Categorizing and Predicting Invalid Vulnerabilities on Common Vulnerabilities and Exposures

Qiuyuan Chen[*], Lingfeng Bao[†], Li Li[‡], Xin Xia[‡], Liang Cai[*]
[*]College of Computer Science and Technology, Zhejiang University, China
[†]School of Computer and Computing Science, Zhejiang University City College, China
[‡]Faculty of Information Technology, Monash University, Australia
{chenqiuyuan, lingfengbao, leoncai}@zju.edu.cn, {li.li, xin.xia}@monash.edu

***Abstract*—To share vulnerability information across separate databases, tools, and services, newly identified vulnerabilities are recurrently reported to Common Vulnerabilities and Exposures (CVE) database. Unfortunately, not all vulnerability reports will be accepted. Some of them might get rejected or be accepted with disputations. In this work, we refer to those rejected or disputed CVEs as invalid vulnerability reports. Invalid vulnerability reports not only cause unnecessary efforts to confirm the vulnerability but also impact the reputation of the software vendors. In this paper, we aim to understand the root causes of invalid vulnerability reports and build a prediction model to automatically identify them. To this end, we first leverage card sorting to categorize invalid vulnerability reports, from which six main reasons are observed for rejected and disputed CVEs, respectively. Then, we propose a text mining approach to predict the invalid vulnerability reports. Our experiments reveal that the proposed text mining approach can achieve an AUC score of 0.87 for predicting invalid vulnerabilities. We also discuss the implications of our study: our categorization can be used to guide new committer to avoid these traps; some root causes of invalid CVEs can be avoided by using automatic techniques or optimizing reviewing mechanism; invalid vulnerability reports data should not be neglected.**

***Index Terms*—invalid CVE, reason categorization, prediction model**

I. Introduction

Software vulnerabilities are exploitable flaws in software systems that pose significant security risks to the host computing systems [11]. Users are recurrently perplexed with the software vulnerabilities, as it is inevitable for software to confront vulnerabilities. When vulnerabilities are discovered and verified, vulnerabilities are documented and exposed to the public via a vulnerability database. One of the most influential vulnerabilities databases is Common Vulnerabilities and Exposures (CVE[1]), which is designed to allow different vulnerability databases and other capabilities to be linked together. CVE provides a standardized identifier for a given vulnerability and maintains a huge list of common identifiers for publicly known cybersecurity vulnerabilities. Though CVE is a database, for convenience, it is also frequently used to refer to a vulnerability report (i.e., each vulnerability report is a CVE or CVE entry).

Researchers have investigated software vulnerabilities in various directions. For example, some researchers are interested in feature selection of a vulnerability per se [6], [28], [29], [39] while others pay more attention to using text content to dissect vulnerabilities [4], [11], [12], e.g., assessing in which ways a system is more likely to be attacked [14], [34]. As an example, Scandariato et al. [26] and Neuhaus et al. [22] focus on revealing vulnerability locations for helping developers fix software flaws. Actually, most of the current research studies focus on exploring valid vulnerabilities, which are exposed with valid scenarios that can indeed threaten the safety of the software.

Different from all the aforementioned approaches, in this work, we are interested in invalid vulnerability reports (i.e., rejected CVEs or such CVEs that are accepted with later disputations followed by), which are usually submitted with low quality descriptions and the corresponding vulnerabilities cannot be easily verified, reproduced, or cannot really impact the safety as the reports claim. The reason why we are interested in invalid vulnerability reports is that these reports may cause a lot of troubles to the ecosystem of CVEs. Indeed, when a vulnerability report is submitted, a certain amount of professional staffs need to be allocated to validate the vulnerability. If it turns out that the vulnerability is an invalid case, the efforts used to confirm this invalidity is actually wasted. Even worse, it might be the case that the CVE is rejected because of insufficient description to reproduce the vulnerabilities, which unfortunately are valid vulnerabilities in practice. These kinds of invalid CVEs, which will not be patched because of their invalidity, will actually open opportunities for attackers to exploit them. Moreover, although invalid CVEs may not really introduce security threats to the software, it might still impact its reputation. Indeed, as argued by Telang et al. [33], software vendors are adversely affected by security-related vulnerability announcements in their products.

Because of the aforementioned reasons, we believe there is a need to understand the root causes of invalid CVEs. Hence, we need to first harvest a set of invalid CVEs. To this end, we resort to the CVE database and crawl all its recorded entries. Because CVEs are usually exposed with limited information, we resort to the America National Vulnerabilities Database (NVD[2]) to further collect (whenever possible) comments or

[1]http://cve.mitre.org/

[2]https://nvd.nist.gov/general

notes attached to the CVEs when they are reviewed. The data is recorded from 2002 in data feeds of NVD and we collect all of them until the end of 2017. This step yields 99,934 independent CVE entries associated with at least 378 vendors and 444 software products. Among the 99,934 CVEs, 5,442 of them (around 5%) are marked as invalid ones (e.g., via tags "REJECTED" and "DISPUTED" attached by authority).

In this paper, based on the harvested invalid CVEs, we conduct a manual exploratory study to understand the potential reasons why they are turned to be invalid. We leverage the open card sorting method to categorize rejected and disputed CVEs that eventually lead to six reasons for rejecting and disputing CVEs, respectively. The dominant reasons of rejecting CVEs are due to duplication or failing to reproduce, while the dominant reason of disputing CVEs is that they cannot be replicated.

After observing the root causes related to invalid CVEs, in this work, we go one step deeper to automatically predict whether a newly submitted CVE will be flagged as invalid. We hence build a machine learning model that learns from the whole dataset of invalid CVEs. The machine learning model adopts several classic classification algorithms including naive Bayes, multinomial naive Bayes, SVM and Random Forest. We use AUC to evaluate the performance of the proposed prediction models in our study because AUC is a threshold-independent measure which has a clear statistical interpretation and it is insensitive to data distribution [8], [36]. We evaluate these classifiers on 10-fold cross-validation and time sequence validation, i.e., we use data of the previous five years to predict the validity of vulnerability reports in the next five years. The experimental results show that our machine learning model is quite effective for predicting whether a CVE will be turned to invalid. Notably, the random forest classifier achieves the best performance (over 0.8 AUC). Apart from producing a prediction model, we are also interested in finding discriminative features that could help in distinguishing valid and invalid CVEs. Hence, we also present the top features based on the information gain scores of all the features.

In summary, to the best of our knowledge, this is the first work that investigates the significance of invalid CVEs (rejected and disputed). The objective of this study is to better understand how is a CVE candidate presented and why is it turned to be invalid. We expect the insights learnt from answering these questions can be used to guide new committers to avoid these traps. The main contributions of this paper are as follows:

1) We perform an exploratory study on invalid vulnerability reports that are harvested from known CVEs. By leveraging a card sorting method, we manually categorize the main reasons causing a CVE candidate be rejected or disputed.
2) We build machine learning models to automatically predict the validity of a newly submitted CVE. Our experimental results show that it is promising to automatically predict whether a CVE will be turned to invalid. We also present the top discriminative features that can help in distinguishing valid and invalid CVEs.

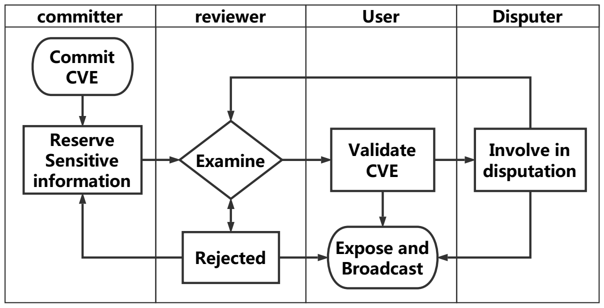


Fig. 1. Lifecycle of a CVE

The rest of the paper is structured as follows. In Section II we describe the lifecycle of a CVE and concept of invalid CVE. Section III presents data collection and dataset. Section IV presents our empirical study results. Section V presents our experiment on invalid CVE. Section VI discusses the implications of our study and the threats to validity. Section VII briefly reviews the related work. Section VIII draws conclusions and presents future work.

II. Background

In this section, we first introduce the mechanism of how a CVE entry is generated and converted to different states, then describe what is an invalid CVE entry.

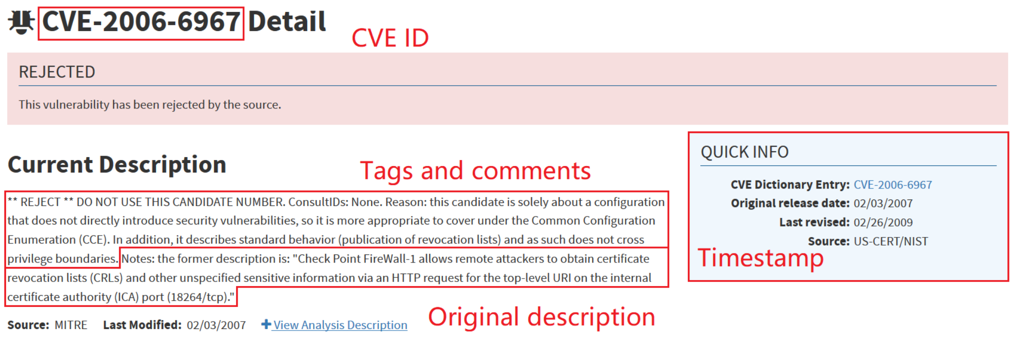
A. Lifecycle of a CVE

Figure 1 presents the lifecycle of a CVE entry. Once committers find a vulnerability in a software product, they will submit a form to the CVE Numbering Authorities[3] – the authority and the manager of CVE. Then this CVE entry will be examined in a reserved state. If reviewers validate it, it will be disclosed and exposed to the public; if it fails to pass the test, it will be rejected with comments or not. Sometimes it retains original description if the authority does not reserve it for security purpose. After a CVE is exposed, it will alert downstream companies and organizations to avoid or to fix this vulnerability. However, if practice shows that this CVE entry does not work as expectancy, or the disputer emerges, the CVE will be sent back to the validating phase. At the same time, the state of the CVE will be switched to "DISPUTED" and will be exposed to the public as well. Then after rechecking, it will be turned to be rejected or valid. Rejected CVE can be rechecked and transit back to valid status as well. The disputation status is a temporary phase before it was determined to be valid or invalid. But some of them are suspended for a long time.

B. Invalid CVE

In our study, an *invalid* CVE entry refers to a non-qualified vulnerability report and its corresponding vulnerabilities cannot be verified, reproduced, or cannot really impact the safety as the reports claim. In this paper, invalid CVE include rejected

[3]http://cve.mitre.org/cve/cna.html



CVE-2006-6967 Detail

REJECTED

This vulnerability has been rejected by the source.

Current Description

** REJECT ** DO NOT USE THIS CANDIDATE NUMBER. ConsultIDs: None. Reason: this candidate is solely about a configuration that does not directly introduce security vulnerabilities, so it is more appropriate to cover under the Common Configuration Enumeration (CCE). In addition, it describes standard behavior (publication of revocation lists) and as such does not cross privilege boundaries. Notes: the former description is: "Check Point FireWall-1 allows remote attackers to obtain certificate revocation lists (CRLs) and other unspecified sensitive information via an HTTP request for the top-level URI on the internal certificate authority (ICA) port (18264/tcp)."

Source: MITRE **Last Modified:** 02/03/2007 +View Analysis Description

QUICK INFO

CVE Dictionary Entry: CVE-2006-6967
Original release date: 02/03/2007
Last revised: 02/26/2009
Source: US-CERT/NIST

Fig. 2. An excerpt of a CVE vulnerability report.

CVE and disputed CVE. Rejected CVE will cause unnecessary troubles, e.g., wasting time to check and fix. Disputed CVE will disturb exposures, as the vulnerabilities cannot be really treated as a threat, and will also cause insignificant panic and defamation of the vendors [33].

Figure 2 is an example of an invalid CVE entry we extract from NVD. The index of this CVE is shown at the top and it is sorted by the time it was represented. The tag "** REJECTED **" indicates this is an invalid CVE. The comment of this invalid entry is used to judge the validation of the CVE and gives its explanation. And the original description introduces the vulnerability and how it works. Its original release date and last modified date are also shown on the right side.

III. Dataset

In this section, we first present the steps we collect CVE data, and then focus on invalid CVE data including identifying and preprocessing of the CVE reports and the corresponding data (i.e., description) to extract more information for further research.

A. Data Collection

We collect all data from NVD[4], a superset of CVE database and the time span of them is from 2002 to 2017. At the end of 2017, the number of CVE entries is 99,934. These vulnerabilities are discovered in a wide range of 36,436 products among the categories of applications, operating systems and hardware [11]. In addition, we find that there exist discarded pages containing historical data that cannot be retrieved by database search engine. It can only be accessed via an URL using certain CVE index number (e.g. CVE-2013-7030). We crawled pages of these rejected CVEs and got historical information which includes timestamps like released time and modified time. In the crawled data, we identified 5,442 invalid CVE entries. The statistics of valid and invalid CVE are presented in Table I.

B. Data Preprocessing

We identify the invalid records in CVE for further investigation by the assigned label `** REJECT **` and `** DISPUTED **`. We mine these raw data step by step and every step of processing will produce data for a certain

[4]https://nvd.nist.gov/vuln/data-feeds

TABLE I
Statistics of total CVE entries

Type		# vulnerability		% vulnerability	
rejected	Invalid (total)	4823	5442	4.84%	5.46%
disputed		619		0.62%	
Valid		94301		94.54%	
Total		99743			

purpose. As shown in Table I, invalid CVE entries account for up to 5%. However, due to technical and secure reasons, in some cases, the CVE entries were rejected before they are moved out of the "RESERVED" state and the original details are discarded as well. We sent emails to the authority of CVEs, but we found that they did not keep it in history, e.g., the metadata associated with the reserved state is discarded. Furthermore, we find that many notes are annotated as follows:

*** REJECT ** DO NOT USE THIS CANDIDATE NUMBER. ConsultIDs: none. Reason: The CNA or individual who requested this candidate did not associate it with any vulnerability during 20XX. Note:* ***none***

This kind of CVE does not have useful information to identify the root causes and should also be discarded. We identify these CVEs leveraging the heuristic rules as well as manual inspection. As a result, around 80% of CVEs are excluded in our work. We manually analyze the remaining 1,128 vulnerability reports to perform the characterization study.

C. Data Analysis

Our study focuses on invalid (rejected and disputed) CVE entries and we are interested in who are involved in invalid CVEs. A CVE can only be rejected by the authority, while condition is complicated in disputed CVEs: a CVE could be disputed by the vendors because it is vendors' duty to check the vulnerabilities in the received reports and give feedback timely; it could also be disputed by third parties, e.g., Red Hat or Google Secure Team. Sometimes it will also be reported by CVE authority itself. We inspect these data and list the participants of these disputations in Table II. Most of these disputations are raised by vendors, while sometimes by third parties as well.

Table III shows the number and proportion of products and vendors that the invalid CVEs have influenced. Since there are lots of rejected CVE entries that have not been connected with vendors and products before they are moved out of

TABLE II
STATISTICS OF INVOLVED VENDORS AND PRODUCTS

Participant	# of CVEs	% of CVEs
Vendor	317	51.21%
Third Party	229	37.00%
Authority	31	5.01%
Unknown	42	6.79%

TABLE III
STATISTICS OF INVOLVED VENDORS AND PRODUCTS

Type	# of Vendors	# of Products
Invalid	378	444
Valid	17093	26174
Proportion	2.20%	1.70%

"reserved" state, we cannot get the vendor names as well as the products. For these invalid CVEs, we also find that Microsoft, Symantec and PHP are very frequently mentioned vendors while Windows, norton_antivirus are most frequently mentioned products.

IV. EMPIRICAL STUDY FOR INVALID CVE CATEGORIZATION

To better understand the reasons of invalid CVEs, we conduct an empirical study based on our extracted data. Considering the impact of invalid vulnerabilities and how they were produced, we apply card sorting [3], [30] to categorize the CVE entries into groups. Card sorting is widely used to generate categories. In card sorting, participants create category names and classify entries into them. There are two phases in our card sort process: In the preparation phase, we create one card for each of the CVE entries. In the execution phase, cards are sorted into meaningful groups with a descriptive title. The first and second authors worked independently to label the card. For each card, we highlighted the keyphrases that are related to the root causes, and they categorized the cards with similar keyphrases into the same group. The two annotators then worked together to make the final agreement, and assigned a meaningful name to each of the category identified. We use Fleiss Kappa[5] [9] to measure the agreement between the two annotators. The overall Kappa value is 0.67, which indicates substantial agreement between the participants. In this way, we get six main categories for both the rejected and disputed CVE entries, which is shown in Table IV and V, respectively.

A. Categories of Rejected Vulnerabilities

Table IV shows six major categories of rejection reasons and their corresponding distribution in our collected rejected CVE. **Duplication** and **withdrawn by further investigation** are the two dominant reasons. In the following paragraphs, we discuss the detail of each category.

Duplicate: This category refers to different CVE entries that describe the same or overlapped vulnerability. Being aware of every vulnerability is scarcely possible because a software product could be spread worldwide and independent

[5]Fleiss Kappa of [0.01, 0.20], (0.20, 0.40], (0.40, 0.60], (0.60, 0.80] and (0.80, 1] is considered as slight, fair, moderate, substantial, and almost perfect agreement, respectively.

TABLE IV
STATISTICS OF THE REJECTED DATA

Rejected Reason	Description	#CVE	%CVE
Duplicate	same vulnerability with different description	642	56.91%
Withdrawn by further investigation	further investigation invalidate the original CVE	186	16.49%
Identified wrongly	the vulnerability is multiple or different identified	79	7.00%
Lack of detail	the vulnerability lacks essential authentic information	50	4.43%
Configuration	the report is about configuration setting and should be included in CCE	30	2.66%
Out of scope	the description is out of secure scope	23	2.04%
Others	potential or trivial reasons	118	10.46%

organizations may confront and report the same vulnerability. Thus, it will cause duplication problem. Some duplication is easy to understand but sometimes the situation is complicated. For example, CVE-2013-6405 is a duplication of CVE-2013-7263, CVE-2013-7264, CVE-2013-7265, and CVE-2013-7281. There is one excerpt from the duplicated CVE-2013-7263:

- *"The Linux kernel ... **obtain sensitive information** from kernel stack memory via a **(1) recvfrom, (2) recvmmsg, or (3) recvmsg** ..."*

The rest three CVE entries are about three different functions named **"l2tp_ip_recvmsg"**, **"pn_recvmsg"**, **"dgram_recvmsg"**. They all have relevant potential dangers about three system calls named **"recvfrom"**, **"recvmmsg"**, **"recvmsg"**, and they are all about the certain length of values and unsafe kernel stack memory. Since CVE-2013-6405 is also about these three system calls, it is unnecessary and is rejected.

Withdrawn by further investigation: This category refers to the accepted CVEs which turned out to be invalid after further investigation. In this category, some CVEs are rejected before moved out from the "reserved" state. And some are accepted at first, then in the future involved in disputation and rejected eventually. A real vulnerability needs to be not only judged by experts but also validated by industrial practitioners. Some CVE entries are regarded as vulnerabilities and made known to the public at first but fail to exert themselves eventually. The following comment shows one example:

- *"This was originally intended for a report about TCP Wrappers and the **hosts_ctl API function**, but further investigation showed that this was documented behavior by that function. Notes: Future CVE identifiers might be assigned to applications that **misuse the API** in a **security-relevant fashion**."*

It shows a situation where a validated behaviour was mistakenly considered as a software flaw at first because the API is in the form of the security function.

Identified Wrongly: This category refers to vulnerability reports that are imprecisely proposed, which means that the description of this CVE originally combines two or more separated issues existing in other entries. one of the rejected CVE CVE-2002-0192 has the following comment:

- *"This candidate was published with a description that **identified a different vulnerability** than what was identified in the original authoritative reference. Notes: Consult CVE-2002-0193 or CVE-2002-1564 to find the identifier for the proper issue."*

It gives an example in which reviewers recommend people not to refer to this wrongly identified entry but other appropriate CVE entries. In our data set, there are 7% of CVE that are identified wrongly.

Lack of Detail: This category refers to vulnerability reports which lack certain details or use ambiguous words. As a result, reviewers cannot easily understand what the flaw is. The description of CVE-2002-1918 is as follow:

- *"Buffer overflow in Microsoft Active Data Objects (ADO) in Microsoft MDAC 2.5 through 2.7 allows remote attackers to have **unknown** impact with **unknown** attack vectors. NOTE: due to the lack of details available regarding this issue, **perhaps it should be REJECTED**."*

We can see that it uses two "unknown" to describe their vulnerability. These words are very ambiguous so it is reasonable to be rejected. The information this CVE gives cannot support the conclusion that the "potential remote attack hazard function" is a real vulnerability.

Configuration: This category refers to a CVE that describes a configuration problem rather than a vulnerability. It is more appropriate to be covered under the Common Configuration Enumeration (CCE[6]) rather than CVE. For example, this problem is mentioned in CVE-2006-6967. In this CVE, committers considered this vulnerability would lead to the exposure of "unspecified sensitive information". But it was rejected since:

- *"This candidate is **solely about a configuration** that does not directly introduce security vulnerabilities, so it is more appropriate to cover under the Common Configuration Enumeration (CCE). In addition, it describes **standard behavior (publication of revocation lists) and as such does not cross privilege boundaries** ..."*

Reviewers confirmed that the reported problem is an error in configuration and is considered as a standard behaviour by authority. Therefore, reports similar to this are all rejected for "configuration" reason.

Out of Scope: This category refers to a CVE entry that does not describe a vulnerability but a bug, designing flaw or feature enhancement. CVE is defined to be publicly known cybersecurity vulnerabilities. Its boundary is quite ambiguous so the out-of-scope problem is usually inevitable. Once committers fail to figure out the scope of CVE, they will regard the vulnerability as a wrong problem. One of the cases is from the notes in CVE-2013-0743: authority comments that the problem about SSL certificates are not within the scope of CVE, stating that *"not a problem that is categorized as a vulnerability within CVE"*. Therefore, CVEs which are rejected with this kind of comment are categorized to "out of scope".

[6]https://cce.mitre.org/

TABLE V
STATISTICS OF THE DISPUTED DATA

Dispute Reason	Description	#CVE	%CVE
cannot replicate	the vulnerability cannot be replicated	162	26.17%
not vulnerability itself	the vulnerability is not caused by identified product itself	124	20.03%
not highly impact	acknowledged vulnerability that has limited influence	112	18.09%
misguiding	the report gives wrong information	61	9.85%
legitimate behaviour	the so-called vulnerability is legitimate	50	8.08%
out of scope	not in security boundary	44	7.11%
others	the reason is not given	66	10.66%

Others: vulnerability reports are rejected for trivial reasons, e.g., systematical errors of NVD, losing essential information leading to rejection, or some records are out of date and substituted by new committers. Therefore, the significance of these CVEs is very limited and we categorize them to others.

B. Categories of disputed vulnerabilities

Table V presents six major categories of disputed CVE entries. *Cannot replicate* is the dominant reason. We will give more explanations about each category in this section.

Cannot Replicate: This category refers to vulnerability reports in which the reported vulnerabilities cannot be reproduced. There are different obstacles impeding the reproduction. To better explain it, we will give some examples.

The first one is CVE-2008-2956, which describes a "memory leak" vulnerability via a `malformed XML`. However, a reviewer found that `malformed XML` gave little useful information and stated: *"I was **never able to identify a scenario** under which a problem occurred and the original reporter **wasn't able to supply** any sort of reproduction details."* This disability of reproduction is due to the lack of details.

Second, even if the information is sufficient when some phases get wrong, it will also lead to reproducing failing. In CVE-2007-2677, committers pointed out **"multiple PHP remote file inclusion vulnerabilities"** while one vendor disputed that *"since the code is defined within a function that is **not called** from within includes/language.php"*. This is a kind of situation where the key phrase – "call vulnerable function" – does not work, so the vulnerability is impossible to replicate.

Last, some vulnerabilities are in the dynamic running environment and the vulnerability can only be reproduced in an unstable way which successes once in several tries. CVE-2005-4486 described a **"SQL injection vulnerability"**. But the flaw was partly hidden and partly visible, which means root cause has not been specifically identified. The vendor disputed this and stated that *"although they could be dynamically generated through use of the product ... but this **could not be repeated** for news.asp."*

Not Vulnerability Itself: This category refers to vulnerability reports in which the reported vulnerabilities are actually under protection or in other relevant products.

First, some CVE entries will declare a process to be unsafe while the vendor points out that this is under the protection mechanism. As in CVE-2008-6544, the original description presented a *"remote attack vulnerability"* while CVE and multiple third parties disputed this issue because *"the files **contain a protection mechanism** against direct request"*.

Second, some reported vulnerabilities are not from the identified product but other components. One of the cases is in CVE-2006-1273: One version of Mozilla Firefox was reported to allow remote attackers to cause a denial of service and then trigger crashes. However, reviewers disputed it, since it was running in the *"IE Tab extension"* and was *"not an issue in Firefox itself"*.

Not Highly Impact: This category refers to vulnerability reports in which the reported vulnerabilities are acknowledged reluctantly but they are still disputed because the flaw and consequence are not severe enough.

One example is that in CVE-2007-0253 *"grsecurity patch"* was reported to be vulnerable but developers disputed that *"the function they claim the vulnerability to be in is a **trivial function**, which can, and has been, easily checked for any supposed vulnerabilities."* They also cited a past disclosure that was not proven. In this CVE entry, the vulnerability is disputed just to be a trivial function and will not influence the whole product.

Another example is in CVE-2007-0050: A remote attack vulnerability was reported but the vendor, as well as the third parties, found that *"there is a small time window of risk before the installation is complete."*, so they dispute that *"since the variable is set before use"*, the hazard only appeared in a very short time and this little risk window was unlikely to affect a lot.

Misguiding: This category refers to vulnerability reports which misguide the reviewer and even lead to quarrel and reproach rather than the discussion of vulnerabilities themselves. An example is CVE-2017-7397, which reported a "denial of service" vulnerability. But the vendor disputed that *"It has been proved that this vulnerability has no foundation and it is **totally fake** and based on false assumptions."* one of the cases was that the disputer criticized that the CVE gives wrong information.

And an extreme case is in CVE-2005-3497 which presented a "SQL injection vulnerability" but got vendors' strong refutation, i.e., *"this is **100% false reporting**, this is a slander campaign from a customer who had a vulnerability in his SERVER, not the software."* However, follow-up investigation strongly suggestd that **the original report is correct**. We can see that sometimes the vendors will deny the vulnerability not based on the fact but for some other reasons (e.g. reputation or profit). Besides these, when vendors think committers are not reliable, they will also dispute it.

Out of Scope: Similar to the category we identify in rejected CVEs, disputed CVEs in this category refer to the CVE entry that does not describe a vulnerability but a bug, designing flaw or feature enhancement. For example, in CVE-2017-8912 and CVE-2006-7141:

- *"CMS Made Simple (CMSMS) 2.1.6 **allows remote authenticated administrators to execute arbitrary PHP code** ..."*
- *"...when utl_file_dir is set to a wildcard value or "CREATE ANY DIRECTORY to PUBLIC" privileges exist, **allows remote authenticated users to read and modify arbitrary files** ..."*

However, the vendors of CMS commented CVE-2017-8912 as *"A **feature**, not a bug."*. Reviewers commented CVE-2006-7141 is not *"an inherent vulnerability"*. They are considered out of security boundary and are involved in disputation.

Legitimate Behaviour: This category refers to vulnerability reports which are disputed to be legitimate and totally under control.

The first example is an directory traversal vulnerability about "Cell Request Service" in HP Data Protector in CVE-2014-5160. However, the vendor of the HP disputed that this behavior is just **"by design."**. The second example is in CVE-2006-6165 which reported that a *"harmful environment variables"* allowed local users to gain privileges. But this was disputed by a third party, stating that *"it is the responsibility of the application to properly sanitize the environment."*. In this disputation, though the legacy potential harmful variables do exist, it will be properly sanitized by application, so the behaviour is also legitimate.

Besides these, some reported vulnerabilities can be under actual business logic. One of the complicated disputations between software vulnerability and business morality is in CVE-2017-8769:

- *"Facebook WhatsApp ... associated with a chat, even after that chat is deleted. There may be users who **expect file deletion to occur upon chat deletion**, or who expect encryption ... "*

This CVE described an action that the source file will not be deleted with its corresponding chat, but *"Facebook Whatsapp"* did not **"consider these to be security issues"** saying it is reasonable because users may want to preserve it **"regardless of whether its associated chat is deleted."**. In this case, the committers thought the user privacy may be strongly related to software vulnerability while the vendor assumed it did not matter, this kind of value divergence also lead to disputation.

Others: disputed vulnerability reports do not give specific contents of disputation. Most of them just comment that the vendor or a reliable third party has disputed this CVE. Therefore details of these CVEs are unclear and we categorize them to others.

V. Invalid CVE Prediction

The previous section summarizes our empirical investigation on the manual exploration of invalid CVEs. In this section, we present our approach to predict invalid CVEs.

A. Overall Framework

Figure 3 presents the overall framework of our proposed approach. The whole framework includes two phases: model building phase and prediction phase. As to our experiment

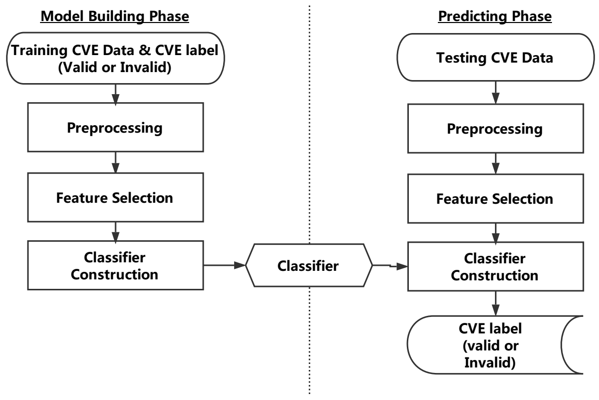


Fig. 3. Framework of our prediction model

TABLE VI
STATICS OF CVE AFTER PREPROCESSING FOR EXPERIMENT

Type		# vulnerability		% vulnerability	
rejected	Invalid	167	786	0.18%	0.83%
disputed		619		0.65%	
Valid		94153		99.17%	
Total		94939			

approach, the posterior information is not appropriate to be included. To avoid getting posterior information, we separate reviews and comments from original descriptions and exclude CVE entries which contain only reviews and comments. As shown in Table VI, after cleaning the data, we use preprocessed data to build models.

For each CVE description, our framework tokenizes them, removes stop words (e.g., a, the), stems them (e.g. reduces them to their root forms), then represents them in the form of a "bag of words" [2]. After we select textual features, our framework next constructs a classifier based on the selected textual features of the training CVE data. The model building phase would compare and contrast the features of CVE that are valid or invalid. In this paper, we investigate 4 text mining techniques, i.e., random forest, SVM, naive Bayes and multinomial naive Bayes. We also use a random guess classifier as the baseline.

In the prediction phase the classifier is then used to predict whether an unknown CVE is valid or not. For each of such CVE, our framework first preprocesses and extracts textual features from it, and represents it by using the features selected in the model building phase. Next, these features are inputted into the classifier in the classifier application step. This step would output the prediction results, i.e., valid or invalid.

B. Feature Selection

Previous studies show that feature selection techniques could improve the performance of text categorization [27], [37] We use Information Gain (IG) to measure the number of bits of information required for predicting a label [35]. We denote a vulnerability reports collection as $VR = (V_1, I_1), (V_2, I_2), ..., (V_N, I_N)$, where V_n represents the n^th vulnerability report and I_n represents whether this is invalid (i) or not ($\bar{i}$), and the terms in VR_n are denoted as $VR_n =< t_1, t_2, ..., t_{|VR_n|} >$. For a term t and label i, for a vulnerability report, there would be four possible relationships: (t, i) represents V_i contains the term t, and it is invalid (i.e. i). $(t, \bar{i})$ represents V_i contains the term t, and it is valid (i.e. $\bar{i}$). $(\bar{t}, i)$ represents V_i does not contain the term t, and it is invalid (i.e. i). $(\bar{t}, \bar{i})$ represents V_i does not contain the term t, and it is valid (i.e. $\bar{i}$). Then we can compute information gain (IG). The information gain score of term t and label i is computed as:

$$IG(t,i) = \sum_{i' \in \{i,\bar{i}\}} \sum_{t' \in \{t,\bar{t}\}} p(t', i') \times log \frac{p(t', i')}{p(t') \times p(i')} \quad (1)$$

After we apply the feature selection to compute the scores, we rank these scores from high to low to generate a ranked list. As different thresholds (i.e., 10%, 20%, 30%) are applied to our experiments and the results have little variation, we choose the top 10% of the total number of terms following the previous work [15], [35] by default.

C. Evaluation Metrics

The entries are labelled respectively as 'valid' and 'invalid'. As introduced before, what we care more about are abnormal cases, so we regard the 'invalid' as positive. In this study, we use AUC to evaluate the effectiveness of the proposed prediction models.

AUC: AUC is Area Under the ROC (Receiver Operating Characteristic), a common measure to evaluate a prediction model especially binary classifier [7], [8], [17], [19], [25]. In the ROC curve, the true positive rate (TPR) is plotted as a function of the false positive rate (FPR) across all thresholds. It is also a widely used evaluation metric in past software engineering studies [8], [36].

In summary, The reasons that we choose AUC are as follows:

1) Common threshold-dependent measures like F1-measures often rely on a probability threshold (e.g., 0.5) for constructing a confusion matrix, while AUC is independent to the threshold. Therefore, we use AUC to avoid the threshold setting problem. Some researchers also suggest using threshold-independent measures (e.g., AUC) instead of threshold-dependent measures such as F1-measure. For example, Lessmann et al. recommended AUC as the primary accuracy indicator for comparative studies [19]; Tantithamthavorn et al. suggested to use AUC to avoid conflicting conclusions [32].
2) While AUC is insensitive to imbalanced data [19] and our dataset is imbalanced, the AUC is a better choice. AUC is a more robust metric in front of class distribution than other measures such as F1-measure. It is unfair to compare two prediction models in an unbalanced dataset using other measures such as F1-measure [21], [24].
3) The AUC has a clear statistical interpretation [19]. In our approach, it can be explained that given a pair of valid and invalid CVE, the AUC will evaluate the probability of the situation in which the classifier gives higher scores to invalid CVE than valid CVE.

There is also a threshold of the results that a functional prediction with adequate classification performance requires AUC score to be above 0.7 [25].

D. Experiment Results

Based on the collected CVE data and our proposed approach, we are interested to answer the following research questions:

RQ1. Can we effectively predict invalid CVE?

Motivation: The emergence of invalid CVE is inevitable under the current reviewing mechanism. We would like to effectively predict whether a new commit of CVE will be involved in a rejection. Therefore, we use original textual contents of CVE in the database and apply different prediction models to investigate whether it is feasible to build accurate models that help to predict validity of CVE.

Approach: We use sci-kit learn tool [23] to implement the prediction models with default parameters and a baseline model, i.e. random guess prediction, is used to compare our proposed prediction models. Considering the diversity of CVE across 36,436 products, to give an overall insight, we use stratified 10-fold cross-validation to evaluate the effectiveness of the model. In the experiment, we applied oversampling technique to handle imbalance data problem. Besides, we use the same longitudinal data setup to simulate the usage of our approach in practice [31], which means we keep the time order of CVE committing to the database. According to the time order and interval of 5 years, we divide our data into three folds (2002-2007, 2008-2012, 2013-2017). Sequentially we use the previous fold as training data to predict next fold. The distributions of invalid and valid reports in 2002-2007, 2008-2012, and 2013-2017 are 509 vs. 28,309, 155 vs. 26,138, and 122 vs. 39,706, respectively.

Results: Table VII, and Table VIII present the results of AUC of each prediction model for 10-fold cross-validation and time sequence validation, respectively. On 10-fold cross-validation experiment, SVM achieves the best performance, while random forest ranks the second. On time-series experiment random forest achieves the best performance because its AUCs on all are much larger than those of the other prediction models. We apply Wilcoxon signed-rank test and find that there is no statistically significant difference between the AUCs of random-forest and other classifiers in 10-fold-cross-validation. Considering both experiments settings, all AUCs of the random forest model are larger than 0.7, which indicates promising performance [25]. Besides, random-forest has the best performance in time-sequence-validation, which simulates the usages of our approach in practice. We also use false negative rate (FNR) to evaluate the risk of misclassification. As we set the invalid reports as the positive class, FNR means the rate of misclassification of valid to invalid which is truly critical. FNRs of all models are less than 0.0001, indicating low risk of misclassification. Therefore, our approach can effectively predict whether a CVE will be turned to invalid.

TABLE VII
AUCS OF EACH PREDICTION MODEL IN TEN-FOLD CROSS-VALIDATION

	Random Guess	Naive Bayes	Multinomial Naive Bayes	SVM	Random Forest
Average	0.500	0.753	0.773	0.818	0.793

TABLE VIII
AUCS OF EACH PREDICTION MODEL IN TIME SEQUENCE VALIDATION

	Random Guess	Naive Bayes	Multinomial Naive Bayes	SVM	Random Forest
fold1 pred. fold2	0.500	0.494	0.510	0.770	0.887
fold2 pred. fold3	0.500	0.497	0.497	0.536	0.872

RQ2. What are the most important features for discriminating invalid CVEs from valid ones?

Motivation: As our training data is text content, the features are represented as term frequency corresponding to the number of times terms appear in all of the descriptions. Although there are lots of characteristics that affect a CVE to be invalid, we are also interested in finding what kind of discriminative features to what extent that could help in distinguishing valid and invalid CVE.

Approach: Based on textual content, we extract thousands of text features using term frequency. Then, we use information gain as the features important values for each feature and rank the them according to the information gain.

Results: We report the top 10 features sorted based on their information gain scores in Table IX. We notice that the information gain score is low (the highest possible value would be 1), which represents that in such a huge corpus one feature alone is not sufficient to classify valid CVE from invalid CVE. Note that “reject” is not the label but usually used to describe vulnerability action (e.g. “does not reject a negative value”). As for “php”, the proportion of php-related reports in invalid CVEs is larger than that in both valid CVEs and all CVEs. Php files are frequent attacking targets (cf. CVE-2005-4349). It induces rejection because of existent duplicate CVEs. This demonstrates the necessity to investigate invalid vulnerability reports so as to avoid such attempts (i.e., efforts wasted). Some other features, such as “inclusion”, “allows” are good indicators to identify invalid CVE because sometimes these kinds of actions descriptions are not specific enough and are rejected or disputed according to our empirical study.

VI. Discussion

A. Implication

Our categories in the empirical study give reasons why a CVE turns invalid and our prediction models achieve promising performance. Based on these results, we discuss the implications of our study.

First, ***Duplication* and *withdrawn by further investigation* easily lead to rejection**. In software engineering, various approaches have been proposed to detect duplicate bugs and bug reports [16]–[18]. Duplication detection techniques can also be used to help reduce the duplicate vulnerability reports, which should be labelled invalid. In addition, there is no proof that existing duplication detection approaches are practical for vulnerability reports. Thus, it would be beneficial to know

TABLE IX
TOP-10 MOST DISCRIMINATIVE FEATURES BASED ON INFORMATION GAIN SCORES

Invalid	Information Gain	Invalid	Information Gain
reject	0.0270	attackers	0.0063
inclusion	0.0120	php	0.0062
file	0.0099	remote	0.0060
blocked	0.0084	allows	0.0053
directory	0.0068	arbitrary	0.0049

whether the current detection tools can detect duplication in software vulnerability databases.

Also, in practice it is hard to identify the rejected CVEs due to *withdrawn by further investigation*. One possible reason is that there is a lack of effective communication tunnels. Thus, researchers should also investigate how to build an effective communication mechanism between committers and reviewers (e.g., a new review system for vulnerability reports), which will make a significant difference on the vulnerability examination and verification.

Second, we also find that ***failing to replicate*** **will induce disputation**. Once released, vulnerability reports will have a wide influence. This gives the authority of CVE an implication that carefully verifying a new-commit CVE is very important. A vulnerability report should not be exposed before it is totally replicated. Rejecting it before it comes to the public is much better than be involved in disputation. Besides these, according to our categories of invalid vulnerabilities, a guide for the new committers can be created to avoid such traps.

Last but not the least, besides efforts of the authority of CVE database, our proposed classifiers can also be extended to predict what kind of CVE is more likely to be invalid. Though by now there is a lot of reserved data, we are convinced that given enough original details our data-driven approach can play an important role in practice. Invalid vulnerabilities are valuable to research. Researchers should not neglect invalid vulnerability report data not only in CVE database but also in other vulnerability databases.

B. Threats to Validity

Threats to internal validity relate to the bias of our card sorting on invalid CVE and errors in our experiments. We have manually checked the categories to ensure each is discussed by two different people independently and reach a consensus. We have also double checked our experiments and the datasets collected from NVD, still, there could be errors that we do not notice.

Threats to external validity relate to the generalizability of our results. There are many reserved CVE that cannot be taken into account. While there is a huge number of CVE that will be invalid, but we can not have a whole look at them all because they will be discarded for safety and reputation or other reasons. In the future, we plan to reduce this threat further by trying to collecting and analyzing more invalid CVE from the CVE authority.

Threats to construct validity relate to the suitability of our evaluation measures. In our approach, AUC is used to evaluate the performance of prediction models. The AUC score is widely used to evaluate the effectiveness of various software engineering studies [17], [19]. As a threshold-independent measure, AUC is recommended to measure the performance of a classifier by many researchers [19], [32].

VII. RELATED WORK

Studies on Software Vulnerabilities. Software vulnerabilities have big impact on industrial companies, so they proposes some rating systems to categorize software vulnerabilities. There are four major rating systems, i.e., Common Vulnerability Scoring System(CVSS) [7], Microsoft Security Bulletin Severity Rating System [8], US-CERT Vulnerability Notes Database [9] and SANS Critical Vulnerability Analysis Archive [10]. They establish severity of vulnerabilities according to a certain severity definition, which many works rely on. Some studies are also done on designing vulnerability rating systems using information in vulnerability database (e.g. NVD) [10], [20]. Our study is not about the exploration of vulnerabilities severity rating or designing a rating system, but to research the vulnerability validity.

Building a category system for existent vulnerabilities can help people to identify vulnerabilities as well as to better organize and secure their products. Howard and Michael conduct a research on the category improvement and make clear different vulnerabilities [13]. Different from putting different kinds of vulnerabilities in order, our research is conducted from the perspective of whether a vulnerability report is valid or not and the reasons behind it. This complements the existing evaluation systems and gives more hints.

Prediction Model in Software Vulnerabilities. The body of work on vulnerability prediction is smaller comparing to defect prediction [12], [38]. Vulnerabilities are different from faults in that vulnerabilities always represent abusive functionalities instead of wrong or insufficient functionalities associated with faults [5]. As the definition is more strict, the size of reported vulnerabilities is much fewer than the reported faults in many projects [1]. Our work on the invalid CVE can shed light on what kind of reported vulnerability is less likely to take effect.

For the vulnerability prediction, a lot of studies have been done with combinations of different features like software metrics [26], code churn [28], developer activity metrics [6], [22], [28], code complexity [29], etc. Zimmermann et al. conducted a study on the correlation between vulnerabilities and various features including churn, code complexity, dependencies, code coverage, organizational measures and actual dependencies [39]. Along with the features in low-level granularity, text mining is also used to predict vulnerabilities [12], as it describes the vulnerabilities from a more macroscopic view. Some researchers focus on how to recognize vulnerable components [22], [26], or assess in which ways a system is more likely to be attacked [14], [34]. Han et al. proposed

[7]https://www.first.org/cvss/
[8]https://technet.microsoft.com/zhcn/security/gg309177.aspx
[9]http://www.kb.cert.org/vuls/html/fieldhelp
[10]http://www.sans.org/newsletters/cva/

a deep learning based model to predict the severity of a CVE based on CVSS [11]. Inspired by the previous work of different predicting techniques in these studies, we propose our prediction approach to predict the validity of invalid vulnerability reports for a better vulnerability analysis.

VIII. Conclusion & Future Work

In this paper, we focus on invalid CVE, which will influence the security, reputation and efficiency of vendors, downstream companies and users. First, we contribute an invalid CVE data set, which is intended to be ignored by researchers. Then we conduct an empirical study to categorize the rejected and disputed vulnerability reports and explain details for each category. Finally, we apply several prediction models to predict whether a CVE will be involved in invalid status and random forest achieves the best AUC with more than 0.87. In the future, we plan to improve the effectiveness of our proposed tool further. We also plan to experiment with even more invalid vulnerability reports from more security platforms.

References

[1] Omar H Alhazmi, Yashwant K Malaiya, and Indrajit Ray. Measuring, analyzing and predicting security vulnerabilities in software systems. *Computers & Security*, 26(3):219–228, 2007.

[2] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[3] Lingfeng Bao, David Lo, Xin Xia, Xinyu Wang, and Cong Tian. How android app developers manage power consumption?: An empirical study by mining power management commits. In *Proc. of MSR*, pages 37–48. ACM, 2016.

[4] Mehran Bozorgi, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Beyond heuristics: learning to classify vulnerabilities and predict exploits. In *KDD*, pages 105–114. ACM, 2010.

[5] Felivel Camilo, Andrew Meneely, and Meiyappan Nagappan. Do bugs foreshadow vulnerabilities? a study of the chromium project. In *MSR*, pages 269–279. IEEE, 2015.

[6] Istehad Chowdhury and Mohammad Zulkernine. Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. *Journal of Systems Architecture*, 57(3):294–313, 2011.

[7] Yuanrui Fan, Xin Xia, David Lo, and Ahmed E Hassan. Chaff from the wheat: Characterizing and determining valid bug reports. *IEEE Transactions on Software Engineering*, 2018.

[8] Yuanrui Fan, Xin Xia, David Lo, and Shanping Li. Early prediction of merged code changes to prioritize reviewing tasks. *Empirical Software Engineering*, March 2018.

[9] Joseph L Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378, 1971.

[10] Hamza Ghani, Jesus Luna, Abdelmajid Khelil, Najib Alkadri, and Neeraj Suri. Predictive vulnerability scoring in the context of insufficient information availability. In *Proc. of Risks and Security of Internet and Systems (CRiSIS)*, pages 1–8. IEEE, 2013.

[11] Zhuobing Han, Xiaohong Li, Zhenchang Xing, Hongtao Liu, and Zhiyong Feng. Learning to Predict Severity of Software Vulnerability Using Only Vulnerability Description. In *ICSME*, pages 125–136. IEEE, September 2017.

[12] Aram Hovsepyan, Riccardo Scandariato, Wouter Joosen, and James Walden. Software vulnerability prediction using text analysis techniques. In *Proceedings of the 4th international workshop on Security measurements and metrics*, pages 7–10. ACM, 2012.

[13] Michael Howard. Improving software security by eliminating the cwe top 25 vulnerabilities. *IEEE Security & Privacy*, 7(3), 2009.

[14] Michael Howard, Jon Pincus, and Jeannette M Wing. Measuring relative attack surfaces. In *Computer Security in the 21st Century*, pages 109–137. Springer, 2005.

[15] Qiao Huang, Emad Shihab, Xin Xia, David Lo, and Shanping Li. Identifying self-admitted technical debt in open source projects using text mining. *Empirical Software Engineering*, 23(1):418–451, 2018.

[16] Nicholas Jalbert and Westley Weimer. Automated duplicate detection for bug tracking systems. In *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pages 52–61. IEEE, 2008.

[17] Ahmed Lamkanfi, Serge Demeyer, Emanuel Giger, and Bart Goethals. Predicting the severity of a reported bug. In *MSR*, pages 1–10. IEEE, May 2010.

[18] Alina Lazar, Sarah Ritchey, and Bonita Sharif. Improving the accuracy of duplicate bug report detection using textual similarity measures. In *Proc. of MSR*, pages 308–311. ACM, 2014.

[19] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. *IEEE Transactions on Software Engineering*, 34(4):485–496, July 2008.

[20] Jian Luo, Kueiming Lo, and Haoran Qu. A software vulnerability rating approach based on the vulnerability database. *Journal of Applied Mathematics*, 2014, 2014.

[21] Jaechang Nam and Sunghun Kim. Clami: Defect prediction on unlabeled datasets (t). In *Proc. of ASE*, pages 452–463. IEEE, 2015.

[22] Stephan Neuhaus, Thomas Zimmermann, Christian Holler, and Andreas Zeller. Predicting vulnerable software components. In *CCS*, pages 529–540. ACM, 2007.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[24] Foyzur Rahman, Daryl Posnett, and Premkumar Devanbu. Recalling the imprecision of cross-project defect prediction. In *Proc. of FSE*, page 61. ACM, 2012.

[25] D. Romano and M. Pinzger. Using source code metrics to predict change-prone Java interfaces. In *ICSM*, pages 303–312, September 2011.

[26] Riccardo Scandariato, James Walden, Aram Hovsepyan, and Wouter Joosen. Predicting vulnerable software components via text mining. *IEEE Transactions on Software Engineering*, 40(10):993–1006, 2014.

[27] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[28] Yonghee Shin, Andrew Meneely, Laurie Williams, and Jason A Osborne. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. *IEEE Transactions on Software Engineering*, 37(6):772–787, 2011.

[29] Yonghee Shin and Laurie Williams. An empirical model to predict security vulnerabilities using code complexity metrics. In *ESEM*, pages 315–317. ACM, 2008.

[30] Donna Spencer. *Card sorting: Designing usable categories*. Rosenfeld Media, 2009.

[31] Ahmed Tamrawi, Tung Thanh Nguyen, Jafar M Al-Kofahi, and Tien N Nguyen. Fuzzy set and cache-based approach for bug triaging. In *Proc. of FSE*, pages 365–375. ACM, 2011.

[32] Chakkrit Tantithamthavorn and Ahmed E Hassan. An experience report on defect modelling in practice: Pitfalls and challenges. *forest*, 15(17):71–73, 2018.

[33] Rahul Telang and Sunil Wattal. An empirical analysis of the impact of software vulnerability announcements on firm stock price. *IEEE Transactions on Software Engineering*, 33(8):544–557, 2007.

[34] Lingyu Wang, Tania Islam, Tao Long, Anoop Singhal, and Sushil Jajodia. An attack graph-based probabilistic security metric. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 283–296. Springer, 2008.

[35] Xin Xia, David Lo, Weiwei Qiu, Xingen Wang, and Bo Zhou. Automated configuration bug report prediction using text mining. In *Proc. of COMPSAC*, pages 107–116. IEEE, 2014.

[36] Meng Yan, Xin Xia, Emad Shihab, David Lo, Jianwei Yin, and Xiaohu Yang. Automating change-level self-admitted technical debt determination. *IEEE Transactions on Software Engineering*, 2018.

[37] Yiming Yang and Jan O Pedersen. A comparative study on feature selection in text categorization. In *Icml*, volume 97, pages 412–420, 1997.

[38] Yun Zhang, David Lo, Xin Xia, Bowen Xu, Jianling Sun, and Shanping Li. Combining software metrics and text features for vulnerable file prediction. In *ICECCS*, pages 40–49. IEEE, 2015.

[39] Thomas Zimmermann, Nachiappan Nagappan, and Laurie Williams. Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista. In *ICST*, pages 421–428. IEEE, 2010.