOR RESEARCH ON SYSTEMATIC MAPPING STUDY

The use of SMSs is commonplace in SE. Rahman et al. [14] studied 32 publications related to infrastructure as code (IaC) [13] and observed that research in IaC was mostly focused on implementing or extending the practice of IaC. Morrison et al. [17] performed a mapping study on 71 papers and reported 324 unique software life cycle security metrics. For each metric, they also identified the subject being measured, how the metric had been validated, and how the metric was used. Venson et al. [18] collected 54 papers related to secure software development. They categorized the papers according to the approach to software security cost analysis and observed that performing security reviews, applying threat modeling, and performing security testing were the three most frequent activities related to cost. Mohammed et al. [12] conducted an SMS to identify the primary studies on the use of software security techniques in the software development lifecycle. They selected and categorized 118 primary studies and showed that the most frequent used approaches were static analysis and dynamic analysis that provide security checks in the coding phase.

Zein et al. [15] conducted an SMS to structure the research evidence that has been published in the field of mobile application testing techniques and the challenges reported. They mapped 79 empirical studies to a classification schema and suggested that there is a need for eliciting testing requirements early during the development process and for conducting research in real-world development environments. Curcio et al. [19] conducted a mapping study on requirements engineering in agile software development by analyzing 104 related publications. They identified obstacles related to the environment, people, and resources, that practitioners face while dealing with requirements engineering in an agile context.

Alshuqayran et al. [20] conducted a mapping study with 33 publications and identified potential research gaps in the domain of microservices. Pahl and Jamshidi [21] performed a systematic mapping analysis of 21 publications related to microservices and reported that microservices are placed within a continuous development context, but also related to containers. Alharby and Moorsel [22] did a mapping study on blockchain-based smart contracts using 24 papers and reported that about two-thirds of the papers focus on identifying and tackling smart contract issues. Macrinici et al. [23] compiled 64 publications related to smart contracts and reported that the most discussed issues and solutions in those literature relate to the security, privacy, and scalability of blockchain and the programmability of smart contracts.

We observe the lack of SMSs in the domain of security bug reports. To the best of our knowledge, this is the first systematic mapping review to quantitatively categorize multiple aspects of the experiments recorded in several articles related to security bug reports. Through our SMS, we aim to identify research domains in the field of software vulnerability research that needs attention.

IV. RESEARCH METHOD

We conduct an SMS with publications that use security bug reports for software vulnerability research. We follow the recommendations of Felderer et al. [24] to conduct our SMS. Our SMS process consists of the following four steps: (i) publication searching, (ii) publication selection for mapping study, (iii) publication quality assessment, and (iv) data analysis and results reporting.

A. PUBLICATION SEARCHING

The purpose of the search process is to identify the publications needed for our SMS. We use an automated search mechanism where we use scholar databases to identify relevant studies for inclusion in the SMS. We use five scholar databases: (i) IEEE Xplore, (ii) ACM Digital Library, (iii) ScienceDirect, (iv) Wiley Online Library, and (v) Springer Link. We select these databases as these databases are recommended for SMS execution on SE domain by Kuhrmann et al. [25]. A literature search should address the most common sources. Instead of searching specific conference proceedings, search queries should be designed to work with entire digital libraries. For software engineering, our selected libraries are considered as standard databases [25].

Search Strings Set: We use a set of search strings for searching the selected databases. We start with an exploratory search in Google Scholar using the string ‘bug report’. Based on the search result, we observe the string ‘bug report’ yields publications that are not related to security. Therefore, to identify publications that are actually related to software vulnerabilities, we add ‘security’, and the search string becomes ‘security bug report’. From manual inspection, we observe all publications identified from the search string ‘security bug report’ to belong to software security.

We want to identify publications related to software vulnerabilities. So, we add the search string ‘vulnerability’ to our set. However, ‘vulnerability’ is a concept used in domains that are not related to SE, such as in the domain of psychology [26] and social science [27]. To identify publications that study vulnerabilities in software projects we add the string ‘software engineering’, using which we derive the search string ‘vulnerability’ AND ‘software engineering’. Finally, we have the following two search strings in our final set of search string.

- ‘security bug report’
- ‘vulnerability’ AND ‘software engineering’

We search each of our five scholar databases using the two search strings which result in a collection of publications.

Quasi-Gold Set: To assess if our search strings can yield publications that use security bug reports, we apply the quasi-gold technique proposed by Zhang et al. [28]. The quasi-gold approach validates whether our set of search strings is adequate to identify all the publications that use security bug reports for software vulnerability research. To build the quasi-gold set, we perform the following steps.

- *First*, we identify publications that use security bug reports and have at least 300 citations. Our assumption is that by selecting publications that are related to security bug reports and have high citations will enable us to build a quasi-gold set, as highly cited papers are more likely to be used by other researchers [29].
- *Second*, we apply the snowballing technique recommended by Wohlin et al. [30] to build our quasi-gold set. We apply forward and backward snowballing on the publications selected in the previous step. For forward snowballing, we identify publications that cite the selected publications. For backward snowballing, we identify the publications that are cited by the selected publications. Our snowballing process is completed when we do not identify any publications that use security bug reports of vulnerability research and are not included in the quasi-gold set.
- *Third*, we exclude publications that are not related to software vulnerabilities. We exclude unrelated publications by reading the titles of the collected publications. If we are unable to decide from the title, we read the full publication.
- *Fourth*, we included publications that are peer-reviewed and in English.

We use two raters to mitigate the subjectivity of our selection process. The first and last authors separately conduct this step. We calculate the agreement rate between the two authors using Cohen's Kappa [31], and interpret Cohen's Kappa using Landis and Koch's interpretation [32]. Then, the first and the last author resolve the disagreements upon discussion.

If the set of papers returned by our search string include papers in the quasi-gold set, then we can state that the two search strings are capable of identifying publications needed for our SMS. We use the quasi-sensitivity metric (QSM) using Equation 1 to quantify how many papers identified by the search string are included in the quasi-gold set.

$$QSM = \frac{\text{\# of publications from search strings and in quasi-gold set}}{\text{\# of publications in quasi-gold set}} \tag{1}$$

B. PUBLICATION SELECTION FOR MAPPING STUDY

The publication selection process involves defining the selection criteria, performing the selection process, and determining the relationship between paper and studies [24]. The collection of publications obtained from the five databases may contain irrelevant results that are out of scope for our research study. We use explicit exclusion and inclusion criteria that we describe below:

Exclusion criteria: As part of the exclusion process, (i) we exclude publications that are not peer-reviewed, for example, keynote abstracts and book chapters, (ii) we exclude publications that are not written in English, (iii) we exclude publications that are published before the year 2000, (iv) we exclude publications that are duplicates of other publications, and (v) we exclude publications that are not available for download.

Inclusion criteria: The criterion for inclusion is whether the title, abstract, keyword, and introduction content of a publication explicitly states that the publication uses security bug reports and is related to software vulnerability research. For the inclusion process, *first*, we only read the title to identify if a publication is relevant for us. *Next*, we read the abstract and introduction of the paper to determine their relevance. We use two raters to conduct this step. The first and second authors perform the inclusion process as independent raters. We calculate the agreement rate between the two raters. For each disagreement, first, both raters read the full paper and then upon discussion, we resolve the disagreements.

Upon applying the inclusion and exclusion criteria, we obtain a set of publications related to security bug reports.

C. PUBLICATION QUALITY ASSESSMENT

After deriving our set of publications, we manually read each publication to assess the quality of these publications. We use nine criteria proposed by Kitchenham et al. [9] for performing our quality assessment. A higher quality score means that the publication has concrete goals, actionable results, discussion on the limitations, and a clear presentation structure. The quality criteria provided by Kitchenham et al. [9] is listed below:

- **Q1** (Aim): Do the authors clearly state the goals of the research?
- **Q2** (Units): Do the authors describe the sample and experimental units?
- **Q3** (Design): Do the authors describe the design of the experiment?
- **Q4** (Data Collection): Do the authors describe the data collection procedures and define the measures?
- **Q5** (Data Analysis): Do the authors define the data analysis procedures?
- **Q6** (Bias): Do the authors discuss potential experimenter bias?
- **Q7** (Limitations): Do the authors discuss the limitations of the study?
- **Q8** (Clarity): Do the authors state the findings clearly?
- **Q9** (Usefulness): Is there evidence that the Experiment/ Quasi - Experiment can be used by other researchers/ practitioners?

Based on the response to each of the above questions, a rater provides a score: 0 (not at all), 1 (very little), 2(somewhat), 3(mostly), and 4 (fully). A higher score means that the authors of the publication have provided a thorough description for replications [9]. Since the process is susceptible to subjectivity, we use two raters who independently scores each publication. We report the average quality, which is the average score for each question and for each publication. Upon completion of this phase, we obtain a quality assessment for the selected publications that we use to answer our RQs.

Validity reported in the publications: The validity of a publication denotes the reliability of the findings, as well as the degree to which the findings are accurate and not biased by the subjectivity of the researcher's point of view [33]. When conducting research studies, there may be validity threats that either need to be accounted for or considered as possible limitations. Explicit reporting of threats or limitations is indicative of a high-quality academic publication [9]. For each of the selected publications, we check if the following four types of validity have been reported by the authors.

- **Conclusion Validity**: Threats to the conclusion validity are concerned with issues that influence the ability to draw a correct conclusion on the relationship between the treatment and the outcome of an experiment [30].
- **Internal Validity**: Threats to internal validity are factors that may impact the conclusion about a potential causal relationship between treatment and outcome [30].
- **Construct Validity**: Construct validity are limitations that relate to the generalization of the findings of the experiment to the concept or theory behind the experiment of the study [30].
- **External Validity**: Threats to external validity are factors that hinder our ability to generalize the findings of our experiment in other contexts [30].

The above-mentioned validity types are obtained from Wohlin et al. [30]. We do not make any assumptions on the threats in research that have not been documented by the authors.

D. DATA ANALYSIS ANS RESULTS REPORTING

In this section we provide the methodology to answer the following research questions:

1) Answer to RQ1: What research topics have been investigated in publications that use security bug reports for software vulnerability research?

We use a qualitative analysis called open coding [10] to derive topics that are studied in each publication. Open coding is a qualitative analysis technique that summarizes the underlying theme from unstructured text [10]. We conduct open coding with sentences collected from each publication. From the sentences, we identify initial codes. From these initial codes we derive codes, which are merged into topics based upon commonalities observed in the codes.

Figure 2 provides a hypothetical example of our open coding process. We first analyze raw text from the collected publications. Next, we extract text snippets as initial codes that describe the main topic of the publication. For example, from the raw text in the top left corner, we separate the text snippet 'the more SQL hotspots a file contains per line of code, the higher the probability that file will contain any type of vulnerability', as it describes the action of classifying vulnerabilities based on the number of SQL hotspots a file contains per line of code. Next, from the text snippet we generate an initial category called 'Prioritization heuristic for vulnerability classification'. Two initial categories 'Prioritization heuristic for vulnerability classification' and 'Predictive functionality for vulnerability classification' are combined into one category 'Vulnerability Classification', as they both indicate vulnerability classification research using security bug reports.

The process of generating topics is subjective, which we mitigate using two raters. The first and second authors individually conduct the process on the selected publication and independently generate the topics. Two topic names that are synonyms is counted as an agreement. The disagreements are resolved upon discussion. We record the agreement rate between the first and second authors. We record Cohen's Kappa [31], and interpret Cohen's Kappa using Landis and Koch's interpretation [32].

Answer to RQ1 provides us a list of topics that are studied in publications that use security bug reports for software vulnerability research. Each of our collected publications can relate to one or more of the identified topics.

2) Answer to RQ2: How frequently do identified research topics appear in publications that use security bug reports for software vulnerability research?

To answer RQ2, we use two approaches to quantify temporal trends: (i) an overall trend, and (ii) temporal trends per topic.

First, we compute the overall trend of publications that use security bug reports for software vulnerability research by counting how many publications have been published per year since 2000. Second, we compute the temporal trends exhibited for each identified topic by counting the number of publications that belong to each topic and are published each year.

3) Answer to RQ3: Which properties of security bug reports are used in publications for software vulnerability research?

This RQ can help identify which properties of bug reports have been used in software vulnerability research. We manually examine each publication to identify what properties of the bug reports were used in the paper. We use open coding similar to RQ1. We extract sentences from the publications to apply open coding and identify techniques to conduct the proposed research.

4) Answer to RQ4: What automated techniques are used in publications that use security bug reports for software vulnerability research?

Similar to RQ3, we use open coding to find the automated techniques used in publications that use security bug reports for software vulnerability research. Similarly to RQ3, we extract sentences from the publications to apply open coding and identify techniques to conduct the proposed research. Examples of techniques that answer to RQ4 could yield are natural language processing and statistical learners.

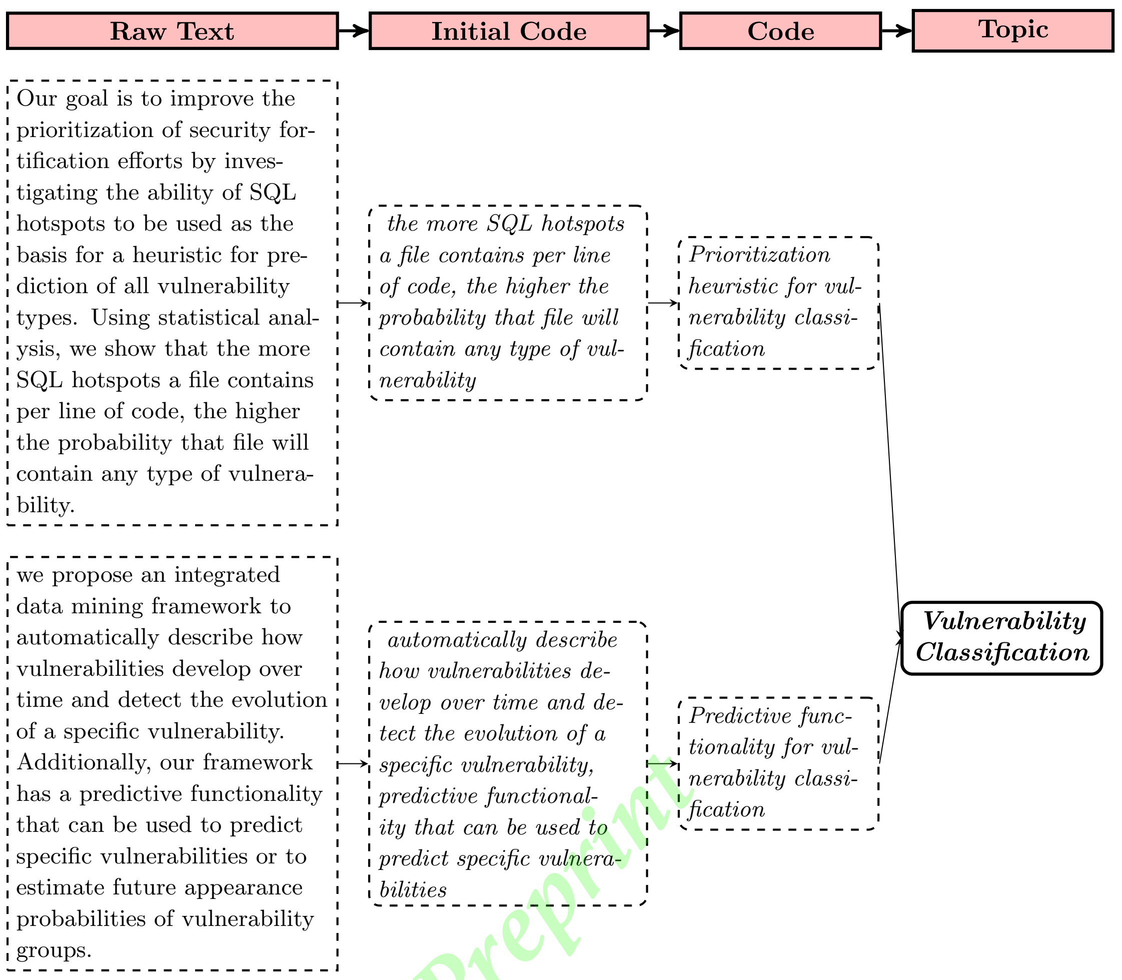

Raw Text
Initial Code
Code
Topic
Our goal is to improve the prioritization of security fortification efforts by investigating the ability of SQL hotspots to be used as the basis for a heuristic for prediction of all vulnerability types. Using statistical analysis, we show that the more SQL hotspots a file contains per line of code, the higher the probability that file will contain any type of vulnerability.
the more SQL hotspots a file contains per line of code, the higher the probability that file will contain any type of vulnerability
Prioritization heuristic for vulnerability classification
we propose an integrated data mining framework to automatically describe how vulnerabilities develop over time and detect the evolution of a specific vulnerability. Additionally, our framework has a predictive functionality that can be used to predict specific vulnerabilities or to estimate future appearance probabilities of vulnerability groups.
automatically describe how vulnerabilities develop over time and detect the evolution of a specific vulnerability, predictive functionality that can be used to predict specific vulnerabilities
Predictive functionality for vulnerability classification
Vulnerability Classification

FIGURE 2: An example to demonstrate the methodology of generating topics from publications related to security bug reports using qualitative analysis.

V. RESULTS

In this section, we describe the results of our SMS.

We search five scholar databases using our derived two search strings. As shown in Figure 3a, altogether, we obtain 45,077 publications as our search result. In Table 1, we report the number of publications for each database. We collect the titles of all the 45,077 publications as comma-separated value (CSV) files on August 2020.

TABLE 1: Number of Publications from the Search Results

Scholar Database	Count
IEEE Xplore	2,435
The ACM Digital Library	12,842
ScienceDirect	12,164
Wiley Online Library	8,376
Springer Link	9,260

Quasi-Gold Set: We identify 10 publications that belong to our quasi-gold set. Initially, the first author identifies 11 and the last author identifies 9 publications. Among these, 7 publications are common between the two authors' lists, which are added to our quasi-gold set. We calculate the agreement rate between the two authors. The recorded Cohen's Kappa is 0.34, which is 'fair' agreement according to Landis and Koch [32]. The first and the last author resolve their disagreements by discussing the contents of the publications for which both raters disagreed. Upon discussion, 3 more publications are added to the quasi-gold set, which results in a total of 10 publications in our final quasi-gold set. The list of publications included in our quasi-gold set is shown in Table A1 of the Appendix. Using Equation 1 reported in section IV-A, we record a QSM score of 1.0 for the collected publications, which indicates that our search strings are capable of identifying publications that use security bug reports for software vulnerability research.

We describe the details of our selection process as follows. As shown in Figure 3a, first we exclude 33,710 non-peer reviewed publications, which left us with 11,367 publications. During our selection process we observe the scholar databases to yield publication titles that are not peer-reviewed, such as keynote abstracts, call for papers, presenta-

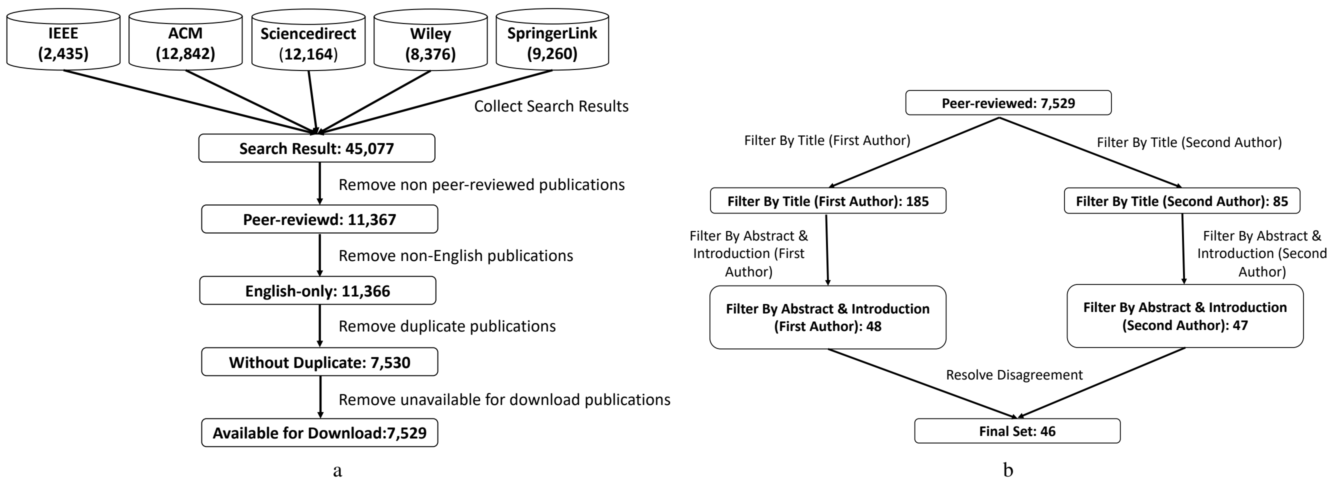

IEEE (2,435)
ACM (12,842)
Sciencedirect (12,164)
Wiley (8,376)
SpringerLink (9,260)
Collect Search Results
Search Result: 45,077
Remove non peer-reviewed publications
Peer-reviewd: 11,367
Remove non-English publications
English-only: 11,366
Remove duplicate publications
Without Duplicate: 7,530
Remove unavailable for download publications
Available for Download:7,529
a
Peer-reviewed: 7,529
Filter By Title (First Author)
Filter By Title (Second Author)
Filter By Title (First Author): 185
Filter By Title (Second Author): 85
Filter By Abstract & Introduction (First Author)
Filter By Abstract & Introduction (Second Author)
Filter By Abstract & Introduction (First Author): 48
Filter By Abstract & Introduction (Second Author): 47
Resolve Disagreement
Final Set: 46
b

FIGURE 3: Our process of obtaining the final set of 46 publications. Figures 3a summarizes the process of obtaining 7,529 publications from the initial 45,077 search results collected from five scholar databases. Figure 3b summarizes the process of obtaining the final set of 46 publications from the set of 7,529 publications collected.

tions, newsletters, and books. Next by reading the title of the 11,367 publications, we identify 1 publication that is written in non-English and separate out 11,366 publications all written in English. Then, from the set of 11,366 publications, we remove duplicates and separate out 7,530 publications. Among these 7,530 papers, 1 paper is not available for download. All the other 7,529 publications are accessible and available for download. The results of each of the exclusion criterion are shown in Table 2.

TABLE 2: Publication Selection Using the Exclusion Criteria

Exclusion Criterion	Count
Search Result	45,077
Peer-reviewed	11,367
English-only	11,366
Without Duplicate	7,530
Available for Download	7,529

From the collected set of 7,529 publications, we determine if each of the publications use security bug reports for software vulnerability research. As shown in Figure 3b, the first and second author performs as raters and individually complete this step to determine related publications. Both raters individually read the title of all the 7,529 publications and excluded 7,344 and 7,444 publications respectively, which are irrelevant to security bug reports.

Application of the exclusion criteria results in 185 and 85 publications respectively for the two raters. Next by reading the abstract and introduction of each of the remaining publications, the two raters respectively identify 48 and 47 publications. The two raters together select a total of 53 unique papers, 42 of which were in common between the two raters. The agreement rate between the two raters is 79%.

While determining relevant publications we record disagreements between the two raters. For example, the first rater marks the publication 'Research on the architecture of vulnerability discovery technology' [34] as relevant, but the second rater finds this publication to be irrelevant. The second rater marks the publication 'At the Edges: Vulnerability Prediction and Resilience' [35] as relevant, but the first rater observes this publication as irrelevant. For the disagreements, the two raters read the full papers, and then resolve the disagreements by discussing the contents of the papers. The results of each of the exclusion criteria are shown in Table 3.

TABLE 3: Publication Selection Using the Inclusion Criteria

Inclusion Criterion	Count	
	First Author	Second Author
Peer-reviewed	7,529	
Filter By Title	185	85
Filter by Abstract & Introduction	48	47
Final Set	46	

Upon resolving all the disagreements, we obtain a set of 46 publications that use security bug reports for software vulnerability research. Each of the names of the publications is listed in Table A2 of the Appendix. We index each publication as 'P#', for example, the index 'P1' refers to the publication 'A Vulnerability Analysis and Prediction Framework'.

Quality Analysis of Publications in Our Set: After selecting 46 relevant publications, we evaluate each of the publication's quality using the criteria provided by Kitchenham et al. [9]. Each rater independently provides scores for each of the nine quality criteria described in Section IV-C. We report our findings as the average score for each criterion and for each publication in Table 4. The title of each criterion is also specified in Table 4 alongside each quality criterion. For example, the quality criterion Q1 relates to the criterion of a publication's aim or goal being clearly stated and publication P1 has a quality score of 2.5 for this quality criteria Q1. In the last row 'Avg.', we report the average score of all publications for each quality criterion. For example, for the

TABLE 4: Quality Assessments of Each Publications

Index	**Q1** Aim	**Q2** Units	**Q3** Design	**Q4** Data Collection	**Q5** Data Analysis	**Q6** Bias	**Q7** Limitations	**Q8** Clarity	**Q9** Usefulness	**Avg.**
P1	2.5	4	4	4	4	0	1.5	4	4	3.11
P2	2.5	4	4	4	4	4	4	4	4	3.83
P3	4	4	4	4	2.5	4	4	4	4	3.83
P4	4	4	4	4	3.5	4	4	4	4	3.94
P5	4	4	4	4	4	4	4	4	4	4
P6	4	4	4	4	4	4	4	4	4	4
P7	4	4	4	4	4	4	4	4	4	4
P8	1.5	4	4	4	4	0	1.5	4	4	3
P9	3	2.5	4	3.5	3.5	4	4	4	4	3.61
P10	4	3.5	4	4	3.5	4	4	4	3.5	3.83
P11	3	0	4	2.5	3	0	0	1	1	1.61
P12	3	3.5	4	3.5	4	0	0	4	4	2.89
P13	4	4	3.5	2.5	3	1.5	3.5	4	3.5	3.28
P14	4	4	4	4	4	3.5	4	4	3.5	3.89
P15	4	2.5	3.5	3.5	2.5	0.5	0.5	4	3.5	2.72
P16	4	3.5	4	4	3.5	0	0	4	3.5	2.94
P17	4	3	2	4	3	0	0	2.5	3	2.39
P18	4	2.5	4	3.5	3	2	0	3.5	3	2.83
P19	4	3	4	4	4	0	0	4	3	2.89
P20	4	4	3.5	2	3	4	4	4	2.5	3.44
P21	4	4	3.5	4	4	4	4	4	4	3.94
P22	4	3	3.5	4	4	2.5	3	4	4	3.56
P23	4	4	3.5	4	2	4	4	3.5	3	3.56
P24	1.5	2.5	4	4	3	1.5	2	4	4	2.94
P25	2	2	3.5	3.5	3	0	1	4	3	2.44
P26	3	4	4	4	4	4	4	4	4	3.89
P27	2	4	4	3.5	3.5	4	4	4	4	3.67
P28	2.5	4	4	3.5	1.5	0	0	4	4	2.61
P29	2	2	4	4	1.5	0	0	3.5	1.5	2.06
P30	4	4	4	4	3.5	1	3.5	4	4	3.56
P31	1.5	3.5	4	4	4	4	4	4	4	3.67
P32	1.5	4	4	4	4	0	0	4	4	2.83
P33	3.5	4	4	3.5	4	0	0	4	4	3.00
P34	4	3.5	4	2.5	4	3	4	4	4	3.67
P35	3	4	4	4	2.5	4	4	4	4	3.72
P36	2.5	3.5	3.5	4	2.5	3	4	4	2.5	3.28
P37	3	4	4	4	4	4	4	4	4	3.89
P38	3	4	3	3	3.5	0	0	4	3	2.61
P39	4	4	4	4	4	4	4	4	4	4
P40	4	0.5	3.5	1	1	0	0	4	2	1.78
P41	3	4	4	4	4	4	4	4	4	3.89
P42	4	4	4	4	4	4	4	4	4	4
P43	2.5	3.5	4	4	3	0	0	4	4	2.78
P44	4	4	4	4	4	4	4	4	4	4
P45	2.5	4	4	4	4	2.5	2.5	4	4	3.5
P46	3	4	4	4	4	4	4	4	4	3.89
Avg.	3.25	3.48	3.84	3.68	3.41	2.27	2.49	3.87	3.59	

quality criterion Q1, the average score of all 46 publications is 3.25. In the last column 'Avg.', we report the average score of all quality criteria for each publication. For example, for the publication P1, the average score of all 9 quality criteria is 3.11.

Except for the Q6 and Q7 quality criteria, all the other criteria have an average score of more than 3, which indicates that our selected publications satisfy most of the criteria including, expressing the aim of the publication, describing the design of the experiment, explaining the data collection, and analysis procedure and stating the findings.

The two quality assurance criteria Q6 and Q7 with an average score of less than 3 correspond to the discussion of potential experimenter bias and the discussion of the limitations of the study. According to Kitchenham et al.'s guidelines [9] [36], research publications should disclose the potential bias of the experimenter and the threats associated with the research publication. Future research studies can take our findings into account while conducting research that uses security bug reports, and report possible biases and limitations that could arise while conducting their research studies.

In our set of 46, we observe 30 publications to explicitly report the limitations. The index of the 30 publications is listed in Table 5. In each cell, we report whether or not a specific category of threats is reported in the corresponding publications. For example, we observe that the publication P1 only mentioned the internal validity of their studies, but did not explicitly state the conclusion, construct, or external validity. The findings of Table 5 indicate that research studies that use security bug reports for software vulnerability research do not sufficiently report the threats of their research work. We advocate for better reporting of research threats in studies that use security bug reports following the guidelines of Wohlin et al. [30].

We observe a wide range of techniques to be used to detect security bug reports, such as the use of existing labels available in bug reports, application of natural language processing, and manual labeling using raters. P2, P3, P7, P12, P37, and P39 used available labeled bug reports, where each bug report is labeled as security bug report (SBR) or non-security bug report (NSBR). P19 [37] leveraged information that are usually available in a bug report, including meta features and textual features, to automatically identify the security bug reports via natural language processing and machine learning techniques. P5 [38] used different strategies for labeling positive samples (i.e., SBRs) and negative samples (i.e., NSBRs). The positive samples are labeled according to CVE entries: if a record of source bug reports is related to any CVE entries (i.e., is linked to CVE entry), they labeled that record as a positive sample (i.e., SBR). If a source bug report is related to a CVE entry, the CVE identifier would be included in the "Title" or "Reference" of the source bug report. They selected 50 negative samples (i.e., NSBRs) via strategy like "card sorting" [39], which is a widely applied approach for generating categories.

Publications P22, P24, and P35 used a semi-automated process to detect security bug reports. They used data-mining techniques to identify software vulnerabilities, then they classify them into different categories by using the Bugs Framework proposed by the National Institute of Standards and Technology (NIST). P24 [40] first manually labeled part of their dataset, then used both supervised and unsupervised approaches for the classification of software bug reports to security and non-security related. P4 [41] used a manual process using two raters. Authors of P4 [41] independently classified each bug report as a vulnerability or as not a vulnerability. They used a list of keywords determined by Shin et al. [42] for the initial mining. Then for each bug, they looked for a "dif", or record of code changes, attached to the bug that had been positively reviewed by a security team member (indicated by the sec-approval+ tag) or a release manager (indicated by approval-X or review+ tags). If all the diffs had reviews from a security team member or a release manager, they included them in their vulnerability list.

P10, P13, P23 also used a manual process to extract security bugs from several bug reports of software bug repositories. P10 [43] performed the manual vulnerability curation for over a year from 2017, employing a team of security researchers. Authors of P10 [43] performed keyword-based filtering with security-related keywords, including keywords such as "security", "advisory", "authorized", "NVD", etc. P13 [44] consider a component as vulnerable if it was modified in order to fix the vulnerability. P23 [45] conducted manual classification of software bug reports using the information provided in the 'Title', 'Subject', 'Description', 'Recommended Actions', and 'Solution' fields from the issue tracking systems.

P16 [46] used a team of professional security researchers who manually investigated the collected data. The security researchers checked every single commit and bug report. To ensure the accuracy of results, for an entry (a commit or bug report) that is related to a vulnerability, the security researchers conduct two rounds of analysis on it. In the first round, one security researcher would first check if the vulnerability is published publicly in National Vulnerability Database (NVD), then analyze the exploitation process, CVSS score, vulnerable code, affected versions, and document it in a vulnerability report. In the second round, another security researcher will verify the vulnerability report by examining all the details. In addition, all disputed reports are set aside for team discussion before a final decision. P9 [47] downloaded vulnerability dataset from the NIST National Vulnerability Database and categorized them by security experts employed at their industrial partner.

P5, P10 and P37 used list of keywords to identify security bug reports. P5 listed the following keywords related to security bug reports: "leak", "leakage", "memory", "attack", "security", "https", "password", "risk", "javascript", "access", "ssl", "vulnerability", "vulnerabilities", "attacker", "directory", "token", "PKI", "port", "scan", "bypass", "authorization", "admin". P10 performed a keyword-based filtering with security-related keywords, including keywords such as, "security", "advisory", "authorized", "NVD", etc. P37 [48] initially used historical bug reports, which are labeled either as a security bug report or not. Then they automatically identified security related keywords from the security bug reports of a project using the term frequency-inverse document frequency (TF-IDF) technique. Each security related keyword is scored according to its frequency in both security and non-security reports. Using the keyword scores of bug reports, they removed non-security reports with scores as high as those of security bug reports. The remaining bug reports are used to build the vulnerability classification model. They reported a list of 500 security related keywords. Example keywords include: "file", "security", "chrome", "page", "http", "download", "user", "starred", "person", "notified". They reported that their identified keywords are similar to those found in other studies [49] [50] [51]. P35 [52] automatically constructed an up-to-date security-relevant keyword list from two sources: labeled data and the Common Vulnerabilities and Exposures (CVE) database.

A. ANSWER TO RQ1: WHAT RESEARCH TOPICS HAVE BEEN INVESTIGATED IN PUBLICATIONS THAT USE SECURITY BUG REPORTS FOR SOFTWARE VULNERABILITY RESEARCH?

The first and second author individually identify five topics. Four topics are common between both raters, which are (i) classifying absence and presence of vulnerabilities, (ii) vulnerability category classification, (iii) vulnerability report summarization, and (iv) vulnerability dataset construc-

TABLE 5: Reported Threats for Each Publication

Index	Conclusion	Construct	External	Internal
P1	N	N	N	Y
P2	N	N	Y	Y
P3	Y	Y	Y	N
P4	N	Y	Y	Y
P5	N	N	Y	Y
P6	N	N	Y	Y
P7	N	N	Y	Y
P9	N	N	Y	Y
P10	Y	N	Y	Y
P13	N	N	N	Y
P14	Y	Y	Y	Y
P20	Y	Y	N	Y
P21	N	Y	Y	Y
P22	N	N	Y	N
P23	Y	Y	Y	Y
P25	Y	Y	Y	Y
P26	Y	Y	Y	Y
P27	N	N	Y	N
P30	N	Y	N	N
P31	Y	Y	Y	Y
P34	N	N	Y	N
P35	N	N	Y	Y
P36	N	Y	Y	Y
P37	N	Y	Y	Y
P39	N	N	Y	Y
P41	Y	Y	Y	Y
P42	Y	Y	Y	Y
P44	N	Y	Y	Y
P45	N	Y	N	N
P46	Y	Y	Y	Y

tion.

We calculate the agreement rate between the two authors. The recorded Cohen's Kappa is 0.56, which is 'moderate' agreement according to Landis and Koch [32]. Upon discussion, we resolve the disagreements and identify four topics. Later we merge two identified topics 'classifying absence and presence of vulnerabilities' and 'vulnerability category classification' into one topic called 'vulnerability classification', as both of these topics are related to classification, one being related to classifying vulnerability categories, such as classifying if a vulnerability is an injection vulnerability or an overflow vulnerability, and the other identified topic is related with classifying absence or presence of vulnerabilities. The final identified three topics are:

(i) vulnerability classification, (ii) vulnerability report summarization, and (iii) vulnerability dataset construction.

A complete mapping of each of the three topics and the publications that discuss the topic is available in Table 6. A publication can belong to multiple topics. For example, P14 [53] belongs to both vulnerability classification and vulnerability dataset construction because they performed their research on both dataset construction as well as vulnerability classification. We describe each topic below:

- **Vulnerability Classification**: This topic includes publications that investigate techniques on how to classify the presence of vulnerabilities and vulnerability categories. We identified two sub-topics that are described below:

  Absence/Presence of Vulnerabilities: Detecting the absence or presence of vulnerabilities is a research topic that has gained interest amongst researchers. This topic refers to a binary classification problem where a file or a software component is classified as vulnerable or not [54]. Publications related to this topic construct models that classify whether or not software artifacts, such as a class, a file, or a binary is likely to include a vulnerability or not. The motivation for conducting this research is to prioritize inspection and testing efforts so that vulnerabilities can be mitigated early. We provide a brief summary of prior work that belongs to this category as follows:

  -- In P1 [6], researchers classify software artifacts with vulnerabilities by finding a correlation between vulnerabilities and the evolution of the vulnerabilities from its genesis. Based on the evolution of a specific vulnerability, P1 [6] also predicts future appearance probabilities of vulnerability groups. P3 [54] used a fault prediction model to build a vulnerability classification model using metrics such as complexity, code churn, and fault history. P6 [42] investigated whether software metrics obtained early in the software development life cycle are discriminative of vulnerable code locations and whether the metrics are predictive of vulnerabilities. Their study was performed using metrics that can be obtained from the development process as well as from source code. They proposed a vulnerability classification model using complexity, code churn, and developer activity metrics to predict vulnerable code locations.

  -- We notice publications using natural language processing (NLP) and text mining for vulnerability classification (P4, P16, P27, P31, P37, P45). P4 [41] compared the performance of multiple vulnerability classification models to find a different combination of features for a better prediction model. P19 [37] also presented an automated security bug report identification model via multi-type features analysis. From security bug reports they mined meta-features and textual features, to automatically identify the security bug reports via NLP and ML techniques. P16 [46] explores the identification of vulnerabilities in commit messages and issue reports/pull requests using NLP and ML techniques. Their k-fold stacking ensemble approach discovers hidden vulnerabilities in 5,002 projects spanning over 6 programming languages.

  -- P8 [55] not only predicts the presence of vulnerabilities but also predicts the number of latent vulnerabilities that can potentially be present in a software system but may not have been found yet. They observed that the values of vulnerability densities fall within a range, and for similar products they are closer together. The researchers analyzed the Windows and Red Hat Enterprise Linux operating systems and observed that the ratio of vulnerabilities to the total number of faults falls in the range of 1% to 5%. If this ratio is constantly true across projects, the discovered vulnerability densities can be a useful indicator to estimate the efforts to find

TABLE 6: Mapping Between the Topics and Publications

Topic	Publication Index	Count
Vulnerability Classification	P1, P2, P3, P4, P6, P7, P8, P9, P10, P11, P12, P13, P14, P16, P17, P18, P19, P20, P21, P22, P23, P24, P25, P26, P27, P28, P29, P30, P31, P32, P33, P34, P35, P36, P37, P39, P40, P42, P43, P44, P45, P46	42
Vulnerability Report Summarization	P9, P15, P23, P38, P41	5
Vulnerability Dataset Construction	P5, P10, P14	3

undiscovered vulnerabilities. Their method is useful for estimating the effort required to identify and correct undiscovered security vulnerabilities.

-- P12 [56] presented a bug classification technique in the process of identifying bug reports as security or non-security. They focus on security vulnerabilities and present a bug mining system for the identification of vulnerabilities and non-security bugs using the TF-IDF technique.

-- P13 [44] used data from the National Vulnerability Database (NVD) [9] to predict the time to the next vulnerability for various software products using data-mining techniques. They experimented with various features constructed using the information available in NVD and applied various statistical learners to examine the predictive power of the data. They showed that the data in NVD generally have poor prediction capability, with the exception of a few vendors and software applications.

-- P14 [53] evaluated the effectiveness of vulnerability classification methods and mentions the challenges in vulnerability classification research, such as having a lack of reliable vulnerability dataset and lack of replication framework for comparative analysis of existing methods. P20 [57] proposed to improve vulnerability classification through parameter tuning of learners and data pre-processor. They applied hyper-parameter optimization to the control parameters of a learner. The data pre-processing methods handle cases where the target class is a small fraction of all the data. They showed that optimizing the pre-processor is more useful than optimizing the learners.

-- P7 [58] proposed a novel approach called 'LTRWES' that incorporates learning to rank and word embedding into the identification of security bug reports. LTRWES is a content-based data filtering and representation framework that exploited the ranking model to efficiently filter non-security bug reports. LTRWES also applies word embedding technology to transform the text features into low-dimensional real-value vectors.

-- P32 [59] introduced a tool called 'Vulture' to automatically map vulnerability reports to vulnerable components in the Mozilla codebase. Using the tool, they observed common patterns of imports (#include) and function calls in vulnerable components using pattern mining techniques. Vulture identified 45% of all vulnerable components (recall) using import analysis, and 70% of all identified vulnerable components were true vulnerable components (precision) using function call analysis. Vulture was also able to identify 82% of actual vulnerabilities in the top 30% of files ranked in the order of predicted vulnerabilities.

-- P37 [48] proposed a framework FARSEC that improved the prediction performance by choosing the security-related keywords automatically and filtering non-security bug reports. FARSEC selected the top 100 terms in security bug reports with the highest TF-IDF values as security-related keywords and represented each bug report as a 100-dimensional feature vector for filtering non-security bug reports.

-- P10 [43] used semi-supervised learning to predict how likely a software artifact is susceptible to a vulnerability. The proposed technique supports a complete pipeline from data collection, model training, and prediction, to the validation of new models before deployment. It is executed iteratively to generate better models as new input data become available. The authors used self-training to significantly and automatically increase the size of the training dataset, opportunistically maximizing the improvement in the models' quality at each iteration. For the final prediction, the authors used a stacking ensemble that consists of six statistical learners: random forest (RF), naive bayes (NB), k nearest neighbor (KNN), support vector machines (SVM), gradient boosting, and AdaBoost.

-- In addition to supervised and semi-supervised learning methods, unsupervised learning methods are also applied to construct classification models. P24 [40] designed an automated classification of software bug reports to security and non-security related bugs, using both supervised and unsupervised approaches. For this purpose, they used the title, subject, and description of bug reports.

-- While P24 [40] used the title, subject, and description of bug reports, P25 [60] showed that the prediction of vulnerabilities can be performed even when solely the title is available for training and scoring.

-- P26 [61] used active learning for prediction. Their tool HARMLESS is an incremental support vector machine tool that built a vulnerability classification model from source code, then suggested what source code files should be inspected next. HARMLESS reduced the time and effort required to achieve the desired level of recall for finding vulnerabilities. The tool provided feedback on when to stop while at the same time, correcting

[9]https://nvd.nist.gov/

human errors by double-checking suspicious files.

Prediction models constructed from the above-mentioned proposed research suffer from a high false-positive rate (P1, P4, P5, P7, P13, P14, P16, P35, P42, P45). Vulnerability classification can exhibit a decrease in recall with an increase of precision for a certain range of classification thresholds as shown in P1 [6].

Classifying Vulnerability Categories: This topic includes publications that construct models to classify the categories of vulnerabilities. For example, P39 [5] classified vulnerabilities based on the predicted severity level of the vulnerabilities using ML models on historical vulnerability data.

-- P2, P28, and P39 classified vulnerabilities according to their severity levels. P2 [4] proposed a framework for vulnerability severity classification using five statistical learners, namely, RF, KNN, decision tree (DT), NB, and SVM. P28 [62] used word embeddings and a one-layer shallow convolutional neural network (CNN) to automatically capture discriminative words and sentence features of bug report descriptions. P39 [5] presented another framework for vulnerability severity classification using the Bellwether analysis (i.e., exemplary data) [63]. They applied the NLP techniques on bug report descriptions. They also developed an algorithm called 'Bellvul' to identify and select an exemplary subset of data (referred to as Bellwether) to be considered as the training set to four statistical learners, namely, deep neural network (DNN), logistic regression (LR), KNN, and RF.

-- P45 [2] and P11 [3] classified hidden impact vulnerabilities, i.e., vulnerabilities that appeared long after the vulnerability has been made public. Their methodology utilized a text mining process that extracts syntactical information of the bug reports and compresses the information for easier manipulation. The compressed information is then utilized to generate a feature vector that is presented to a classifier. They used three statistical learners: NB, Multinomial NB, and DT. P11 [3] also classified the absence and presence of hidden impact vulnerabilities using text mining features.

-- P9 [47] proposed an approach to categorize vulnerabilities according to a taxonomy based on summaries of vulnerabilities available as advisory reports. They used statistical learners, such as RF, BayesNet, Simple LR, and NB.

-- P17 [64] proposed an automated software vulnerability classification model based on DNNs. Their methodology uses the TF-IDF technique to compute the frequency and weight of each word in the textual descriptions of software vulnerability.

-- P18 [65] applied NB and artificial neural network (ANN) to classify what security objectives, namely, availability, confidentiality, and integrity are violated in a reported vulnerability. Their experimental results showed that the ANN algorithm was superior to the NB algorithm in vulnerability classification.

-- P21 [66] classified security bug reports into valid and invalid vulnerabilities. First, they used card sorting [39] to categorize invalid vulnerability reports, from which six main reasons were observed for rejected and disputed CVEs, respectively. Next, they proposed a text mining approach to predict the invalid vulnerability reports using NB, SVM, and RF.

-- By mining text features P22 [67] constructed a model to predict the weakness that is related to a reported vulnerability.

-- P23 [45] classified software vulnerabilities into software weakness types based on the Common Weakness and Enumeration (CWE) taxonomy. CWE is a community-developed database of software security weaknesses [68]. The goal of CWE is to (i) understand software security weaknesses, (ii) create tools to identify, fix, and prevent security weaknesses, and (iii) serve as a baseline for vulnerability identification and mitigation efforts [68].

- **Vulnerability Report Summarization**: This topic includes publications that focus on understanding the structure and meaning of security bug reports. Vulnerabilities are described using natural language in vulnerability databases, which makes automated summarization of vulnerability challenging. To address this challenge researchers [47] [69] have focused on mining text features from security bug reports to summarize vulnerabilities and rank vulnerabilities based on priority.
  – Supervised Learning Approach: P9 [47] proposed an approach to automatically generate summaries of reported vulnerabilities, and categorize them according to a taxonomy modeled for the industry. P15 [69] also automatically characterized software vulnerabilities from the textual description included in the CVE reports and mapped each vulnerability descriptions to the U.S. NIST Vulnerability Description Ontology [70]. Both P9 [47] and P15 [69] used statistical learners, such as NB, SVM, and RF to automatically generate summaries of reported vulnerabilities.
  – Semi-supervised Learning Approach: P38 [71] used a semi-supervised language embedding technique called 'word2vec' with paragraph vectors to capture the semantic content of bug descriptions in forms of numeric vectors.
  – NLP Approach: P41 [72] used NLP to automatically analyze security reports to extract vulnerability-specific features to construct an intrusion detection system (IDS).
  – Manual Approach: P23 [45] manually created vulnerability profiles from security bug reports where they determined when the vulnerabilities were introduced in which locations.
- **Vulnerability Dataset Construction**: This topic includes

publications that focus on the construction and quality of vulnerability datasets. Quality vulnerability datasets help to build and evaluate automated techniques in vulnerability classification research.

- For supervised or semi-supervised ML models, researchers leverage labeled security bug reports. Research work on vulnerability classification might assume that predictive models are trained with respect to perfect labelling information [73] [59] [74]. P14 [53] performed an empirical study to investigate three vulnerability classification approaches in two settings: with and without the unrealistic labeling assumption. They reported a Matthews Correlation Coefficient (MCC) [75] of 0.77, 0.65, and 0.43 for Linux, OpenSSL, and Wireshark when trained on sufficient and accurately labeled data. However, MCC is respectively 0.08, 0.22, 0.10 when considering realistic partial and mislabeled data. The unrealistic labeling assumption can mislead the scientific conclusions drawn; suggesting effective and deployable prediction results might not hold if we account for realistically available labeling in the experimental methodology.
- P5 [38] proposed an automated data labeling approach based on iterative voting classification. Their approach starts with a set of 42,940 ground-truth samples, which were labeled with the help of authoritative vulnerability records hosted in CVE. Using their proposed approach the authors labeled over 120K more bug reports using their proposed approach.
- P10 [43] used a semi-supervised learning approach to curate a library vulnerability database. The authors applied their production models to the unlabeled data to significantly increase the amount of labeled data for training new models.

TABLE 7: Comparison Among the Publications Based on Dataset Usage

Dataset	Publication Index	Topics
NVD Dataset	P1, P2, P9, P10, P13, P14, P15, P17, P21, P22, P28, P34, P43, P44	Vulnerability Classification, Vulnerability Report Summarization, Vulnerability Dataset Construction
Mozilla Firefox Dataset	P3, P4, P6, P7, P11, P12, P19, P26, P29, P39	Vulnerability Classification
Chromium	P7, P37, P39	Vulnerability Classification
Eclipse	P11	Vulnerability Classification
NASA Dataset	P23, P24	Vulnerability Classification
Microsoft Dataset	P25	Vulnerability Classification
CISCO Dataset	P27	Vulnerability Classification
Existing Dataset	P20	Vulnerability Classification

As shown in Table 7 We observe commonalities amongst publications with respect to dataset usage. We provided the datasets that are commonly used and provided a comparison in Table 7. The NVD dataset is used by 14 of the collected 46 publications for vulnerability classification, vulnerability report summarization and vulnerability dataset construction. Some of the other commonly used datasets, such as the Mozilla Firefox dataset, the Chromium dataset, the NASA dataset are used for vulnerability classification.

Answer to RQ1: We identify three topics that are investigated in publications that use security bug reports for vulnerability research: vulnerability classification, vulnerability report summarization, and vulnerability dataset construction. With respect to publication count, the most frequently occurring topic is vulnerability classification.

B. ANSWER TO RQ2: HOW FREQUENTLY DO IDENTIFIED RESEARCH TOPICS APPEAR IN PUBLICATIONS THAT USE SECURITY BUG REPORTS FOR SOFTWARE VULNERABILITY RESEARCH?

To describe the overall trend of publications that use security bug reports for software vulnerability research, we first provide the count of publications that are published each year in Table 8. Even though our search process included publications starting from 2000, our earliest publication date is the year 2007. For the year 2019, we observe the highest publication count, which is 11. As our search process included publications up to August 2020, our publication set only includes publications until the month of August.

To compute the temporal trends exhibited for each identified topic, we report the frequency of publications for each topic in Table 9. the columns 'Classification', 'Summarization', and 'Construction' respectively denotes vulnerability classification, vulnerability report summarization, and vulnerability dataset construction. We observe the number of publications for all topics is highest in 2019, which is consistent with the overall trend shown in Table 8.

TABLE 8: Frequency of Publications that Use Security Bug Reports for Software Vulnerability Research. X Denotes Absence of a Publication in Our Publication Set for that Year.

Year	Count
2020	8
2019	11
2018	4
2017	7
2016	2
2015	1
2014	4
2013	1
2012	1
2011	2
2010	3
2009	X
2008	X
2007	2

TABLE 9: Frequency of Publications per Year for Each Topic. X Denotes Absence of a Publication in Our Publication Set for that Year.

Topic	Classification	Summarization	Construction
2020	7	X	2
2019	9	3	1
2018	4	X	X
2017	6	2	X
2016	2	X	X
2015	1	X	X
2014	4	X	X
2013	1	X	X
2012	1	X	X
2011	2	X	X
2010	3	X	X
2009	X	X	X
2008	X	X	X
2007	2	X	X

Answer to RQ2: Publications related to vulnerability classification have remained prevalent from 2007 to 2020.

C. ANSWER TO RQ3: WHICH PROPERTIES OF SECURITY BUG REPORTS ARE USED IN PUBLICATIONS FOR SOFTWARE VULNERABILITY RESEARCH?

For efficient management of bugs information technology (IT) organizations use bug tracking systems, such as GitHub issues, JIRA, and Bugzilla. Bug tracking systems allow teams of practitioners to keep track of bugs, new feature requests, and enhancements in their products effectively.

Bug Tracking Systems Used in Publications: We identify the following bug tracking systems that are used in our set of 46 publications:

- **GitHub**: As of January 2020, GitHub has 40 million users and hosts 190 million software repositories [76]. GitHub provides issues that can be used to track enhancements, feature requests, and bugs [77].
- **Jira**: According to Atlassian, Jira is used by over 75,000 customers in 122 countries around the globe [78]. Jira offers utilities to track issues and tasks, such as the Jira Query Language (JQL). JQL can be used to conduct search and reporting of issues, custom dashboards, and third-party plugins that allow various features, such as duplicate bug detection [77].
- **Bugzilla**: Bugzilla is a bug tracking system developed by Mozilla. Companies, such as Wikipedia and Mozilla use Bugzilla for their software projects [79]. Bugzilla is designed to be a dedicated issue tracker with features, such as automatic duplicate detection, advanced search options, and customizable workflows [77].
  From our mapping study we observe the most commonly used bug tracking system to be Bugzilla. As shown in Table 10, we observe 18 out of the 46 publications to mine bug reports hosted on Bugzilla.

Bug Report Properties: We observe researchers to mine

TABLE 10: Bug Tracking Systems Used in Publications that Use Security Bug Reports for Software Vulnerability Research

Tool	Publication Index
GitHub	P10, P16
Jira	P5, P10, P16, P30, P33, P40
Bugzilla	P3, P5, P6, P7, P8, P10, P11, P12, P16, P19, P26, P29, P30, P32, P35, P36, P44, P45

properties from bug reports to conduct research related to vulnerability classification, vulnerability dataset construction, and vulnerability report summarization. We have listed a set of properties that are mined from security bug reports in our set of 46 publications:

- Bug Severity: Bug severity is a measure of the effect that a bug can have on the development or operation of an application feature as it is being used. Bug severity can be divided into different levels depending on how much of a threat the bug can pose to the software:
  Low: This category of bugs will not result in any noticeable breakdown of the system.
  Minor: This category of bugs will result in some unexpected or undesired behavior, but not enough to disrupt system function.
  Major: This category of bugs are capable of collapsing large parts of the system.
  Critical: This category of bugs are capable of triggering a complete system shutdown.
- Common Vulnerability Scoring System (CVSS) score: CVSS is an open framework for the communication of the characteristics and severity of software vulnerabilities. The NVD provides CVSS scores for reported vulnerabilities.
- Code Complexity: Complex code is more difficult to understand and evaluate, so developers may have a greater risk of introducing latent vulnerabilities. Shin et al. [42] observed that complexity metrics, such as McCabe's complexity can indicate security vulnerabilities.
- Code Churn: Code is continuously changing in the development process and any new change in the system introduces the possibility of introducing vulnerabilities [42]. Examples of code churn metrics are the total number of new code lines and code lines changed since the creation of a file. Certain bug reports have entries that show code churn of a file linked with the bug report of interest.
- Developer Activity: Software development is typically carried out by development teams working together on a shared project. Lack of team cohesion or miscommunications can correlate with software artifacts that include vulnerabilities [80] [42]. An example developer activity that correlates with software vulnerabilities is multiple developers modifying the software artifact of interest. From a version control data can be mined to extract metrics using social network analysis [81].

– Free Form Entries: This group of bug report entities allows practitioners to provide information about a specific bug in terms of comments and descriptions. Typical entities of bug reports that are free form entries are: (i) a title that summarizes the bug, (ii) descriptions that describe the bugs, and (iii) include comments that discuss how to reproduce and resolve the bug of interest. Researchers apply text mining on free form entries of bug reports. We call these bug report entries 'free form' as there is no structure on how to specify text input. As shown in Table 11, 14 publications out of our set of 46 mine free form entries from bug reports to conduct research investigations.

TABLE 11: Features Used in Publications that Use Security Bug Reports for Software Vulnerability Research

Features	Publication Index
Bug Severity	P2, P10, P19
CVSS Score	P13, P17, P18, P28, P38, P39
Code Complexity	P3, P4, P6, P9, P14, P34, P42, P44
Code Churn	P3, P4, P6, P9, P14, P27, P30, P31, P34, P40, P42, P44, P46
Developer Activity	P6, P9, P31, P34
Free Form Entries	P2, P4, P7, P9, P10, P12, P15, P16, P19, P21, P24, P25, P28, P3

Answer to RQ3: Bugzilla is the most frequently used bug tracking system amongst our studied publications. Researchers mine properties from bug reports, such as free form entries, CVSS scores, and bug report severity to conduct research investigations.

D. ANSWER TO RQ4: WHAT AUTOMATED TECHNIQUES ARE USED IN PUBLICATIONS THAT USE SECURITY BUG REPORTS FOR SOFTWARE VULNERABILITY RESEARCH?

We organize this section in two subsections: (i) *first*, we describe the techniques used in our set of 46 publications, and (ii) *second*, we describe how publications that belong to the three identified topics use the identified techniques.

1) Techniques Used To Conduct Software Vulnerability Research

- **Text Mining:** NLP is the domain of computer science that derives, evaluates, and applies algorithms that enable program computers to process and analyze natural language data. [82]. Since security bug reports are expressed using natural language, NLP techniques such as tokenization, summarization, and word-embedding are used in our set of 46 publications, which we describe below:
  – TF-IDF: Security bug reports are described using natural languages, which means the text data need to be cleaned and convert to a numerical representation to be used by statistical learners. TF-IDF can be used to convert the text data into numerical representations [83]. TF-IDF evaluates the importance of a term to a text in the collection of documents.
  – Word Embedding: Word embedding is a word representation technique that allows words with similar meaning to have a similar representation. One technique is 'Word2Vec' which is considered as an efficient approach to automatically learn from a text corpus [84].
  – Tokenization: Tokenization involves decomposing a stream of text into words, phrases, syllabus, or other meaning elements termed tokens. This process reduces textual data by removing unnecessary words for improved performance of the models and transforms the entire vulnerability text information into the smallest semantic unit.
  – Stop Words Filtering: Stop word filtering refers to filtering the words that appear frequently in text and contribute little or no contribution to the content or classification of text information.
  – Stemming and Lemmatization: Stemming is used to convert derived words to their base words. Most publications used the Porters stemming algorithm [85]. Stemming also includes converting the plural form of a noun to the singular form. From the perspective of data mining, these words should belong to the same category of semantically similar words. For example, using stemming 'attack', 'attacking', and 'attacked' can be expressed as 'attack'. Lemmatization is used to convert multiple words that have the same meaning into one word. For example, with lemmatization 'error', 'inaccurate', and 'false' is converted to 'error'.
  – Keyphrase extraction: Keyphrase extraction is a technique that identifies phrases that occur in a document using intrinsic properties, such as frequency and length of the phrase.
  – Information Gain (IG): IG can be used to select features by calculating the gain of each variable in the scope of the target variable [86].
- **Statistical Learners:**
  – Statistical learners are algorithms that identify patterns in historical data and make predictions on unseen data [87]. The statistical learners that are used in our set of 46 publications are:
  Decision Tree (DT): DT is a statistical learner where each fork is a split in a predictor variable and each node at the end has a prediction for the target variable [88].
  K Nearest Neighbor (KNN): KNN is a statistical learner that stores all available prediction outcomes based on training data and classifies test data based on similarity measures [89].
  Support Vector Machine (SVM): Support vector machines predict labels by quantifying the amount of separation for features between multiple classes [90].

Naive Bayes (NB): To predict the class of unknown data sets, NB works on the Bayesian theorem of probability with the assumption of conditional independence between every pair of features given the value of the class variable [91].
Logistic Regression (LR): LR estimates the probability that a data point belongs to a certain class, given the values of features [92]. LR provides good performance for classification if the features are roughly linear [92].
Neural Network (NN): The neural network is a series of statistical learners that aim to identify underlying relationships in a data set through a mechanism that mimics the way the human brain works [93].
Ensemble Method: Ensemble methods are techniques that create multiple statistical learners and then combine them to produce improved results. Ensemble methods produce more accurate solutions than a single learner would. Three are three types of ensemble methods: bagging, boosting, and stacking [94].
Bagging: Bagging ensembles statistical learners that are similar in nature, learns them independently from each other in parallel, and combines them. RF is an example of ensemble tree-based statistical learners that creates a set of DTs from a random selected subset of training set [95].
Boosting: Boosting ensembles statistical learners that are similar in nature, learns them sequentially in an adaptive way (a base model depends on the previous ones) and combines them in a deterministic manner.
Stacking: Stacking ensembles statistical learners that are not similar in nature, learns them in parallel, and combines them to output a prediction.

– Semi-supervised learners [87] not only learn from historical data, but also extrapolate their conclusions to unseen data. P46 [96] used an algorithm called 'CoForest', which applies semi-supervised learning on RF. It is a disagreement-based, semi-supervised learner proposed by Li and Zhou [97]. P10 [43] used self-training, as defined by Nigam and Ghani [98], to implement a semi-supervised model.
– Unsupervised learners [87] do not rely on historical data to find patterns.
– Active learning [99] is an iterative process where a statistical learner can interactively query a user or other information source to label new data points with the desired outputs. There are situations in which unlabeled security bug reports are abundant but manual labeling is expensive. In such a scenario, statistical learners can query for labels.

- **Evaluation Measure:** For evaluation, our set of 46 publications use the following metrics:
Classification Metrics: Vulnerability classification is a binary classification problem that can have two kinds of errors: false positive (FP) and false negative (FN). FP occurs when a file is classified as vulnerable when it is not. FN occurs when a file is classified as non-vulnerable when it has one or more vulnerabilities. Correct classifications are called True Positives (TP) and True Negatives (TN). These four metrics are summarized in Table 12, which is known as a confusion matrix. The extended version of this confusion matrix is used for multi-class classification predictions. Using TP, FP, TN, and FN the following metrics can be calculated as done in our set of 46 publications:

-- Accuracy: Accuracy is defined as the percentage of correct predictions for the test data. Equation 2 is used to calculate accuracy.

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

-- Precision: Precision is defined as the fraction of relevant examples among all of the examples that were predicted to belong in a certain class. Equation 3 is used to calculate precision.

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

-- Recall: Recall is defined as the fraction of examples that were predicted to belong to a class with respect to all of the examples that belong in the class. Equation 4 is used to calculate recall.

$$Recall = \frac{TP}{TP + FN} \quad (4)$$

-- FPR Positive Rate (FPR): FPR is defined as the fraction of examples which were predicted not to belong to a class with respect to all of the examples that do not belong in the class. Equation 5 is used to calculate FPR.

$$FPR = \frac{FP}{FP + TN} \quad (5)$$

-- F-score: F-score is the harmonic mean of precision and recall. Increase in precision, often decreases recall, and vice-versa [100]. F-score is calculated using Equation 6.

$$F - score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (6)$$

TABLE 12: Binary Classification Measurement

	Predicted Yes	Predicted No
Actual Yes	TP	FN
Actual No	FP	TN

-- ROC Curve: A receiver operating characteristic (ROC) curve is a graph showing the efficiency of a classification model at all classification thresholds. A ROC curve plots true positive rate vs. false positive rate at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both true positives and false positives. Figure 4 shows an example ROC curve.

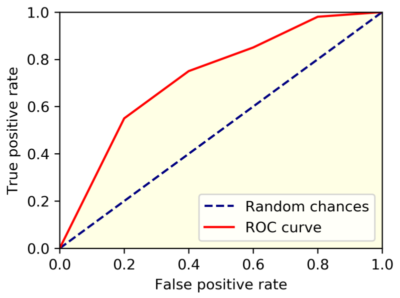


FIGURE 4: ROC Curve

- AUC: AUC (Area under the ROC Curve) measures the entire two-dimensional area underneath the entire ROC curve from (0,0) to (1,1). Figure 5 shows the AUC for a typical ROC curve. AUC provides an aggregate performance assessment across all possible classification thresholds.

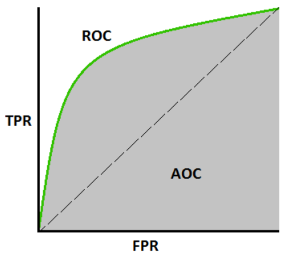


FIGURE 5: AUC Score

Regression Metrics: Evaluation metrics for regression models are different from that of classification models because regression works with predicting in a continuous range instead of a limited number of classes.

- Mean Squared Error (MSE): MSE is defined as the average of squared differences between the predicted output and the true output. Equation 7 is used to calculate MSE.

$$MSE(y_{true}, y_{pred}) = \frac{1}{n_{samples}} \sum (y_{true} - y_{pred})^2 \tag{7}$$

In Equation 7, y_{true}, y_{pred} and $n_{samples}$ respectively, stands for true output, predicted output and number of samples.

2) Usage of Techniques Amongst Publications

Vulnerability Classification

Out of 42 publications that investigate vulnerability classification, 25, 38, and 35 publications respectively use text mining, statistical learners, and evaluation measures.

Vulnerability Summarization

Of the 5 publications related to vulnerability summarization, P38 and P15 use text mining techniques. P38 [47] used word embedding techniques, while P15 [69] used TF-IDF. 3 of the 5 publications, namely P9, P15, and P41 used classification metrics as evaluation techniques.

Vulnerability Dataset Construction

We notice P10, which is related to vulnerability dataset construction, to use text mining methods. P10 [43] used word2vec for word embedding. All 3 papers related to vulnerability dataset construction reported precision and recall. Only P10 reported AUC scores.

Out of the 46 publications, 15 used TF-IDF method as shown in Table 13. 8 publications used word embedding techniques, such as word2vec to obtain the vectors for the representation of text features of security bug reports. 9 publications used tokenization in their feature selection process. 4 publications namely, P19, P24, P36, and P44 used the library NLTK [101] to implement NLP techniques. We observe 2 publications to use a statistical test called the Chi-Squared [102] test.

Table 14 shows what statistical learners are used in our set of 46 publications. Bayesian learners, such as NB are used by 20 publications. Of the 19 publications that used ensemble method, 17 used RF. We also observe publications used a combination of learners to implement bagging (P5, P15), boosting (P14) and stacking (P10, P16) algorithms. 13 publications used variants of neural networks, such as multi-layer perceptron (P5), CNN (P2), and RNN (P28).

The majority of the publications in our set used precision and recall to evaluate their statistical learners as shown in Table 15. Of the 46 publications, 65% used precision for evaluating their learners.

TABLE 13: Feature Selection Methods Used in Our Set of Publications

Feature Selection method	Publication Index
TF-IDF	P2, P11, P12, P13, P15, P17, P22, P24, P25, P28, P30, P33, P35, P37, P39
Work Embedding	P7, P10, P16, P22, P24, P28, P31, P38
Tokenization	P2, P4, P17, P19, P29, P31, P36, P37, P44
Stop Words Removal	P2, P12, P17, P19, P24, P27, P29, P36, P37, P45
Stemming/Lemmatization	P2, P17, P24, P27, P29, P36, P45
Index Term Extraction	P2
Chi-squared	P11, P36
Information Gain	P2, P17, P21

Answer to RQ4: Text mining, application of statistical learners, classification metrics, and regression metrics are commonly used techniques in our set of publications.

TABLE 14: Statistical Learners Used in Publications

Statistical Learner	Publication Index
Regression	P1, P3, P4, P5, P7, P8, P9, P13, P14, P15, P20, P22, P25, P30, P34, P37, P39, P43, P46
Bayesian Network	P3, P4, P5, P6, P7, P9, P11, P12, P15, P16, P18, P20, P21, P22, P24, P25, P29, P30, P36, P37, P45
Decision Tree	P2, P4, P14, P15, P25, P30, P45
Support Vector Machine	P2, P7, P15, P16, P19, P21, P22, P24, P26, P30, P32, P33, P34, P35
K-Nearest Neighbor	P2, P7, P14, P16, P20, P24, P37, P39, P43
Neural Network	P1, P2, P5, P7, P13, P17, P18, P20, P27, P28, P30, P37, P39
Ensemble Method	P2, P3, P4, P5, P7, P9, P10, P14, P15, P16, P20, P21, P22, P24, P30, P31, P37, P39, P46
Unsupervised Learning	P24
Semi-supervised Learning	P10, P35, P38, P46
Active Learning	P26

TABLE 15: Evaluation Measures Used in Publications

Evaluation measure	Publication Index
Accuracy	P2, P5, P12, P15, P17, P18, P19, P24, P25, P27, P28, P31, P33, P46
Precision	P1, P2, P3, P4, P5, P6, P7, P9, P10, P12, P14, P16, P17, P19, P20, P22, P24, P27, P28, P30, P31, P32, P34, P35, P37, P39, P41, P42, P44, P46
Recall	P2, P3, P4, P5, P6, P7, P9, P10, P14, P16, P17, P19, P20, P22, P24, P26, P27, P28, P30, P31, P32, P34, P35, P37, P39, P41, P42, P44, P46
F-score	P2, P3, P4, P5, P7, P9, P17, P19, P22, P24, P28, P30, P33, P35, P37, P39
AUC	P2, P10, P21, P25, P30, P36
Goodness of Fit	P15, P8, P36, P42
P-value	P8, P15
MSE	P13, P30

VI. DISCUSSION

We discuss our findings in this section.

A. SUMMARY OF RESEARCH THAT USE SECURITY BUG REPORTS FOR SOFTWARE VULNERABILITY RESEARCH

We summarize our research findings as follows:

Publication Count: We have collected 46 publications from our initial search results of 45,077 publications. Table 8 shows that amongst 46 publications, the count of publications is highest in 2019, indicating an increasing trend in the field of software vulnerability research.

Studied Topics: With respect to publication count vulnerability classification is the most frequent topic. While we observe extensive research related to vulnerability classification in software projects, the outcomes of these publications are yet to impact practice [103] [44] [53]. We foresee that researchers will build upon existing work and invest more efforts in vulnerability classification before acceptable performance will be achieved.

Our mapping study shows vulnerability dataset construction to also be of interest to researchers. Furthermore, we observe researchers to work on vulnerability report summarization.

Usage of Security Bug Reports : We find our studied publications to use GitHub, Jira, or Bugzilla to track the reported bugs of open source projects. The most commonly used bug tracking system is Bugzilla, as we observe 18 of the 46 publications to use Bugzilla to collect security bug reports. We also observe the majority of the publications to mine code churn and free form entries from bug reports for their research work.

Analysis Techniques: Security bug reports are described using natural languages, which necessitates the mined text features to be cleaned and converted to a numerical representation so that these features can be used as input to statistical learners. To convert the text data into numerical representations, we found publications to use NLP techniques, such as TF-IDF, word2vec, tokenization, and lemmatization. We also observe most of the publications in our set used supervised statistical learners, such as SVM and regression. To evaluate the statistical learners, precision and recall are used in most of the publications.

Reporting of Results: As shown in Section V, none of the publication in our set has a perfect score of 4.0 for the quality check criteria provided by Kitchenham et al. [8], [9]. We advise researchers to follow guidelines provided by Kitchenham et al. [8] [9], when reporting their findings related to software vulnerability research.

B. RESEARCH DIRECTIONS

We identify the following research directions that could be of interest to researchers:

Empirical Studies Related to Vulnerability Evolution: We advocate for empirical analysis on a large corpus of security bug reports to understand how vulnerabilities evolve through time. Such analysis may help in understanding the nature of vulnerabilities, the change in the complexity of vulnerabilities, and the change in vulnerability detection strategies.

Vulnerability Discovery Strategies: 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 NVD. Security bug reports can be leveraged to understand how vulnerabilities are discovered that could aid practitioners to characterize exploitation of vulnerabilities, which seems to be a promising research direction.

Handling Data Imbalance: Available vulnerability datasets are imbalanced as there is an unequal distribution of classes in the dataset. Imbalanced classifications pose a challenge to predictive modeling as statistical learners are usually used for classification based on the assumption of an equal number of examples for each class [104]. This results in models with low predictive performance, especially for vulnerability classification [58] [53] [46] [52]. Researchers can explore how classifiers can be designed to account for unbalanced datasets.

Vulnerability Repair: Automated defect and vulnerability repair techniques, such as example-based techniques [105], rely on a mature collection of examples to generate repairs for defects and vulnerabilities. Security bug reports that are closed can include repairs to the reported vulnerabilities. We hypothesize that the collection and curation of vulnerability repair patches could aid researchers who engage in automated vulnerability repair techniques. We predict two avenues of research in this regard: (i) empirically establish that security bug reports include legitimate vulnerability repairs, and (ii) systematically construct and curate datasets of vulnerability repair patches by mining security bug reports. Future research can also complement existing vulnerability repair techniques [106] by utilizing the security bug reports along with CVE characteristics.

Feature Selection: Researchers could be interested in mining features for better classification of vulnerabilities. Combining text features with other information in security bug reports, such as product and component information, can be used to improve the performance of vulnerability classification. The severity level attribute of a bug report is one example of properties that can be leveraged for efficient verification and validation efforts in order to mitigate vulnerabilities early. Future work can also investigate the impact of the wrapper and embedded feature selection techniques [107] on software vulnerability classification models.

Improve Vulnerability Classification: More research efforts are desirable to develop effective methods that generate fewer false positives while classifying vulnerabilities. Researchers can investigate hyper-parameter tuning methods [108] to further improve the performance of prediction models. Unsupervised learners might also be a promising step forward in the applicability of prediction models to real problems.

Transfer Learning: Research efforts can be allocated to investigate whether constructed datasets can perform cross-project. This research problem is challenging as target projects often do not have enough training data. Transfer learning can be used to solve this challenge by learning patterns from one project and applying the patterns to another project [109] [110]. None of the publications in our set of 46 have investigated how transfer learning can be helpful for vulnerability-related research, such as vulnerability classification and vulnerability summarization. Existing literature have used transfer learning for defect prediction [111] [112] [113]. Since, vulnerability classification research uses similar metrics as defect prediction [54], it might be possible to use transfer learning in vulnerability-related research successfully.

VII. THREATS TO VALIDITY

In this section, we discuss the limitation of our SMS.

- **Conclusion Validity**: We apply a set of inclusion and exclusion criteria for selecting publications that use security bug reports for software vulnerability research. We acknowledge that the selection process for these publications can be subjective, with the possibility of missing relevant publications. We reduce subjectivity by using two raters who have independently identified which publications use security bug reports for software vulnerability research.
  We also used open coding to determine the topics that are explored in the selected publications. We acknowledge the process of generating topics may be subjective. We mitigate this limitation by using two qualitative raters who independently perform the process.
- **External Validity**: Our findings are not generalizable as our SMS depends on our collection of 46 publications collected in August 2020. We mitigate this threat by using five scholar databases recommended by Kurhamm et al. [25]. To mitigate this limitation we used two raters, and reported all the inclusion/exclusion reasons as suggested by Kitchenham et al. [8].
- **Internal Validity**: We acknowledge that our search process is not comprehensive. We used five scholar databases to mitigate this limitation. Our two search strings also might not be comprehensive, as the search strings may leave out publications during the search process. We mitigated this limitation by constructing a quasi gold set and measuring the quasi-sensitivity metric (QSM) with a score of 1.0.

VIII. CONCLUSION

Characterization of vulnerabilities in software projects is pivotal to secure software development. An SMS of publications that use security bug reports for software vulnerability research can characterize existing publications and lay the groundwork to conduct new research activities, which may yield techniques and tools to facilitate secure software development.

Using five scholar databases, we identify 46 publications that use security bug reports through a systematic inclusion and exclusion criteria. We identify three topics that are addressed in our set of 46 publications by conducting open coding. These three topics are: (i) vulnerability classification, (ii) vulnerability summarization, and (iii) vulnerability dataset construction. Vulnerability classification is the most frequently investigated topic in our set of publications. We also observe that text mining, application of statistical learners, and use of classification and regression metrics, are commonly used techniques in publications that use security bug reports for software vulnerability research.

Although these papers have advanced the field of vulnerability analysis, our SMS indicates that there are potential research opportunities for further development in this field. As vulnerabilities in software systems can cause serious consequences, we advocate for research studies that derive techniques for improving classification results that can help practitioners in early mitigation of software vulnerabilities. With regard to the reporting of research results, we advise researchers to follow the guidelines from Kitchenham et al. [8] on writing good publications. We hope our SMS will facilitate more research in the domain of software vulnerabilities.

ACKNOWLEDGMENT

We thank the PASER group at Tennessee Technological University (TTU) for their valuable feedback. This research was partially funded by the National Science Foundation (NSF) award # 2026869 and the Cybersecurity Education, Research and Outreach Center (CEROC) at TTU.

A. Appendix

Table A1: List of 10 Publications Included in the Quasi-Gold Set

Index	Publication
QG1	Shin, Yonghee, and Laurie Williams. "Can traditional fault prediction models be used for vulnerability prediction?." Empirical Software Engineering 18.1 (2013): 25-59.
QG2	Scandariato, Riccardo, et al. "Predicting vulnerable software components via text mining." IEEE Transactions on Software Engineering 40.10 (2014): 993-1006.
QG3	Gegick, Michael, Pete Rotella, and Tao Xie. "Identifying security bug reports via text mining: An industrial case study." 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010). IEEE, 2010.
QG4	Goseva-Popstojanova, Katerina, and Jacob Tyo. "Identification of security related bug reports via text mining using supervised and unsupervised classification." 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS). IEEE, 2018.
QG5	Wijayasekara, Dumidu, et al. "Mining bug databases for unidentified software vulnerabilities." 2012 5th International Conference on Human System Interactions. IEEE, 2012.
QG6	Wijayasekara, Dumidu, Milos Manic, and Miles McQueen. "Vulnerability identification and classification via text mining bug databases." IECON 2014-40th Annual Conference of the IEEE Industrial Electronics Society. IEEE, 2014.
QG7	Peters, Fayola, et al. "Text filtering and ranking for security bug report prediction." IEEE Transactions on Software Engineering (2017).
QG8	Shin, Yonghee, et al. "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities." IEEE transactions on software engineering 37.6 (2010): 772-787.
QG9	Neuhaus, Stephan, et al. "Predicting vulnerable software components." Proceedings of the 14th ACM conference on Computer and communications security. 2007.
QG10	Walden, James, Jeff Stuckman, and Riccardo Scandariato. "Predicting vulnerable components: Software metrics vs text mining." 2014 IEEE 25th international symposium on software reliability engineering. IEEE, 2014.

Table A2: List of 46 Publications for the Systematic Mapping Study

Index	Publication
P1	Williams, Mark A., et al. "A vulnerability analysis and prediction framework." Computers & Security 92 (2020): 101751.
P2	Chen, Jinfu, et al. "An Automatic Software Vulnerability Classification Framework Using Term Frequency-Inverse Gravity Moment and Feature Selection." Journal of Systems and Software (2020): 110616.
P3	Shin, Yonghee, and Laurie Williams. "Can traditional fault prediction models be used for vulnerability prediction?." Empirical Software Engineering 18.1 (2013): 25-59.
P4	Theisen, Christopher, and Laurie Williams. "Better together: Comparing vulnerability prediction models." Information and Software Technology 119 (2020): 106204.
P5	Wu, Xiaoxue, et al. "CVE-assisted large-scale security bug report dataset construction method." Journal of Systems and Software 160 (2020): 110456.
P6	Shin, Yonghee, et al. "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities." IEEE transactions on software engineering 37.6 (2010): 772-787.
P7	Jiang, Yuan, et al. "LTRWES: A new framework for security bug report detection." Information and Software Technology (2020): 106314.
P8	Alhazmi, Omar H., Yashwant K. Malaiya, and Indrajit Ray. "Measuring, analyzing and predicting security vulnerabilities in software systems." computers & security 26.3 (2007): 219-228.

Continued on next page

Table A2 – continued from previous page

Index	Publication
P9	Russo, Ernesto Rosario, et al. "Summarizing vulnerabilities' descriptions to support experts during vulnerability assessment activities." Journal of Systems and Software 156 (2019): 84-99.
P10	Chen, Yang, et al. "A Machine Learning Approach for Vulnerability Curation." Proceedings of the 17th International Conference on Mining Software Repositories. 2020.
P11	Patel, Krishna A., and Rohan C. Prajapati. "A Survey-Vulnerability Classification of Bug Reports using Multiple Machine Learning Approach." Compusoft 5.3 (2016): 2071.
P12	Behl, Diksha, Sahil Handa, and Anuja Arora. "A bug mining tool to identify and analyze security bugs using naive bayes and tf-idf." 2014 International Conference on Reliability Optimization and Information Technology (ICROIT). IEEE, 2014.
P13	Zhang, Su, Doina Caragea, and Xinming Ou. "An empirical study on using the national vulnerability database to predict software vulnerabilities." International conference on database and expert systems applications. Springer, Berlin, Heidelberg, 2011.
P14	Jimenez, Matthieu, et al. "An Empirical Study on Vulnerability Prediction of Open-Source Software Releases." Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE). 2019.
P15	Gonzalez, Danielle, Holly Hastings, and Mehdi Mirakhorli. "Automated Characterization of Software Vulnerabilities." 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2019.
P16	Zhou, Yaqin, and Asankhaya Sharma. "Automated identification of security issues from commit messages and bug reports." Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. 2017.
P17	Huang, Guoyan, et al. "Automatic classification method for software vulnerability based on deep neural network." IEEE Access 7 (2019): 28291-28298.
P18	Gawron, Marian, Feng Cheng, and Christoph Meinel. "Automatic vulnerability classification using machine learning." International Conference on Risks and Security of Internet and Systems. Springer, Cham, 2017.
P19	Zou, Deqing, et al. "Automatically identifying security bug reports via multitype features analysis." Australasian Conference on Information Security and Privacy. Springer, Cham, 2018.
P20	Shu, Rui, et al. "Better security bug report classification via hyperparameter optimization." arXiv preprint arXiv:1905.06872 (2019).
P21	Chen, Qiuyuan, et al. "Categorizing and predicting invalid vulnerabilities on common vulnerabilities and exposures." 2018 25th Asia-Pacific Software Engineering Conference (APSEC). IEEE, 2018.
P22	Adhikari, Thamali Madhushani, and Yan Wu. "Classifying Software Vulnerabilities by Using the Bugs Framework." 2020 8th International Symposium on Digital Forensics and Security (ISDFS). IEEE, 2020.
P23	Goseva-Popstojanova, Katerina, and Jacob Tyo. "Experience report: security vulnerability profiles of mission critical software: empirical analysis of security related bug reports." 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2017.
P24	Goseva-Popstojanova, Katerina, and Jacob Tyo. "Identification of security related bug reports via text mining using supervised and unsupervised classification." 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS). IEEE, 2018.
P25	Pereira, Mayana, Alok Kumar, and Scott Cristiansen. "Identifying Security Bug Reports Based Solely on Report Titles and Noisy Data." 2019 IEEE International Conference on Smart Computing (SMARTCOMP). IEEE, 2019.
P26	Yu, Zhe, et al. "Improving vulnerability inspection efficiency using active learning." IEEE Transactions on Software Engineering (2019).
P27	Gegick, Michael, Pete Rotella, and Tao Xie. "Identifying security bug reports via text mining: An industrial case study." 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010). IEEE, 2010.

Continued on next page

Table A2 – continued from previous page

Index	Publication
P28	Han, Zhuobing, et al. "Learning to predict severity of software vulnerability using only vulnerability description." 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2017.
P29	Wijayasekara, Dumidu, et al. "Mining bug databases for unidentified software vulnerabilities." 2012 5th International Conference on Human System Interactions. IEEE, 2012.
P30	Bulut, Fatma Gül, Haluk Altunel, and Ayşe Tosun. "Predicting Software Vulnerabilities Using Topic Modeling with Issues." 2019 4th International Conference on Computer Science and Engineering (UBMK). IEEE, 2019.
P31	Walden, James, Jeff Stuckman, and Riccardo Scandariato. "Predicting vulnerable components: Software metrics vs text mining." 2014 IEEE 25th international symposium on software reliability engineering. IEEE, 2014.
P32	Neuhaus, Stephan, et al. "Predicting vulnerable software components." Proceedings of the 14th ACM conference on Computer and communications security. 2007.
P33	Gromova, Anna, et al. "Raising the Quality of Bug Reports by Predicting Software Defect Indicators." 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C). IEEE, 2019.
P34	Zimmermann, Thomas, Nachiappan Nagappan, and Laurie Williams. "Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista." 2010 Third International Conference on Software Testing, Verification and Validation. IEEE, 2010.
P35	Mostafa, Shaikh, et al. "SAIS: Self-Adaptive Identification of Security Bug Reports." IEEE Transactions on Dependable and Secure Computing (2019).
P36	Das, Dipok Chandra, and Md Rayhanur Rahman. "Security and performance bug reports identification with class-imbalance sampling and feature selection." 2018 Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and 2018 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR). IEEE, 2018.
P37	Peters, Fayola, et al. "Text filtering and ranking for security bug report prediction." IEEE Transactions on Software Engineering (2017).
P38	Peeples, Cody R., Pete Rotella, and Mark-David McLaughlin. "Textual analysis of security bug reports." 2017 IEEE International Symposium on Technologies for Homeland Security (HST). IEEE, 2017.
P39	Kudjo, Patrick Kwaku, et al. "The effect of Bellwether analysis on software vulnerability severity prediction models." Software Quality Journal (2020): 1-34.
P40	Sultana, Kazi Zakia. "Towards a software vulnerability prediction model using traceable code patterns and software metrics." 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2017.
P41	Feng, Xuan, et al. "Understanding and securing device vulnerabilities through automated bug report analysis." SEC'19: Proceedings of the 28th USENIX Conference on Security Symposium. 2019.
P42	Smith, Ben, and Laurie Williams. "Using SQL hotspots in a prioritization heuristic for detecting all types of web application vulnerabilities." 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation. IEEE, 2011.
P43	Last, David. "Using historical software vulnerability data to forecast future vulnerabilities." 2015 Resilience Week (RWS). IEEE, 2015.
P44	Jimenez, Matthieu, Mike Papadakis, and Yves Le Traon. "Vulnerability prediction models: A case study on the linux kernel." 2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM). IEEE, 2016.
P45	Wijayasekara, Dumidu, Milos Manic, and Miles McQueen. "Vulnerability identification and classification via text mining bug databases." IECON 2014-40th Annual Conference of the IEEE Industrial Electronics Society. IEEE, 2014.
P46	Shar, Lwin Khin, Lionel C. Briand, and Hee Beng Kuan Tan. "Web application vulnerability prediction using hybrid program analysis and machine learning." IEEE Transactions on dependable and secure computing 12.6 (2014): 688-707.

REFERENCES

[1] P. Doctor Inc., "Pc diagnostic & system information solutions pre-installed on pc/android systems," https://www.pc-doctor.com/solutions/oems, 2019, [Online; accessed 21-Oct-2020].
[2] D. Wijayasekara, M. Manic, and M. McQueen, "Vulnerability identification and classification via text mining bug databases," in *IECON 2014-40th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2014, pp. 3612–3618.
[3] K. A. Patel and R. C. Prajapati, "A survey-vulnerability classification of bug reports using multiple machine learning approach," *Compusoft*, vol. 5, no. 3, p. 2071, 2016.
[4] J. Chen, P. K. Kudjo, S. Mensah, S. A. Brown, and G. Akorfu, "An automatic software vulnerability classification framework using term frequency-inverse gravity moment and feature selection," *Journal of Systems and Software*, p. 110616, 2020.
[5] P. K. Kudjo, J. Chen, S. Mensah, R. Amankwah, and C. Kudjo, "The effect of bellwether analysis on software vulnerability severity prediction models," *Software Quality Journal*, pp. 1–34, 2020.
[6] M. A. Williams, R. C. Barranco, S. M. Naim, S. Dey, M. S. Hossain, and M. Akbar, "A vulnerability analysis and prediction framework," *Computers & Security*, vol. 92, p. 101751, 2020.
[7] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," *Information and Software Technology*, vol. 64, pp. 1–18, 2015.
[8] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," 2007.
[9] B. Kitchenham, D. I. Sjøberg, T. Dybå, O. P. Brereton, D. Budgen, M. Höst, and P. Runeson, "Trends in the quality of human-centric software engineering experiments–a quasi-experiment," *IEEE Transactions on Software Engineering*, vol. 39, no. 7, pp. 1002–1017, 2012.
[10] J. Saldaña, *The coding manual for qualitative researchers*. Sage, 2015.
[11] Ehsan Akhgari, "Bug 1284372 (cve-2016-5267)," https://bugzilla.mozilla.org/show_bug.cgi?id=1284372, 2016, [Online; accessed 19-Oct-2020].
[12] N. M. Mohammed, M. Niazi, M. Alshayeb, and S. Mahmood, "Exploring software security approaches in software development lifecycle: A systematic mapping study," *Computer Standards & Interfaces*, vol. 50, pp. 107–115, 2017.
[13] K. Morris, *Infrastructure as code: managing servers in the cloud*. " O'Reilly Media, Inc.", 2016.
[14] A. Rahman, R. Mahdavi-Hezaveh, and L. Williams, "A systematic mapping study of infrastructure as code research," *Information and Software Technology*, vol. 108, pp. 65–77, 2019.
[15] S. Zein, N. Salleh, and J. Grundy, "A systematic mapping study of mobile application testing techniques," *Journal of Systems and Software*, vol. 117, pp. 334–356, 2016.
[16] B. A. Kitchenham, D. Budgen, and O. P. Brereton, "Using mapping studies as the basis for further research–a participant-observer case study," *Information and Software Technology*, vol. 53, no. 6, pp. 638–651, 2011.
[17] P. Morrison, D. Moye, R. Pandita, and L. Williams, "Mapping the field of software life cycle security metrics," *Information and Software Technology*, vol. 102, pp. 146–159, 2018.
[18] E. Venson, X. Guo, Z. Yan, and B. Boehm, "Costing secure software development: A systematic mapping study," in *Proceedings of the 14th International Conference on Availability, Reliability and Security*, 2019, pp. 1–11.
[19] K. Curcio, T. Navarro, A. Malucelli, and S. Reinehr, "Requirements engineering: A systematic mapping study in agile software development," *Journal of Systems and Software*, vol. 139, pp. 32–50, 2018.
[20] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture," in *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2016, pp. 44–51.
[21] C. Pahl and P. Jamshidi, "Microservices: A systematic mapping study." in *CLOSER (1)*, 2016, pp. 137–146.
[22] M. Alharby and A. Van Moorsel, "Blockchain-based smart contracts: A systematic mapping study," *arXiv preprint arXiv:1710.06372*, 2017.
[23] D. Macrinici, C. Cartofeanu, and S. Gao, "Smart contract applications within blockchain technology: A systematic mapping study," *Telematics and Informatics*, vol. 35, no. 8, pp. 2337–2354, 2018.
[24] M. Felderer and J. C. Carver, "Guidelines for systematic mapping studies in security engineering," *arXiv preprint arXiv:1801.06810*, 2018.
[25] M. Kuhrmann, D. M. Fernández, and M. Daneva, "On the pragmatic design of literature studies in software engineering: an experience-based guideline," *Empirical software engineering*, vol. 22, no. 6, pp. 2852–2891, 2017.
[26] K. R. Conner, P. R. Duberstein, Y. Conwell, L. Seidlitz, and E. D. Caine, "Psychological vulnerability to completed suicide: a review of empirical studies," *Suicide and Life-Threatening Behavior*, vol. 31, no. 4, pp. 367–385, 2001.
[27] S. L. Cutter and C. Finch, "Temporal and spatial changes in social vulnerability to natural hazards," *Proceedings of the National Academy of Sciences*, vol. 105, no. 7, pp. 2301–2306, 2008.
[28] H. Zhang, M. A. Babar, and P. Tell, "Identifying relevant studies in software engineering," *Information and Software Technology*, vol. 53, no. 6, pp. 625–637, 2011.
[29] A. H. Goodall, "Highly cited leaders and the performance of research universities," *Research Policy*, vol. 38, no. 7, pp. 1079 – 1092, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S004873330900095X
[30] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.
[31] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 37–46, 1960. [Online]. Available: http://dx.doi.org/10.1177/001316446002000104
[32] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *Biometrics*, vol. 33, no. 1, pp. 159–174, 1977. [Online]. Available: http://www.jstor.org/stable/2529310
[33] R. K. Yin, *Case study research and applications: Design and methods*. Sage publications, 2017.
[34] Y.-C. Zhang, Q. Wei, Z.-L. Liu, and Y. Zhou, "Research on the architecture of vulnerability discovery technology," in *2010 International Conference on Machine Learning and Cybernetics*, vol. 6. IEEE, 2010, pp. 2854–2859.
[35] A. D. Dotz, "At the edges: Vulnerability, prediction, and resilience," in *Yearbook on Space Policy 2016*. Springer, 2018, pp. 255–264.
[36] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam, and J. Rosenberg, "Preliminary guidelines for empirical research in software engineering," *IEEE Transactions on software engineering*, vol. 28, no. 8, pp. 721–734, 2002.
[37] D. Zou, Z. Deng, Z. Li, and H. Jin, "Automatically identifying security bug reports via multitype features analysis," in *Australasian Conference on Information Security and Privacy*. Springer, 2018, pp. 619–633.
[38] X. Wu, W. Zheng, X. Chen, F. Wang, and D. Mu, "Cve-assisted large-scale security bug report dataset construction method," *Journal of Systems and Software*, vol. 160, p. 110456, 2020.
[39] D. Spencer, *Card sorting: Designing usable categories*. Rosenfeld Media, 2009.
[40] K. Goseva-Popstojanova and J. Tyo, "Identification of security related bug reports via text mining using supervised and unsupervised classification," in *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2018, pp. 344–355.
[41] C. Theisen and L. Williams, "Better together: Comparing vulnerability prediction models," *Information and Software Technology*, vol. 119, p. 106204, 2020.
[42] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," *IEEE transactions on software engineering*, vol. 37, no. 6, pp. 772–787, 2010.
[43] Y. Chen, A. E. Santosa, A. M. Yi, A. Sharma, A. Sharma, and D. Lo, "A machine learning approach for vulnerability curation," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 32–42.
[44] S. Zhang, D. Caragea, and X. Ou, "An empirical study on using the national vulnerability database to predict software vulnerabilities," in *International conference on database and expert systems applications*. Springer, 2011, pp. 217–231.
[45] K. Goseva-Popstojanova and J. Tyo, "Experience report: security vulnerability profiles of mission critical software: empirical analysis of security related bug reports," in *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2017, pp. 152–163.
[46] Y. Zhou and A. Sharma, "Automated identification of security issues from commit messages and bug reports," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 914–919.

[47] E. R. Russo, A. Di Sorbo, C. A. Visaggio, and G. Canfora, "Summarizing vulnerabilities' descriptions to support experts during vulnerability assessment activities," *Journal of Systems and Software*, vol. 156, pp. 84–99, 2019.

[48] F. Peters, T. Tun, Y. Yu, and B. Nuseibeh, "Text filtering and ranking for security bug report prediction," *IEEE Transactions on Software Engineering*, 2017.

[49] M. Gegick, P. Rotella, and T. Xie, "Identifying security bug reports via text mining: An industrial case study," in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. IEEE, 2010, pp. 11–20.

[50] A. Hindle, N. A. Ernst, M. W. Godfrey, and J. Mylopoulos, "Automated topic naming to support cross-project analysis of software maintenance activities," in *Proceedings of the 8th Working Conference on Mining Software Repositories*, 2011, pp. 163–172.

[51] D. Pletea, B. Vasilescu, and A. Serebrenik, "Security and emotion: sentiment analysis of security discussions on github," in *Proceedings of the 11th working conference on mining software repositories*, 2014, pp. 348–351.

[52] S. Mostafa, B. Findley, N. Meng, and X. Wang, "Sais: Self-adaptive identification of security bug reports," *IEEE Transactions on Dependable and Secure Computing*, 2019.

[53] M. Jimenez, R. Rwemalika, M. Papadakis, F. Sarro, Y. Le Traon, and M. Harman, "An empirical study on vulnerability prediction of open-source software releases," in *Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2019.

[54] Y. Shin and L. Williams, "Can traditional fault prediction models be used for vulnerability prediction?" *Empirical Software Engineering*, vol. 18, no. 1, pp. 25–59, 2013.

[55] O. H. Alhazmi, Y. K. Malaiya, and I. Ray, "Measuring, analyzing and predicting security vulnerabilities in software systems," *computers & security*, vol. 26, no. 3, pp. 219–228, 2007.

[56] D. Behl, S. Handa, and A. Arora, "A bug mining tool to identify and analyze security bugs using naive bayes and tf-idf," in *2014 International Conference on Reliability Optimization and Information Technology (ICROIT)*. IEEE, 2014, pp. 294–299.

[57] R. Shu, T. Xia, L. Williams, and T. Menzies, "Better security bug report classification via hyperparameter optimization," *arXiv preprint arXiv:1905.06872*, 2019.

[58] Y. Jiang, P. Lu, X. Su, and T. Wang, "Ltrwes: A new framework for security bug report detection," *Information and Software Technology*, p. 106314, 2020.

[59] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," in *Proceedings of the 14th ACM conference on Computer and communications security*, 2007, pp. 529–540.

[60] M. Pereira, A. Kumar, and S. Cristiansen, "Identifying security bug reports based solely on report titles and noisy data," in *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2019, pp. 39–44.

[61] Z. Yu, C. Theisen, L. Williams, and T. Menzies, "Improving vulnerability inspection efficiency using active learning," *IEEE Transactions on Software Engineering*, 2019.

[62] Z. Han, X. Li, Z. Xing, H. Liu, and Z. Feng, "Learning to predict severity of software vulnerability using only vulnerability description," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2017, pp. 125–136.

[63] B.-C. Chen, R. Ramakrishnan, J. W. Shavlik, and P. Tamma, "Bellwether analysis: Searching for cost-effective query-defined predictors in large databases," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 3, no. 1, pp. 1–49, 2009.

[64] G. Huang, Y. Li, Q. Wang, J. Ren, Y. Cheng, and X. Zhao, "Automatic classification method for software vulnerability based on deep neural network," *IEEE Access*, vol. 7, pp. 28 291–28 298, 2019.

[65] M. Gawron, F. Cheng, and C. Meinel, "Automatic vulnerability classification using machine learning," in *International Conference on Risks and Security of Internet and Systems*. Springer, 2017, pp. 3–17.

[66] Q. Chen, L. Bao, L. Li, X. Xia, and L. Cai, "Categorizing and predicting invalid vulnerabilities on common vulnerabilities and exposures," in *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2018, pp. 345–354.

[67] T. M. Adhikari and Y. Wu, "Classifying software vulnerabilities by using the bugs framework," in *2020 8th International Symposium on Digital Forensics and Security (ISDFS)*. IEEE, 2020, pp. 1–6.

[68] CWE, "Cwe: Common weakness enumeration," https://cwe.mitre.org, 2020, [Online; accessed 21-Oct-2020].

[69] D. Gonzalez, H. Hastings, and M. Mirakhorli, "Automated characterization of software vulnerabilities," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2019, pp. 135–139.

[70] H. Booth and C. Turner, "Vulnerability description ontology (vdo): a framework for characterizing vulnerabilities," National Institute of Standards and Technology, Tech. Rep., 2016.

[71] C. R. Peeples, P. Rotella, and M.-D. McLaughlin, "Textual analysis of security bug reports," in *2017 IEEE International Symposium on Technologies for Homeland Security (HST)*. IEEE, 2017, pp. 1–5.

[72] X. Feng, X. Liao, X. Wang, H. Wang, Q. Li, K. Yang, H. Zhu, and L. Sun, "Understanding and securing device vulnerabilities through automated bug report analysis," in *SEC'19: Proceedings of the 28th USENIX Conference on Security Symposium*, 2019.

[73] Y. Zhang, D. Lo, X. Xia, B. Xu, J. Sun, and S. Li, "Combining software metrics and text features for vulnerable file prediction," in *2015 20th International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, 2015, pp. 40–49.

[74] T. Zimmermann, N. Nagappan, and L. Williams, "Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista," in *2010 Third International Conference on Software Testing, Verification and Validation*. IEEE, 2010, pp. 421–428.

[75] B. W. Matthews, "Comparison of the predicted and observed secondary structure of t4 phage lysozyme," *Biochimica et Biophysica Acta (BBA)-Protein Structure*, vol. 405, no. 2, pp. 442–451, 1975.

[76] Wikipedia, "Github," https://en.wikipedia.org/wiki/GitHub#cite_note-11, 2020, [Online; accessed 19-Oct-2020].

[77] "Comparison of top bug tracking tools," https://instabug.com/blog/bug-tracking-tools/, 2018, [Online; accessed 23-OCT-2020].

[78] "Customers," https://Atlassian.com, 2017, [Online; accessed 19-OCT-2020].

[79] Stackshare, "Bugzilla," https://stackshare.io/bugzilla, 2020, [Online; accessed 19-Oct-2020].

[80] A. Meneely and L. Williams, "Secure open source collaboration: an empirical study of linus' law," in *Proceedings of the 16th ACM conference on Computer and communications security*, 2009, pp. 453–462.

[81] U. Brandes, *Network analysis: methodological foundations*. Springer Science & Business Media, 2005, vol. 3418.

[82] C. Manning and H. Schutze, *Foundations of statistical natural language processing*. MIT press, 1999.

[83] A. Rajaraman and J. D. Ullman, *Mining of massive datasets*. Cambridge University Press, 2011.

[84] Y. Goldberg and O. Levy, "word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method," *arXiv preprint arXiv:1402.3722*, 2014.

[85] M. F. Porter *et al.*, "An algorithm for suffix stripping." *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[86] C. Lee and G. G. Lee, "Information gain and divergence-based feature selection for machine learning-based text categorization," *Information processing & management*, vol. 42, no. 1, pp. 155–165, 2006.

[87] S. Russell and P. Norvig, "Artificial intelligence: a modern approach," 2002.

[88] L. Breiman *et al.*, *Classification and Regression Trees*, 1st ed. New York: Chapman & Hall, 1984. [Online]. Available: http://www.crcpress.com/catalog/C4841.htm

[89] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[90] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[91] J. Joyce, "Bayes' theorem," 2003.

[92] D. Freedman, *Statistical Models : Theory and Practice*. Cambridge University Press, August 2005.

[93] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception*. Elsevier, 1992, pp. 65–93.

[94] L. Rokach, "Ensemble-based classifiers," *Artificial intelligence review*, vol. 33, no. 1-2, pp. 1–39, 2010.

[95] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1. IEEE, 1995, pp. 278–282.

[96] L. K. Shar, L. C. Briand, and H. B. K. Tan, "Web application vulnerability prediction using hybrid program analysis and machine learning," *IEEE Transactions on dependable and secure computing*, vol. 12, no. 6, pp. 688–707, 2014.
[97] M. Li and Z.-H. Zhou, "Improve computer-aided diagnosis with machine learning techniques using undiagnosed samples," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 37, no. 6, pp. 1088–1098, 2007.
[98] K. Nigam and R. Ghani, "Analyzing the effectiveness and applicability of co-training," in *Proceedings of the ninth international conference on Information and knowledge management*, 2000, pp. 86–93.
[99] B. Settles, "Active learning literature survey," University of Wisconsin-Madison Department of Computer Sciences, Tech. Rep., 2009.
[100] T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald, "Problems with precision: A response to "comments on 'data mining static code attributes to learn defect predictors'"," *IEEE Trans. Softw. Eng.*, vol. 33, no. 9, pp. 637–640, Sep. 2007. [Online]. Available: http://dx.doi.org/10.1109/TSE.2007.70721
[101] E. Loper and S. Bird, "Nltk: the natural language toolkit," *arXiv preprint cs/0205028*, 2002.
[102] G. Forman, "An extensive empirical study of feature selection metrics for text classification," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1289–1305, 2003.
[103] P. Morrison, K. Herzig, B. Murphy, and L. Williams, "Challenges with applying vulnerability prediction models," in *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security*, 2015, pp. 1–9.
[104] N. Japkowicz and S. Stephen, "The class imbalance problem: A systematic study," *Intelligent data analysis*, vol. 6, no. 5, pp. 429–449, 2002.
[105] C. Liu, J. Yang, L. Tan, and M. Hafiz, "R2Fix: Automatically generating bug fixes from bug reports," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, 2013, pp. 282–291.
[106] L. Gazzola, D. Micucci, and L. Mariani, "Automatic software repair: A survey," *IEEE Transactions on Software Engineering*, vol. 45, no. 1, pp. 34–67, 2017.
[107] S. Beniwal and J. Arora, "Classification and feature selection techniques in data mining," *International journal of engineering research & technology (ijert)*, vol. 1, no. 6, pp. 1–6, 2012.
[108] D. Yogatama and G. Mann, "Efficient transfer learning method for automatic hyperparameter tuning," in *Artificial intelligence and statistics*, 2014, pp. 1077–1085.
[109] B. Wang, Y. Yao, B. Viswanath, H. Zheng, and B. Y. Zhao, "With great training comes great vulnerability: Practical attacks against transfer learning," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1281–1297.
[110] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.
[111] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Information and Software Technology*, vol. 54, no. 3, pp. 248–256, 2012.
[112] J. Chen, K. Hu, Y. Yang, Y. Liu, and Q. Xuan, "Collective transfer learning for defect prediction," *Neurocomputing*, 2019.
[113] J. Chen, Y. Yang, K. Hu, Q. Xuan, Y. Liu, and C. Yang, "Multiview transfer learning for software defect prediction," *IEEE Access*, vol. 7, pp. 8901–8916, 2019.

FARZANA AHAMED BHUIYAN is a fourth-year Ph.D. student at Tennessee Tech University. Her research interests include Software Security, Deep Learning, Multimodal Learning, and Machine Learning. She graduated with a B.Sc. in Computer Science and Engineering from Bangladesh University of Engineering and Technology. To know more about her work visit her website.

MD BULBUL SHARIF is a fourth-year Ph.D. student at Tennessee Tech University. His research interests include High-Performance Computing and Machine Learning. He graduated with a B.Sc. in Computer Science and Engineering from Bangladesh University of Engineering and Technology. He won the Best Paper Award at PASC in 2020. To know more about his work visit his website.

AKOND RAHMAN is an assistant professor at Tennessee Tech University. His research interests include DevOps and Secure Software Development. He graduated with a Ph.D. from North Carolina State University, an M.Sc. in Computer Science and Engineering from University of Connecticut, and a B.Sc. in Computer Science and Engineering from Bangladesh University of Engineering and Technology. He won the Microsoft Open Source Challenge Award in 2016, the ACM SIGSOFT Doctoral Symposium Award at ICSE in 2018, the ACM SIGSOFT Distinguished Paper Award at ICSE in 2019, and the NC State CSC and CoE Distinguished Dissertation Award in 2020. He actively collaborates with industry practitioners from IBM, RedHat, and others. To know more about his work visit his website.

• • •