The Journal of Systems and Software 160 (2020) 110456

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

CVE-assisted large-scale security bug report dataset construction method

Xiaoxue Wu[a], Wei Zheng[b,*], Xiang Chen[b,c], Fang Wang[d], Dejun Mu[a]

[a] School of Cybersecurity, Northwestern Polytechnical University, Xi'an 710072, China
[b] School of Software, Northwestern Polytechnical University, Xi'an 710072, China
[c] School of Computer Science and Technology, Nantong University, Nantong 226019, China
[d] School of Computer Science, Northwestern Polytechnical University, Xi'an 710072, China

ARTICLE INFO

Article history:
Received 11 September 2019
Revised 22 October 2019
Accepted 1 November 2019
Available online 2 November 2019

Keywords:
Security bug report prediction
Voting classification
Dataset construction
Common vulnerabilities and exposures

ABSTRACT

Identifying SBRs (security bug reports) is crucial for eliminating security issues during software development. Machine learning are promising ways for SBR prediction. However, the effectiveness of the state-of-the-art machine learning models depend on high-quality datasets, while gathering large-scale datasets are expensive and tedious. To solve this issue, we propose an automated data labeling approach based on iterative voting classification. It starts with a small group of ground-truth traing samples, which can be labeled with the help of authoritative vulnerability records hosted in CVE (Common Vulnerabilities and Exposures). The accuracy of the prediction model is improved with an iterative voting strategy. By using this approach, we label over 80k bug reports from OpenStack and 40k bug reports from Chromium. The correctness of these labels are then manually reviewed by three experienced security testing members. Finally, we construct a large-scale SBR dataset with 191 SBRs and 88,472 NSBRs (non-security bug reports) from OpenStack; and improve the quality of existing SBR dataset Chromium by identifying 64 new SBRs from previously labeled NSBRs and filtering out 173 noise bug reports from this dataset. These share datasets as well as the proposed dataset construction method help to promote research progress in SBR prediction research domain.

1. Introduction

Bug reports (BRs) describe issues found during software development and maintenance. Among BRs, security bug reports (SBRs) describes security-critical vulnerabilities in software products. Recently, security bug report prediction has become an active research topic (Peters et al., 2019; Shu et al., 2019; Gegick et al., 2010; Behl et al., 2014; Zaman, S., Adams, B., and Hassan, 2011; Kamongi and Kavi, 2013; Anisetti, 2017) since security has become a huge challenge in the rapid development of cloud services, big data and artificial intelligence (Akhtar and Mian, 2018). Machine learning and natural language processing are dominant techniques utilized by these studies (Bao et al., 2019; Wan et al., 2018). However, due to the characteristics complexity and contextual semantic relevance of security vulnerabilities, the performance (e.g. F1-score) of previous studies are still far from satisfactory for the real-world applications. For example, in the recent work, when considering F1-score, Peter et al. Peters et al. (2019) only achieved 0.37 in the best cases. Deep learning has been proved to be effective for learning semantics in natural language processing and text mining (Lopez and Kalita, 2017; Ren et al., 2019) and is promising for improving the performance of duplicate bug retrieval (Deshmukh et al., 2017). However, deep learning is an extremely data-hungry technique, for which large-scale labeled samples are required to optimize millions of parameters in deep network models (Yu et al., 2016). To the best of our knowledge, currently there are only five available datasets gathered from open-source projects for SBR prediction studies. In particular, four datasets are small-scale datasets (i.e., these datasets have 1000 records[1] at most) and these datasets were shared by Ohira et.al Ohira et al. (2015). The remaining dataset is a large-scale dataset (i.e., this dataset has around 50k records) and this dataset was gathered from the Chromium project. This dataset came from the mining challenge in MSR 2011 conference (MSR2011, 2011) and was further cleaned by Peter et.al Peters et al. (2019). It is not hard to find that these four datasets do not contain enough records for training deep

* Corresponding author.
E-mail addresses: wuxiaoxue00@gmail.com (X. Wu), wzheng@nwpu.edu.cn (W. Zheng), xchencs@ntu.edu.cn (X. Chen), wangf@nwpu.edu.cn (D. Mu).

[1] Notice the record means the bug report in this paper.

https://doi.org/10.1016/j.jss.2019.110456
0164-1212/

Table 1
Three examples of SBRs (security bug reports) mislabeled as NSBRs (non-security bug report) in the dataset Chromium.

Issue ID	Description
Issue 2877	The Google chrome browser is vulnerable to window object based denial of service attack. The Google Chrome fails to sanitize a check when window.close() function is called. The window.close() function is called in a suppressed manner by default which makes it vulnerable to ...
Issue 4739	Google chrome is vulnerable to URI Obfuscation vulnerability. An attacker can easily perform malicious redirection by manipulating the browser functionality. The link can not be traversed properly in status address bar.This could facilitate the impersonation of legitimate web sites in order to steal sensitive information from unsuspecting users. The URI specified with @ character with or without NULL character causes the vulnerability. Proof of Concept: http://www.secniche.org/gcuri/index.html
Issue 873	Security: passwords saved by computer shown 5 MG.POUPARD I find it strange and dangerous that with options ; minor tweaks; show passwords I can see all my logins with passwords and the websites they're for. I can under...

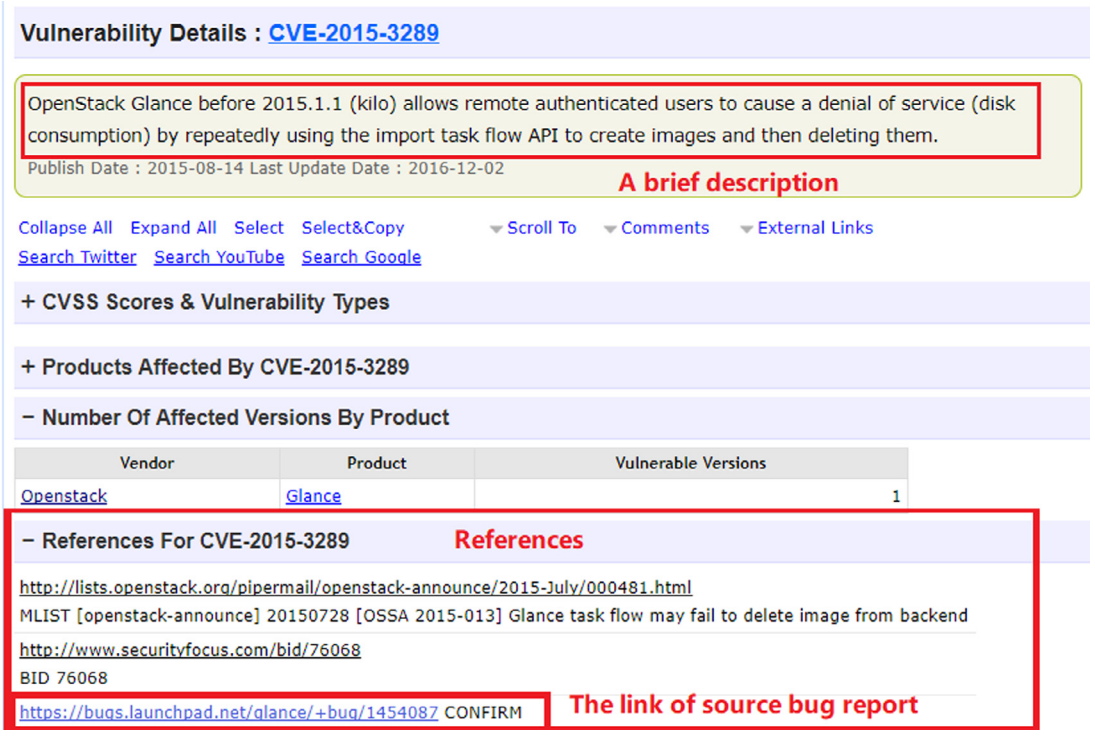

Fig. 1. An example of CVE detail page.

learning models. Moreover, after manually reviewing the records in the Chromium dataset, we find some security bug reports have been mislabeled as non-security bug reports (NSBRs) (Peters et al., 2019; Shu et al., 2019).

Table 1 shows some examples of mislabeled bug reports in the SBR dataset Chromium. The "Issue ID" is the identity in its bug tracking system JIRA (JIRA, 2018). The first bug report (Issue 2877) describes the issue, which would lead to denial of service attack because the function window.close() is called in a suppressed manner. The second bug report (Issue 4739) describes the issue, which would lead to sensitive information leakage. The third bug report (Issue 873) describes the issue, which is about unreasonable password protection strategy. All these bug reports are clearly related to software security and these bug reports should be labeled as security bug reports.

Dataset labeling is a tedious and time-consuming work (Ohira et al., 2015; Gunawi et al., 2014). How to obtain correct labels for constructing a large-scale dataset is a challenging task. To tackle this problem and enable the application of deep learning for SBR prediction, in this paper, we propose an automatic approach for labeling records in the large-scale dataset through voting classification. To ensure dataset quality, we first choose the bug reports related to authoritative security vulnerability data hosted in CVE (Common Vulnerabilities and Exposures) (CVE website, 2019) as the initial positive samples (i.e., security bug reports), which could guarantee its ground truth, since CVE is a dictionary of common names (i.e., CVE Identifiers) for publicly known information security vulnerabilities.

The CVE entries, which are assigned by CVE numbering authorities from around the world, provide a baseline for tool evaluation, and enable data exchange for cybersecurity automation (CVE website, 2019). Up to now, there are more than 100k real industry vulnerabilities hosted in the CVE website (Glanz et al., 2015), and most of these vulnerabilities are gathered from open-source projects, which entails us to get detailed information of these vulnerabilities. Fig. 1 presents the detail page of an example CVE record in project OpenStack (the CVE number is CVE-2015-3289). It describes the vulnerability briefly and gives "CVSS Score and Vulnerability Types", "Affected products", "Number Of Affected Versions by Product", and "References for this CVE Record". However, these information are far from enough for analyzing a software vulnerability (e.g., how to reproduce the issue ? what is the root cause of the issue?). Luckly, the "References" part of CVE detail page lists the link of this CVE record related sources, from which we can find the link of the source bug report (we marked it with red line) is listed as well. Therefore, we can go to the page of the source bug report with this link and get more information of the bug. The detail page of the source bug report of CVE-2015-3289 is shown in Fig. 2. Information like "Title" (a brief description of the bug), "Reporter" (the member who submit the bug), "Affects"

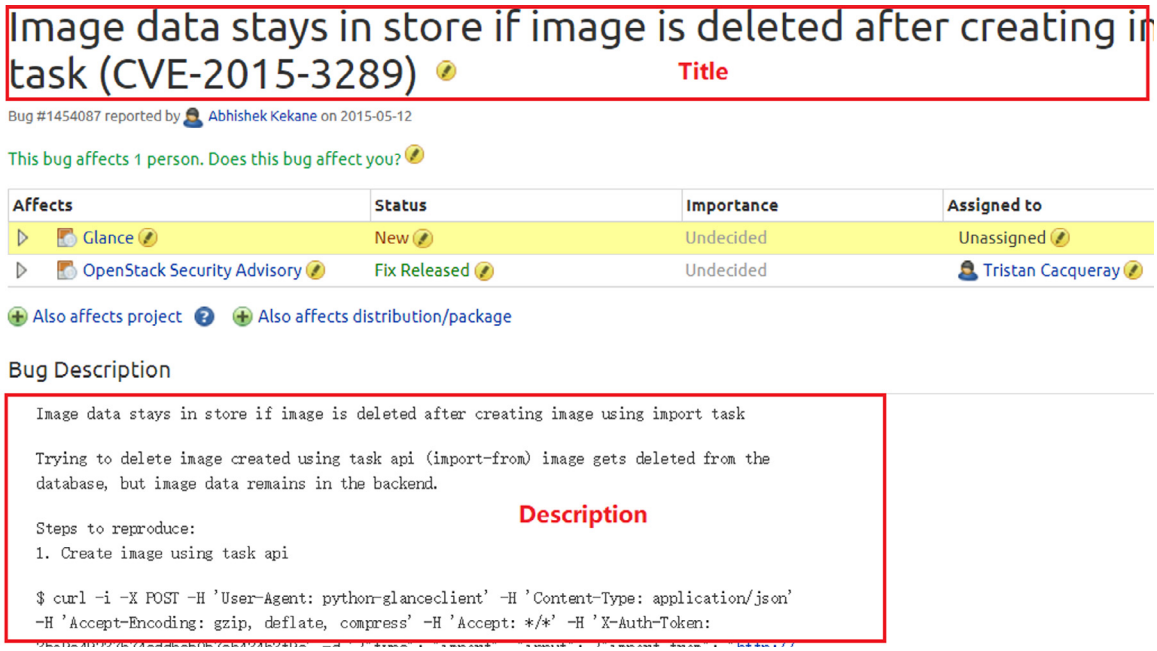

Fig. 2. The source bug report of CVE-2015-3289 (the link of the bug is: https://bugs.launchpad.net/glance/+bug/1454087).

(the affected products), "Status" (reflects the current step of the bug), "Importance" (the importance of the bug), etc. are listed in the page of source bug report. In particular, the field "Bug Description"[2] always provides detail information of the bug, such as the "Step to Reproduce", "Observed Behavior", and "Expected Behavior", which are highly important for developers when triaging and fixing bugs (Oscar Chaparro, 2017).

The goal of this paper is to design an automatic approach for constructing a large-scale dataset for SBR prediction by utilizing CVE entries and its related bug reports from open-source projects. We first evaluate the effectiveness of our approach by comparing it with each of the voting classifiers on two existing SBR datasets (i.e., Derby and Chromium). The results show that our approach can consistently outperform every single classifier with 0.9968, 0.5846, and 0.6786 in terms of accuracy, F1-score, and precision performance measures, respectively. Then we construct a large-scale dataset with over 80k records from the project OpenStack, and further improves the quality of the dataset Chromium by identifying 64 SBRs from its originally labeled NSBRs, and removing 173 noise records.

In summary, this paper makes the following contributions:

- To the best of our knowledge, we are the first to provide an automatic approach for large-scale SBR prediction dataset construction by using only a small group of initial labeled samples.
- We construct a large-scale (i.e., over 80k records) dataset from the "Openstack" project for SBR prediction.
- We improve the quality of the Chromium dataset by identifying and removing noises generated during the process of SBR labeling.
- We share the scripts of our dataset construction approach as well as our gathered OpenStack dataset and the clearned Chromium dataset[3].

The rest of this paper is organized as follows. Section 2 presents the background and related work of our study. Section 3 illustrates the design of our approach. Section 4 details the experimental setup. Section 5 reports the evaluation results. Section 6 concludes this study.

2. Background and related work

Nowadays, security has become a key hinder of its development in fields, such as cloud computing (Anisetti, 2017), smart contract (Akhtar and Mian, 2018).

2.1. Trends of software security

As the world's largest and most authoritative vulnerability repository, changes in CVE data can reflect the trends of exposed software vulnerabilities (Pham and Dang, 2018; Glanz et al., 2015; Chang et al., 2011; Martin, 2003). In the first year (i.e., 1999), there are 894 vulnerabilities exposed by CVE; and by the end of 2018, the number has reached 110599, which is 124 times that of 1999. Fig. 3 shows the number of new exposed vulnerabilities each year from 2009 to 2018. According to this figure, at least 4k new vulnerabilities are exposed each year from 2009, in particular, there is a sharp increase in 2017 and 2018, with more than 30k new records added in these two years, which accounts for 40% of the total of 2009 to 2018 and 27% of the total of 1999 to 2018. The change of CVE data reveals an increasing trend of software security risk.

2.2. Security bug report prediction

A lot of vulnerability prediction approaches have been proposed to identify security issues of software because most vulnerabilities involve exploitable weaknesses introduced through badly written codes (Piao, 2018). Several text mining based approaches have been proposed to identify SBRs from large-scale records in the bug repository (Gegick et al., 2010; Peters et al., 2019; Shu et al., 2019; Goseva-popstojanova et al., 2018). In the earliest study, Gegick et al. Gegick et al. (2010) proposed an approach to identify security bug reports based on keyword mining and conducted an empirical study on the datasets gathered from real industrial projects in Cisco. Later, Wijayasekara et al. Wijayasekara D (2012) conducted an analysis of exposed vulnerabilities in projects Linux

[2] This example only shows the previous part of the "Bug Description" because the text message of "Bug Description" is too long.

[3] https://github.com/wuxiaoxue/cve-assisted.

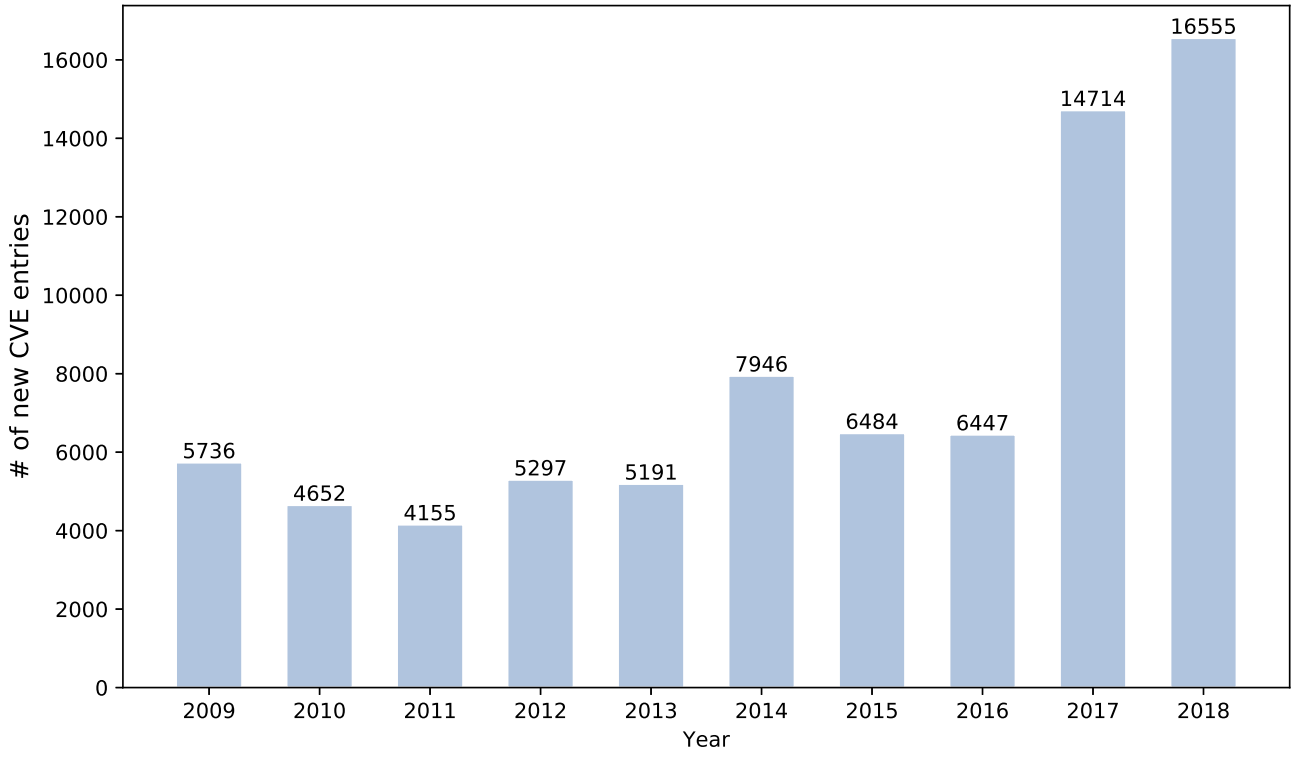

Fig. 3. New created records of CVE over 2009 to 2018. There has been a sharp increase in 2017 and continues to rise in 2018.

kernel and MYSQL, and elaborated the difficulties of mining bug databases for discovery of potential hidden impact vulnerabilities. After that, they proposed a vulnerability identification approach by extracting information from short and long description (the short description indicates the "Summary" or "Title" of a bug; the long description is the "Description" of a bug) of bug databases.

Recently, Peter et al. Peters et al. (2019) presented a noise filtering approach named FARSEC, which is a framework for filtering NSBRs with security related keywords. It first identifies a list of security-related keywords from security bug reports with TF-IDF (Term Frequency - Inverse Document Frequency), and then it assigns a score for each of these keywords according to its frequency in both SBR and NSBR groups. Here the NSBRs with scores as high as those SBRs will be removed from the dataset. The effectiveness of FARSEC is evaluated by five classical classification models on four small-scale datasets shared by the previous study (Ohira et al., 2015) and a large-scale dataset Chromium. Moreover, this approach mitigates the class imbalanced issue and improves the performance of prediction results in most cases on these five datasets. Rui et al. improved the recall of FARSEC by applying a hyperparameter optimization approach SMOTUNED (Shu et al., 2019), which is an improved version of SMOTE and proposed by Agrawal et al. Agrawal (2018). They applied SMOTUNED to both the parameters of classifiers and pre-processing method of the datasets. Final results showed that optimizing the hyperparameters in the pre-processing method is more useful than optimizing the hyperparameters in the classifiers.

These studies mainly identify security-related keywords in bug reports as well as features such as frequency, which are then used to train classification models. However, these studies are suffering from a high false positive rate. For example, FARSEC (Peters et al., 2019) with the combinations of a filter and a classifier that produced the best G-score had recall values in the range from 47.6% to 66.7%, probability of false alarm from 3.0% to 41.8%, and G-score from 53.8% to 71.9% across the five datasets.

Instead of developing new SBR prediction approaches, this paper proposes an automated sample labeling approach for constructing large-scale high-quality datasets to enable the application of new machine models (e.g. deep learning) for SBR prediction in the future.

2.3. Usage Scenario

3. The proposed approach

The framework of our approach is shown in Fig. 4. This framework includes three phases: data preparation phase, iterative voting classification phase, and a manual review and analysis phase. Amongst these three phases, phase II is an automatic process and includes the core algorithm of our approach. However, phase I and phase III are also important for the quality of the final constructed dataset. Phase I is the pre-condition of phase II and the quality of the output of phase I decides the accuracy of the classification results in phase II directly. Phase III is the final step of guarantying the quality of the constructed dataset. Phase I and phase III are two manual processes.

3.1. Phase I - Data preparation

The inputs of phase I includes CVE entries and the unlabeled bug reports of the source project. The output is the initially labeled dataset B_l and the remaining unlabeled source bug reports, we denote it with B_u. The main task of phase I is to generate a small-scale ground-truth labeled dataset B_l based on the two inputs.

3.1.1. Repositories of bug reports

CVE entries: CVE (Common Vulnerabilities and Exposures) (CVE website, 2019) is a publicly available vulnerability database hosted by MITER MITER's CVE. It is a part of the Security Content Automation Protocol (SCAP) SCAP website and it is used

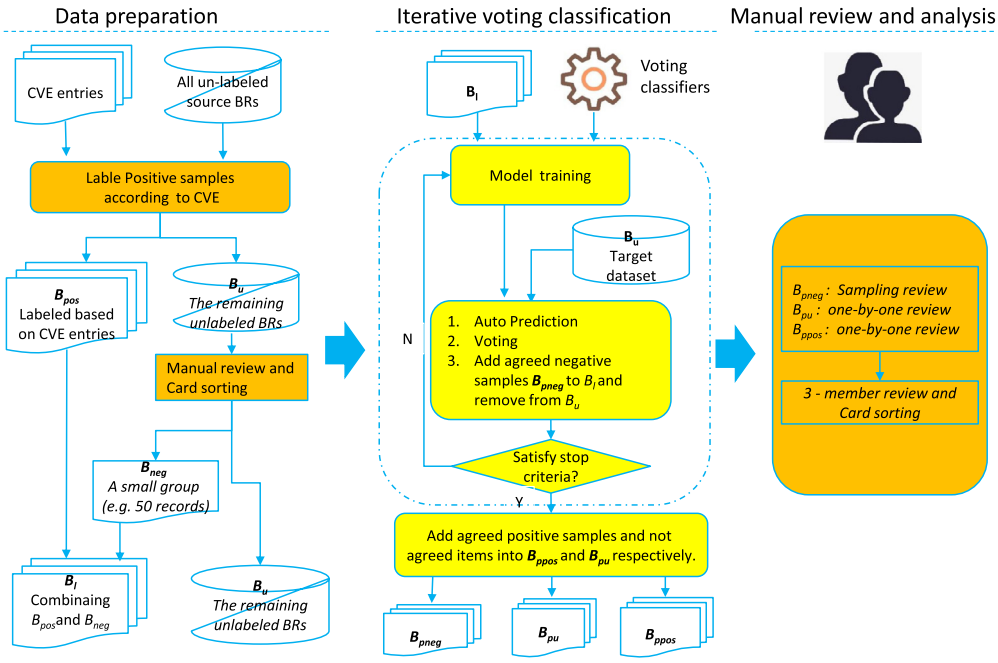

Fig. 4. The framework of our dataset construction approach.

to assign identifiers to publicly known vulnerabilities found in numerous IT products from around the world. CVE provides basic information about the vulnerabilities. More detailed information (e.g. affected products, versions) of the vulnerabilities are described in CVE detail website (CVE detail website, 2019). In particular, for open-source projects, the "Reference" part of CVE detail page lists the link(s) of related source bug reports (i.e., the bug report hosted in the bug tracking systems like JIRA, etc.), which bridges CVE entries and source bug reports. Therefore CVE entries provide a great convenience for the study of this paper.

Source bug reports: Many open-source projects (e.g. OpenStack, Chromium, Apache projects) use open-source bug tracking systems (e.g. JIRA JIRA (2018), LaunchPad LaunchPad OpenStack (2018), Bugzilla) for bug management. In order to distinguish from CVE data, we refer to the bug reports hosted in bug tracking systems as source bug reports. Source bug reports always provide detailed information (e.g. Status, Description, Comments) of bugs. In particular, the "Description" usually describes the input, actual output, reproduce steps and even exception log information of the bug. This informative text is important for bug report analysis and has been widely utilized by previous studies (Peters et al., 2019; Yang et al., 2017; Gegick et al., 2010; Thomas Zimmermann, 2010; Shu et al., 2019; Behl et al., 2014; Goseva-popstojanova et al., 2018).

3.1.2. Initial data labeling

The key idea of our approach is to label the large-scale unknown bug reports iteratively based on a small-group of labeled dataset and several classification models. Any label errors in the initial dataset will be amplified in the final result due to error propagation (Li et al., 2017). Thus, the correctness of initial labeled dataset has a great impact on the final results. To ensure the ground truth of labels for the initial labeled dataset B_l, we take 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), we select this record as a positive sample (i.e., SBR). Actually, in the repositories of source bug reports, if a source bug report is related to a CVE entry, the CVE identifier would be included in the "Title" (just as shown in Fig. 2 of Section I) or "References" of the source bug report. Therefore, we can identify CVE related source bug reports with automated approach (e.g., keywords mapping) to accelerate the initial data labeling process. The identified CVE-related source bug reports are marked as positive samples of the initial labeled dataset, and denoted as B_l. Meanwhile, we select a small group (e.g. 50 records) of negative samples (i.e., NSBRs) with high confidence via strategy like "card sorting" (Chen et al., 2018a; Fleiss and L., 1971), which is a widely applied approach for generating categories.

Table 2
Agreement level mapped with Fleiss Kappa (Fleiss and L., 1971).

Interpretation	Range of Fleiss Kappa value K_p
Poor	$K_p < 0$
Slight	$0.01 \leq K_p \leq .20$
Fair	$0.20 < K_p \leq .40$
Moderate	$0.40 < K_p \leq .60$
Substantial	$0.60 < K_p \leq .80$
Almost perfect	$0.80 < K_p \leq 1$

> **Card sorting** (Fleiss and L., 1971):
> *For the records that need to be reviewed, we create card for each of them, and classify each record into the group of SBR or NSBR. Each record is classified by at least three members independently.*

In this paper, we recruit three experienced security testing members as annotators to conduct the manual review task. To measure the agreement among the three annotators, we calculate the Fleiss Kappa value (Chen et al., 2018a; Bao et al., 2017) of their label results. The agreement level and corresponding range of Kappa value is shown in Table 2.

The prepared positive sample set B_{pos} and negative sample set B_{neg} are then combined together and denoted as B_l, which will be applied as the initial training set in phase II.

3.2. Phase II - Iterative voting classification

The classical method of distinguishing SBR from NSBR is to classify BRs using machine learning-based classification algorithm (e.g., Logistic Regression, Multinomial Naive Bayes, Multilayer Perceptron). Instead of using a single classification algorithm, this paper uses an iterative voting classification approach. The idea behind the voting classification is to combine conceptually different machine learning classifiers and uses a voting strategy to make the final decision. There are two widely applied voting strategies: hard voting (majority vote) and soft voting (the average predicted probabilities). The predicted class label of the hard voting strategy for a particular sample is the class label that represents the majority (mode) of the class labels predicted by each individual classifier; In contrast to the hard voting, the soft voting returns the class label as argmax of the sum of predicted probabilities, and the final result is the class with the highest average probability.

In this paper, we introduce an iterative voting classification approach to maximize the correctness of the prediction results. Instead of using the two commonly applied voting strategies, our voting approach makes the final decision of a sample label only all the voting classifiers give the same result (i.e., SBR or NSBR). Furthermore, we iteratively reduce the unlabeled items by joining the voted negative samples into training set, and retrain the models to predict the remaining unlabeled records. We ignore the voted positive samples during the iteration process because the proportion of positive samples in the initial training set B_l is much higher than in real production (e.g., for the five SBR datasets applied in Peters et al. (2019), the proportion of SBR ranges from 0.5% to 9.0%), which would lead to a very high false positive rate.

Algorithm 1 describes the process of our iterative voting classification approach. It takes the initial labeled set B_l, voting classifiers cl1, cl2, cl3 and the unlabeled target dataset B_u as inputs. Note that the initially labeled dataset B_l is expected to be ground-truth. The output of the algorithm is the prediction result of the target set B_u, it is separated into three sets B_{ppos}, B_{pneg}, B_{pu} according to the voting results. It begins with training the three classifiers with the initial training set B_l (line 3), and predict the target set B_u with each of the three classifiers (line 4), for each bug report in B_u, if the prediction results of the three classifiers are consistently negative (0, NSBR), it will be added into the initial training set B_l and the final negative set B_{pneg} (line 7 and 8), and removed from the target set B_u (line 9). The process then goes to the next iteration with the new increased training set B_l and the decreased set B_u. This iterative process continues until a termination criterion, which we will discuss in the next paragraph, is met. Once the iteration stops, the remaining records in B_u will be split into two parts according to the prediction results of the last iteration: the items with consistently positive results (1, SBR) of the three classifiers will be added into the set B_{ppos} (lines 14–15), indicating these items are predicted as positive by voting; and the others will be added into the subset B_{pu} (line 18), indicating the three classifiers cannot reach the same result for these records.

Algorithm 1: Iterative Voting Classification.

Input: Initial labeled training set B_l; voting classifiers $clf1, clf2, clf3$, Unlabeled target set B_u
Output: Predict results of B_u as three subsets: B_{ppos}, B_{pneg}, B_{pu}.

```
1  begin
2    Initialize B_ppos, B_pneg, B_pu to null ;
3    while !isMatchStopCriteria do
4      fit classifiers clf1, clf2, clf3 with B_l ;
5      predict B_u with each of clf1, clf2, clf3 ;
6      foreach b ∈ B_u do
7        if clf1(b) == 0 and clf2(b) == 0 and clf3(b) == 0 then
8          B_l ← B_l ∪ b ;
9          B_pneg ← B_pneg ∪ b ;
10         B_u ← B_u \ b ;
11       end
12     end
13   end
14   foreach b ∈ B_u do
15     if clf1(b) == 1 and clf2(b) == 1 and clf3(b) == 1 then
16       B_ppos ← B_ppos ∪ b ;
17     end
18     else
19       B_pu ← B_pu ∪ b
20     end
21   end
22   return B_pneg, B_ppos, B_pu.
23 end
```

3.2.1. The termination criteria.

The iteration process of Algorithm 1 is to identify negative samples through the voting strategy. An intuitive stop criterion of the iteration is to terminate while there is no bug report is consistently voted as negative in the current iteration. This is a late stop criterion because the iteration process would continue as long as there is a sample that is uniformly voted as a NSBR in the current iteration. In other words, the iteration stops only if the iterative process can no longer reduce the number of unlabeled samples through identifying consistently voted NSBRs. This is a late stop condition that tries to reduce the number of target data by exploring more confident NSBRs in each iteration. It is useful for minimizing the number of remaining unlabeled samples, therefore, we set it as default stop criterion in our approach. However, this stopping criterion is more suitable for mature projects that have accumulated many CVE records (or labeled SBRs). But, for projects that only have a small number of labeled SBRs, this stopping criterion may result in the loss of potential positive samples (SBRs). To alleviate this issue, we set another hard stop criterion in the algorithm, that is, when the number of remaining unlabeled samples (i.e., the number of B_u) is less than a given threshold, the iteration stops. In summary, there are two stop criteria used in our approach.

Termination criteria:
- *Criterion I: No negative samples found according to voting results in a new iteration;*
- *Criterion II: The threshold for the number of remaining unlabeled bug reports is reached.*

The iteration process stops when either of these two criteria is met.

3.2.2. Text mining techniques.

Text mining is the mostly used technique for bug report analysis. Therefore, this paper uses text mining approach to analyze the “Description” of bug reports for SBR classification. Feature extraction and dimensionality reduction are two basic steps of text mining. Feature extraction is used for extracting feature vectors of each bug report from text information (column “Description”).

In this study, we use the "CountVectorizer" provided by scikit-learn Scikit-Learn for extracting features from "Description" of bug reports. It converts the text of bug report "Description" to a matrix of token counts and produces a sparse representation of the counts. Stop words like "a", "the", "and", which are presumed to be uninformative in representing the content of a text, are removed to avoid them being constructed as signal for prediction.

Dimensionality reduction has been widely applied for maintaining important features, it reduces the number of random variables under consideration by obtaining a set of principal variables Dimensionality reduction wikipedia. It can be divided into two categories: *feature selection* and *feature extraction*. The purpose of dimensionality reduction is to remove features with a low variance to improve estimators' accuracy scores or to boost their performance on very high-dimensional datasets.

Suppose we are given a data matrix A whose columns are grouped into p clusters. Given a term-document matrix, the problem is to find a transformation that maps each document vector in the m dimensional space to a vector in the l dimensional space for some $l < m$. For this, either the dimension reducing transformation $G^T \in R^{l \times m}$ is computed explicitly or the problem is formulated as a rank reducing approximation where the given matrix A is to be decomposed into two matrices B and Y. That is,

$$A \approx BY \tag{1}$$

where $B \in R^{m \times l}$ with $rank(B) = l$ and $Y \in R^{l \times n}$ with $rank(Y) = l$.

In this paper, we make use of SelectFromModel SelectFromModel coupled with LinearSVC LinearSVC introduction in scikit-learn to evaluate feature importance and select the most relevant features for SBR identification. SelectFromModel selects features based on importance weights, it is a meta-transformer that can be used along with any estimator that has a *coef* or *feature_importances* attribute, which denotes the weights assigned to the features. The features are considered unimportant and removed if the corresponding *coef* or *feature_importances* values are below the provided threshold value. Apart from specifying the threshold numerically, there are built-in heuristics for finding a threshold using a string argument. Available heuristics are "mean", "median" and float multiples of these like "0.1*mean". Sparse estimator *linearSVC* is implemented in terms of liblinear (Fan et al., 2008), which is more flexible in choice of penalties and loss functions rather than other approaches such as libsvm (Chang and Lin, 2011). The parameter *penalty* of linearSVC in scikit-learn package Scikit-Learn specifies the norm used in the penalization to avoid over-fitting and its default value is "L2" regularization. Regularization is a very important technique in machine learning to prevent overfitting. It adds a regularization term in order to prevent the coefficients and can avoid overfit problem. "L1" and "L2" are two classical regularization functions (Ng, 2004). L1 regularization adds an L1 penalty equals to the absolute value of the magnitude of coefficients, while L2 regularization adds an L2 penalty equal to the square of the magnitude of coefficients. In this paper, we use "L1" because *linearSVC* with "L1" regularization is superior to linearSVC with "L2" regularization when dealing with many irrelevant features (Lindorfer et al., 2015).

3.3. Phase III - Manual review and analysis

The goal of this phase is to verify the prediction results of our approach. Similar to labeling initial negative samples, we also use "Card sorting" strategy for verifying the prediction results of our approach. Considering the trade-off between the quality of the final dataset and manual review cost, we take different solutions to select samples for manual review.

Table 3
Mapping of z-score and desired confidence level.

Desired confidence level	z-score
80%	1.28
85%	1.44
90%	1.65
95%	1.96
99%	2.58

Results verification solution:
- *One-by-one review: reviewing each record of the set.*
- *Sampling review: selecting specific number of records to review based on statistical sampling theory* (Junk, 1999).

In this paper, our recommend solution for results verification is: review all positive samples (P_{ppos}) and uncertain samples (P_{pu}) with "one-by-one" solution; review negative sample set (P_{pneg}) with "Sampling review" solution. We give this recommendation for three reasons: (a) positive samples are rare and each record matters for the quality of the final dataset; (b) the number of positive set and uncertain set are always not very large, which makes it possible to conduct "one-by-one" review; (c) negative samples account for the majority of the prediction results, therefore, "sampling review" is a feasible solution.

In order to get a set of records that are representative of the predicted negative samples for manual review, we use a publicly available sample size calculator CL and CI to determin how many negative samples need to be reviewed. Confidence level and margin of error are two key terms for sampling, the sample size can be calculated by:

$$samplesize = \frac{\frac{z^2 * p(1-p)}{e^2}}{1 + (\frac{z^2 * p(1-p)}{e^2 N})} \tag{2}$$

where N is the population size, which is the number of predicted negative samples in this paper; e is the margin of error; z is the z-score, which is the number of standard deviations a given proportion is away from the mean. There is a mapping between the value of z-score and desired confidence level, as shown in Table 3. When the sample size is determined, we randomly select the number from prediction result B_{pneg}.

4. Experimental setup

4.1. Voting classifiers

Based on latest studies (Peters et al., 2019; Shu et al., 2019; Tyo, 2018) that focus on SBR prediction with supervised machine learning, we select Logistic Regression, Multinomial Naive Bayes, Multilayer Perceptron because these classification algorithms can achieve better performance in previous studies. We use implementations of these classification algorithms in the Python machine learning package scikit-learn Scikit-Learn.

- Logistic Regression (LR): LR solves the classification problem with a similar idea to regression. The goal is to find the best fitting parameters of a nonlinear function *Sigmoid*. By establishing the cost function, the optimal model parameters are solved iteratively by means of the optimization method. It is generally appropriate when the dependent variable is dichotomous.
- Multinomial Naive Bayes (MNB): MNB is similar to Naive Bayes. It implements the naive Bayes algorithm for multinomially

Table 4
Characteristics of Existing SBR datasets for performance evaluation.

Dataset	# of total	# of SBR	# of NSBR	Description
Derby	1000	88	912	A relational database management system. The time range of data is 2004.9 - 2014.9
Chromium	41,940	192	41,748	Web browser called Chrome. The time range of data is 2008.8 - 2010.6

Table 5
Distribution of datasets for performance evaluation experiments.

Project	Training data (Initial labeled)		Testing data (Target data)	
	Positive	Negative	Positive	Negative
Derby	44	50	44	906
Chromium	96	50	96	41,698

Table 6
Distribution of source bug reports for our dataset construction.

Project	Initial labeled		Target data
	Positive	Negative	
OpenStack	172	50	88,568
Chromium	192	50	41,698

distributed data. It considers the conditional probability of each dimension feature separately, and then combines these probabilities to make a classification prediction of the feature vector in which it is located. It is suitable for classification with discrete features (e.g., word counts for the text classification).

- Multilayer Perceptron (MLP): MLP is a feedforward neural network with one or more layers between an input layer and an output layer and trained with a backpropagation learning algorithm. All parameters of the MLP are the connection weights and offsets between the layers.

4.2. Datasets

The experiment involves two groups of datasets. One is the existing SBR datasets used for evaluating the performance of our proposed iterative voting classification algorithm; the other is the source data with which we are going to construct the SBR datasets.

Group I: performance evaluation datasets. Before applying our proposed approach to real dataset construction, we first conduct a performance evaluation on two off-the-shelf SBR datasets: Derby and Chromium. The dataset "Derby" is one of the four datasets created by Ohira et al. Ohira et al. (2015) through manually reviewing 4000 bug reports of four open source projects Ambari, Camel, Derby and Wicket. They randomly select 1000 issues from each project and review them by graduate students and faculty members to mark labels for the high-impact bugs (e.g., security, performance, blocking), which directly affect user experience and satisfaction with software products. These datasets have been widely used for bug report analysis Yang et al. (2017); Peters et al. (2019); Shu et al. (2019). In this paper, we only choose Derby from the four datasets because the number of SBRs in the other three datasets is too few (≤ 32) to fit a machine learning model. Another selected dataset is Chromium. It is originally provided by MSR (Mining software repository) challenge conference 2010 with 41,940 bug reports and then further cleaned by Peter et al. Peters et al. (2019) via filtering the records which show a 404 not found error or require a unsername and password to gain access. Table 4 shows characteristic information of these two datasets.

Similar to the work of Peter et al. Peters et al. (2019), we split the positive samples of each project into two parts according to the dates the bug reports were submitted. The first part is used for training while the second part is used for testing. To simulate the process of our dataset construction approach, we only use 50 records as the initial labeled negative samples (i.e., negative samples in training data) for both "Derby" and "Chromium". Finally, the distribution of the training data and the testing data can be found in Table 5.

Group II: source data for SBR dataset construction. By using our proposed approach, we aim to improve the label correctness of existing SBR dataset Chromium and construct a new SBR dataset for OpenStack project. We choose Chromium because it is the only off-the-shelf large-scale SBR dataset but its label correctness needs to be improved, as examples shown in Table 1. We choose OpenStack as our target project for SBR dataset construction for three reasons: (a) it is one of the most popular cloud management system, the constructed datasets could benefit not only Openstack team but also general cloud service providers, who use OpenStack to construct their cloud services; (b) it consists of several different components (i.e., Nova, Neutron, Swift), which would increase the diversity of the final dataset (Jia et al., 2015; Xiang et al., 2016; Musavi et al., 2016; Anisetti, 2017); (c) our previous work (Zheng et al., 2019) has conducted an comprehensive review of around 50k OpenStack bug reports, which provides the basic data and experience for constructing OpenStack SBR datasets.

The distribution of source data for SBR datasets distribution is shown in Table 6, the detail of them will be elaborated in subsection 4.6 and 4.7.

4.3. Termination criteria setting

In our experiments, we use Criterion I (i.e., No negative samples found according to voting results in a new iteration) as the stop condition for both datasets OpenStack and Chromium because both of them have a good data condition to obtain most SBRs as initial positive samples. In particular, (1) Openstack already has a lot of CVE entries, which can be used to mark the initial positive samples. (2) Chromium dataset is obtained from the work of Peter et al. Peters et al. (2019) and the initial version of this dataset has 192 positive data records. We re-construct it just because there are some SBRs, which are miss-labeled as NSBRs in the initial version (as discussed in Section 1) and we try to identify these miss-labeled records as many as possible by using our approach.

4.4. Evaluation metrics

For each bug report, the classification results could have four possible outcomes:

- True Positive (TP): an SBR is predicted as SBR;
- False Negative (FN): an SBR is predicted as NSBR;
- True Negative (TN): an NSBR is predicted as NSBR;
- False Positive (FP): an NSBR is predicted as SBR.

Based on these outcomes, the recall, precision, F1-score and accuracy evaluation metrics for SBR prediction can be defined as below.

Recall: Recall indicates the proportion of security bug reports that are correctly predicted.

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

Table 7
Magnitude of Cliff's delta (Romano, 2006).

Magnitude	Range of Cliff's delta
Negligible	$\|d\| < 0.147$
Small	$0.147 \leq \|d\| < 0.33$
Medium	$0.33 \leq \|d\| < 0.474$
Large	$0.474 \leq \|d\|$

Precision: Precision indicates the proportion of correctly classified SBRs among SBRs and NSBRs that have been classified by the model to be SBRs. Precision measures the fraction of actual SBRs in the predicted SBRs.

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

F1-score: F1-score is the harmonic mean that combines both precision and recall.

$$F1 - score = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{5}$$

Accuracy: Accuracy denotes the proportion of real true in prediction results, also called success rate.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6}$$

Statistical analysis method: to further assess whether the performance of our approach is statistically significant than the other methods, we use the Wilcoxon rank-sum test (Romano, 2006) to analyze the statistical significance of the difference between our ovting approach and each of the three classifiers. The Wilcoxon rank-sum tests is often described as a non-parametric test of the null hypothesis since it does not assume known distributions, it does not deal with parameters. For the performance values of two methods compared, the null hypothesis is that there exists no significant difference between the two methods. The term significance level (α) is used to refer to a pre-chosen probability and the term *p-value* is used to indicate a probability calculated after a given study. If the p-value that results from Wilcoxon test is less than significance level, the null hypothesis is rejected. That is, the difference between the two methods is identified as statistically significant. Conventionally, the significance level 0.05, 0.01 and 0.001 are used. Most authors refer to statistically significant as p-value $<$0.05 and statistically highly significant as p-value $<$0.001 (Fan et al., 2018). We further calculate the *effect size* by comparing with each of the single classifiers in terms of F1-score and Accuracy to measure its practical significance. We use Cliff's delta (Yu et al., 2019; Fan et al., 2018) that quantifies the amount of difference between two non-parametric variables beyond the *p-value* interpretation. The Cliff's delta can be computed by:

$$d = 2W/mn - 1 \tag{7}$$

where W is the statistic of the Wilcoxon rank-sum test, and m and n are the sizes of two compared distributions. Here $W = R - n(n+1)/2$, where R is the sum of the rank in the sample and n is the sample size. The magnitude and the corresponding range of Cliff's delta, which is suggested by Romano et al. Romano (2006), is shown in Table 7.

4.5. Manual review strategy

Manual review is needed to produce initial training dataset and verify the correctness of final prediction results. Overall, we use the "Card sorting" approach for all records that selected for review. To further ensure the correctness and fairness of the review results, we recruit three experienced security testing members: one is from the "Information security community" of our school, the other two are from the security testing team of a company that constructs their cloud platform based on OpenStack infrastructure. The members participating in our data review have more than three years of penetration testing experience and gained many awards in domestic and international security competitions - all these could assure that they have a deep understanding of the characteristics of software vulnerabilities and could make resposible review results. In order to make the review result as close to ground-truth as possible, if the review results of the three members are different for a record, the three members discuss together until getting a consistent result.

For the final prediction results of the two key studies in this paper, we select records for review with the suggested sampling strategy of our framework.

- For the voted positive set B_{ppos}, we use the "one-by-one" review strategy, which means each record of B_{ppos} will be manually reviewed and discussed by three experienced members to ensure its label correctness.
- The voted negative set B_{pneg} takes the "sampling review" strategy. The size of voted negative set B_{pneg} would be very large since negative samples are always the majority in bug report datasets. We calculate the number of need to be reviewed records with Eq. 2, the confidence level and margin of error are set as 95% and 5, respectively.
- For the voted ambiguous set B_{pu}, we again use the "one-by-one" review strategy for two reasons: (a) identifying more positive samples can improve the capability of final SBR datasets, while the ambiguous set has a high probability of containing positive samples; (b) the number of ambiguous set B_{pu} is acceptable for manual review (less than 1000 records in our experiments).

4.6. Key study I: Openstack dataset construction

4.6.1. Introduction for openstack

OpenStack is a representative cloud computing management system, which has been widely applied in the industry (e.g. eBay, Progressive insurance, SBAB bank, and Volkswagen financial services) (OpenStack User Survey Report, 2018; Garcia and Benson, 2016; Baset et al., 2013). Users can customize these components to build a cloud computing platform according to their business requirements.

4.6.2. Bug reports of openstack.

Records in LaunchPad: We collected raw data of OpenStack bug reports from version control system Launchpad (LaunchPad OpenStack, 2018) with conditions shown in Table 8. With these conditions, there are totally 89,607 records collected. Because the column "Description" is the most used field for machine learning-based bug report analysis, to create high-quality benchmarks, which will be used by other researchers, we exclude records with empty "Description" information from the raw data. Finally, we get **88790** records as source data from OpenStack to construct the SBR dataset.

Records in CVE: The first exposed security vulnerability of OpenStack is recorded in CVE website in 2011. Since then, there are more than 10 vulnerabilities exposed each year, and by the end of 2018, the number of exposed vulnerabilities of OpenStack has increased to **186** in total. Vulnerabilities are further classified into different types in CVE system, the distribution of OpenStack vulnerabilities are described in Fig. 5. There are eight different vulnerability types involved in OpenStack, of which the number of Dos (denial of service) is the most, accounting for 29.4%, followed by "Gain information" and "Bypass Something", accounting for 24.2 and 20.9, respectively. The number of the other five types are relatively small, ranging from 2% to 10%. The diversity of vulnerability

Table 8
Conditions of OpenStack data collection from LaunchPad.

Condition Name	Condition Value	Note
Time Range	$\leq$ 2018.12.31	The project started its development since 2010.
Status	Fix Committed, Fix Released	Status describes the current state of a bug. There are totally 12 different statuses in LaunchPad. We choose the two that indicate the complete status of a bug.
Importance	Undecided, Critical, High, Medium, Low	Importance expresses the order in which bugs should be fixed. OpenStack bugs have six importance levels: Undecided, Critical, High, Medium, Low and Wishlist. We do not include wishlist because this level is more to be an improvement than a bug.

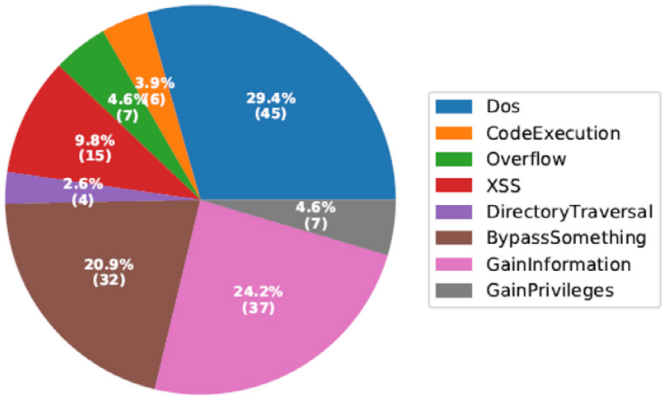

Fig. 5. Vulnerability types and their proportion of OpenStack recorded in CVE website.

types in CVE entries, which are taken as initial positive samples for training the initial model, can effectively improve the generalization ability of the classification model.

4.6.3. Initial labeled dataset.

To conduct supervised machine learning, a group of labeled samples is required. SBR prediction can be described as a binary-classification problem, similar to existing dataset Chromium, we use label '1' to indicate a positive sample (SBR - security bug report) and '0' to indicate a negative sample (NSBR - non-security bug report).

- Positive samples: we obtain the positive samples of OpenStack as the initial dataset based on its CVE records. As stated on the CVE website, each CVE entry represents a publicly known cybersecurity vulnerability (CVE website, 2019). Then, according to the vulnerability management process of OpenStack vmp, the security issue of OpenStack will be connected with a CVE number applied from MITER's CVE (CVE website, 2019) to ensure its traceability. Therefore, if a bug report of OpenStack is related to a CVE number, we can conclude it is a security bug report and mark with label '1'. As a result, we get 172 positive samples, which is a bit smaller than the number of CVE entries (186) because 14 CVE records do not connect with any LaunchPad bug ID.
- Negative samples: In our experiment, we only add 50 negative samples to the initial labeled dataset. We randomly select samples from the unlabeled dataset and review them by three members, to ensure the quality of the initial dataset, a sample that gains different labels by the three members will not be selected.

4.7. Key study II: Chromium dataset validation

To the best of knowledge, the Chromium dataset is the only large-scale SBR dataset gathered from the open-source project for SBR analysis. As introduced in Section 1, it is clear there are some SBRs mislabeled as NSBRs in the previous dataset Chromium. We try to improve its quality by our approach. Instead of re-labeling the positive samples with the help of CVE, we use the labeled 192 SBRs in the dataset as initial positive samples because we assume these SBRs are marked correctly. In order to improve the accuracy of NSBRs with our approach, we treat all their labeled NSBRs as uncertain, and randomly select 50 NSBRs with high confidence through "Card sorting" approach with three experienced annotators. The remaining samples are set as the target unlabeled data.

Table 9
Performance results of each classification algorithm in dataset Derby and Chromium. ("Vote" is short for our proposed approach "Vote classification").

Project	Algorithm	Accuracy	F1-score	Precision	Recall
Derby	Vote	**0.9703**	**0.5846**	**0.6786**	0.5135
	MNB	0.3329	0.1229	0.0659	0.9091
	LR	0.1451	0.1023	0.0539	**1**
	MLP	0.1921	0.0999	0.0527	0.9545
Chromium	Vote	**0.9968**	**0.2873**	**0.2455**	0.3462
	MNB	0.5751	0.0086	0.0043	0.9121
	LR	0.5981	0.0099	0.005	0.9231
	MLP	0.5092	0.0076	0.0038	**0.967**

5. Result analysis

5.1. Performance of the voting model

We first evaluate the performance of our approach by comparing its effectiveness with each of the three voting classifiers (LR, MNB, MLP). We applying each algorithm on the datasets "Derby" and "Chromium" (as shown in Table 5). The results in terms of accuracy, F1-score, precision, and recall are shown in Table 9. The best value of each indicator is bolded. The results show our approach consistently outperforms each of the three classifiers in terms of accuracy, F1-score, and precision. In particular, the accuracy reaches 0.9968, which is a very comparative and positive result for dataset building. The three classifiers win our approach at recall with the cost of dramatic decrease on precision and accuracy, which matter much for dataset construction. For example, classifier "MLP" gets value 0.5092 for Accuracy on Chromium, which means there are more than 20k NSBRs are mislabeled as SBRs. The major reason that contributes to this extreme high recall but low accuracy of each single classifier is the differency of sample distribution between training set and testing set. As shown in Table 5, the proportion of positive samples in training set is much higher than that in testing set. However, our approach uses an iterative process to alleviate the proportion (i.e., positive: negative) difference between training set and testing set. It only takes the consistently voted as negative samples in each iteration, and moves these negative samples into training set to training the classifiers in the next iteration. This iterative process continues until the stop criterion is met, as shown in Algorithm 1.

In order to evaluate the statistical significance of our approach, we perform each classification algorithm in "Derby" and "Chromium" 30 times with shuffling the data at the beginning, and caculate p-value and effect size with Wilcoxon rank-sum test

Table 10
P-value and Effect Size Magnitude of comparing Accuracy, F1-score of our approach with each classifier. The Effect Size Magnitude is mapped with value of Cliff's delta according to Table 7.

Dataset	Algorithm	Accuarcy		F1-score	
		P-value	Effect Size Magnitude	P-value	Effect Size Magnitude
Derby	MNB	<0.001	Large	<0.001	Large
	LR	<0.001	Large	<0.001	Large
	MLP	<0.001	Large	<0.001	Large
Chromium	MNB	<0.001	Large	<0.001	Large
	LR	<0.001	Large	<0.001	Large
	MLP	<0.001	Large	<0.001	Large

Table 11
Statistics of the voted result for target data OpenStack and Chromium.

Project	Voted Positive	Proportion (%)	Voted Negative	Proportion (%)	Remain Uncertain	Proportion (%)
OpenStack	129	0.15	88,146	99.27	515	0.58
Chromium	143	0.34	41,246	98.63	430	1.03

and Cliff's delta in terms of accuracy and F1-score, respectively. The range of p-value and the "magnitude of effect size" is shown in Table 10. According to the results, our iterative voting classification approach significantly outperforms each of the three classifiers.

5.2. Statistics of prediction result

After applying our approach on the target projects "OpenStack" and "Chromium" with the initial data distribution shown in Table 6, the three outputs of each project is shown as Table 11. As shown in the table, the most of the bug reports are voted as negative samples, accounts for 99.27% and 98.63% for OpenStack and Chromium, respectively; the positive samples only takes up 0.15% and 0.34% with number 129 and 143, respectively; the uncertain samples, for which the three classifiers cannot reach the same prediction results, are only 0.58% and 1.03%.

5.3. Label precision

We verify the precision of identifying SBRs and NSBRs of our approach through manually review the prediciton results. Following the manual review strategies described in subsection 4.5, all the "voted positive" and "uncertain" samples of the two projects are selected to manual review. For the "voted negative" samples, with the values 95% and 5 for parameters "confidence level" and "margin of error" respectively, 383 out of 88,146 voted negative samples are selected for OpenStack, and 381 out of 41,246 voted negative samples are selected for Chromium. In a summary, there are totally 1027 and 945 need to review samples for OpenStack and Chromium, respectively. Each of these samples is reviewed by three experienced members by following the "Card sorting" approach. The review results could be "SBR", "NSBR", or "Uncertain", in which the items with label "Uncertain" indicate these records are ambiguous even with manual review for some reasons (e.g., limited information).

According to the review results of the three annotators, OpenStack has 109 records (out of 1027) with inconsistent results, and Chromium has 102 records (out of 945) with inconsistent results. We further explore the factors that contribute to these inconsistent results, and find that the quality of the bug reports itself is the main factor leading to inconsistency of the review results. Let us take Bug 1,364,401 of OpenStack as an example: two annotators mark it as "Uncertain" while the other one mark it as "SBR". The "Description" of this bug report is as below:

> **Description of Issue 1364401:**
> Currently configuration of token backend in keystone looks like:
> [token]
> driver=keystone.token.backends.memcache.
> Token caching=true
> Which means that tokens are stored in memcached and then cached in (presumably the same) memcached. This doesn't make sense, "caching=true" line should be removed.

From the "Description" of issue 1364401, it is asking for not caching the token. We know the "token" of system is always sensitive information, thus, caching it or not might affect the security of the system, but it is not inevitable. Therefore, it is hard to label it confidently. We calculate the Fleiss Kappa value of the review results to evaluate the agreement level of the three annotators on the two datasets. The results show the overall Kappa value of OpenStack and Chromium is 0.81 and 0.74. which indicate "Almost perfect" and "Substantial", respectively. For each of the inconsistent records, the three annotators discuss together until reach a consistent result. Finally, the distribution of final review results are shown in Table 12.

For OpenStack, there are totally 191 samples reviewed as SBRs from all the three outputs (i.e., voted positive, voted negative, uncertain), 724 items are reviewed as NSBRs, and 112 samples are assigned to "uncertain"; for project Chromium, the number of SBRs, NSBRs and uncertain of manual review results is 64, 685 and 174, respectively. A key reason of marking "uncertain" is the "Description" of the sample is only with limited text information, which is insufficient for judging a bug report is security related or not. For example, the description of "Issue 12,187 " in Chromium is: " *Crash in base_unittests involving SystemMonitor observer and NowSingleton 1 person starred this issue and may be notified of changes. phajdan... @chromium.org.*". Only with this text information, it is hard to determine whether this crash is security related or not even for a senior security engineer. Therefore, we mark such items as "uncertain" and recommend to treat such records as noise data and remove them from the final dataset.

Based on our manual review results, the precision of our approach for predicting SBRs and NSBRs in OpenStack and Chromium is calculated and shown in Table 13. This result indicates our ap-

Table 12
Statistics of the review results with "Card sorting" approach. The reviewed records from predicted negative results for OpenStack is **383** and Chromium is **381**, which are calculated with statistical sampling rule with CL(confidence level) = 0.95, and the Margin of Error is 5.

Project	Voted Positive			Voted Negative			Uncertain		
	SBR	NSBR	Uncertain	SBR	NSBR	Uncertain	SBR	NSBR	Uncertain
OpenStack	82	15	32	3	376	4	106	333	76
Chromium	32	41	59	1	377	3	31	267	111

Table 13
Precision for predicting both SBRs and NSBRs of our approach on Projects OpenStack and Chromium.

Project	Precision of Positive	Precision of Negative
OpenStack	0.64	0.98
Chromium	0.24	0.99

proach is extremely responsible for identifying NSBRs, which always take the vast majority of bug report databases. The precision of identifying SBRs is moderate. Especially, the precision of SBR identification for OpenStack is much better than Chromium. A possible reason contribute to this is the data quality - the text information of "Description" in OpenStack bug reports is much longer and complete than that of Chromium bug reports.

6. Discussion

In this section, we discuss the threats to the validity of our proposed approach, as well as the availibility of the constructed datasets.

6.1. Key features for SBR prediciton

According to our manual observation of the SBRs (both initial labeled SBRs and final predicted SBRs), there are indeed some frequently occurred words that possibly related to security in their text of "Description", we summarize some of them as below.

SBR related keywords
"leak", "leakage", "memory", "attack", "security", "https", "password", "risk", "javascript", "access", "ssl", "vulnerability", "vulnerabilities", "attacker", "directory", "token", "PKI", "port", "scan", "bypass", "authorization", "admin"

However, we can not directly classify a bug report as an SBR if any of these security-related keywords occcur in its "Description". For example, a bug report is described like "*... I failed to login the system while I submitted the correct username and password...* ". It is obvious this is just a functional issue instead of an SBR. But, if a bug report is described like "*... my password is displayed on the login page in plain text...* ", then it should be an SBR because it can lead to the disclosure of secret information. Therefore, a more intelligent feature extraction method that understands context semantics or associations between words will be more helpful to improve the accuracy of SBR prediciton. For example, Ren et.al.Ren et al. (2019) constructed a computational structure of CNN (Convolutional Neural Network) to identify key phrases and patterns from text and we can further investigate the effectiveness of their proposed method in the future.

6.2. Avalability of the constructed datasets

As discussed in Section 1, identifying SBRs is of great importance for guaranting the security of software system. We have analyzed the label correctness of the datasets we constructed in Section 5 and gotten very responsible results. The two large-scale datasets we constructed in this paper are out-of-the-box and could be directly applied for SBR related tasks (e.g., SBR prediction). Especially, neural network-based machine learning approaches, which have become dominant for data mining tasks (e.g., bug report prediction (Deshmukh et al., 2017)), are enabled. Furthermore, with our approach, researchers could construct more SBR datasets from open-source projects only with moderate manual effort.

6.3. Threats to validity

6.3.1. Threats to internal validity.

The first threat to internal validity of our approach refers to errors in the implementation code. For the design and implementation code of our approach, we organized a formal code review meeting, and all authors of this paper participated in the meeting to ensure the quality of our implementation. Another threat to internal validity is the selected voters (i.e., classification algorithms). A large number of classification algorithms (e.g., NB, SVM, k-nearest neighbor) have been proposed and any single study can only use a small subset of existing classification algorithms. In our study, we choose three most reliable classifiers based on existing literature (Peters et al., 2019; Yang et al., 2017; Fan et al., 2018) focused on bug report analysis as SBR prediction is a sub-sector of bug report analysis.

6.3.2. Threats to external validity.

The primary threat to external validity for our approach is the quality of the initially labeled dataset as data quality matters for all machine learning-based algorithms (Bao et al., 2019; Shepperd et al., 2013; Chen et al., 2018b), which is also the focus of this paper. However, as suggested by our approach, the positive samples of the initial training set could be labeled according to CVE entries, which are authoritative vulnerabilities that are sufficient to guarantee its correctness. For initial negative samples, we suggest to ask experienced testers to label these data. Luckly, our method requires only a small group (e.g., 50 records in our experiments) of initial labeled negative samples. The second threat to external validity is whether our proposed approach has the generalization ability to the projects in other domains. In our approach, we propose an automated data labeling approach based on iterative voting strategy, which is domain independent. Therefore, we can apply our approach to other projects in different domains.

7. Conclusion and future work

This paper proposes an approach of constructing large-scale datasets for machine learning-based SBR prediction. We first prepare a ground-truth training set with CVE related bug reports as

positive samples and a small group of carefully selected high-quality negative samples, then use an iterative voting classification algorithm to identify NSBRs and SBRs from large-scale unlabeled bug reports. We have constructed an initial version of dataset for OpenStack including around 80k bug reports; furthermore, we identified 64 new SBRs with our approach and manual review effort from original labeled NSBRs in the existing dataset Chromium to improve its label correctness. In the future, we first want to investigate whether we can use our constructed datasets to perform cross-project SBR prediction when the target projects do not have enough training data. The challenge of this issue is to propose effective modeling methods to alleviate the data distribution difference between different projects. Therefore, we will resort to transfer learning (Pan and Yang, 2009) and propose effective modeling methods. Secondly, we will continue to improve our dataset construction approach by incorporating other methods (such as feature selection, deep learning) to further improve the quality of our constructed datasets. we will apply the datasets we constructed to SBR analysis related tasks (e.g., cross-project SBR prediction, deep learning-based SBR prediction); on the other hand, we will continue to improve the accuracy of our data construction approach by introducing solutions like better feature extraction approaches, deep learning-based classification models, etc. to enable more high-quality datasets construction.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We would like thank Fenyu Liu and two security test engineers of a company that does not want to be named. They gave great support to the work of manual bug report review in this paper. With their help, we corrected the FP (False Positive) and FN (False Negative) records predicted by our approach and gave more reasonable results of the OpenStack and Chromium SBR datasets.

References

Agrawal Amritanshu, M.T., 2018. Is “better data” better than “better data miners”? In: Proceedings of the 40th International Conference on Software Engineering. IEEE, pp. 1050–1061.
Akhtar, N., Mian, A., 2018. Threat of adversarial attacks on deep learning in computer vision: a survey. IEEE Access 6, 14410–14430.
Anisetti Claudio Agostino Ardagna, M.E.D.e.a., 2017. A security benchmark for openstack. In: Proceedings of the 10th International Conference on Cloud Computing, Honolulu, CA, USA, 25–30 June 2017. IEEE, pp. 294–301. doi:10.1109/CLOUD.2017.45.
Bao, L., Xia, X., Lo, D., Murphy, G.C., 2019. A large scale study of long-time contributor prediction for github projects. IEEE Trans. Softw. Eng..
Bao, L., Xing, Z., Xia, X., Lo, D., Hassan, A.E., 2017. Inference of development activities from interaction with uninstrumented applications. Emp. Softw. Eng. (1) 1–39.
Baset, S.A., Tang, C., Tak, B.-C., Wang, L., 2013. Dissecting open source cloud evolution: An openstack case study. Proceeding of the 5th USENIX Workshop on Hot Topics in Cloud Computing, SAN JOSE, CA, 25–26 June 2013.
Behl, D., Handa, S., Arora, A., 2014. A bug mining tool to identify and analyze security bugs using naive bayes and tf-idf. In: Proceedings of International Conference on Optimization, Reliabilty, and Information Technology, Faridabad, India, 6–8 Feb. 2014. IEEE doi:10.1109/ICROIT.2014.6798341.
Chang, C.-C., Lin, C.-J., 2011. Libsvm: a library for support vector machines. ACM Trans. Intell. Syst. Technol. 2 (3), 27.
Chang, Y.Y., Zavarsky, P., Ruhl, R., Lindskog, D., 2011. Trend analysis of the CVE for software vulnerability management. In: Proceedings of the Third International Conference on Privacy, Security, Risk and Trust and Social Computing, Boston, MA, USA, 9–11 October 2011. IEEE, pp. 1290–1293. doi:10.1109/PASSAT/SocialCom.2011.184.
Chen, Q., Bao, L., Li, L., Xia, X., Cai, L., 2018. Categorizing and predicting invalid vulnerabilities on common vulnerabilities and exposures. In: 2018 25th Asia-Pacific Software Engineering Conference (APSEC). IEEE, pp. 345–354.
Chen, X., Zhao, Y., Wang, Q., Yuan, Z., 2018. Multi: multi-objective effort-aware just-in-time software defect prediction. Inf. Softw. Technol. 93, 1–13.
Cl and ci. https://www.surveysystem.com/sscalc.htm.
Cve detail, 2019. https://www.cvedetails.com/.
Cve website, 2019. https://cve.mitre.org/.
Cve website. http://cve.mitre.org/.
Deshmukh, J., Podder, S., Sengupta, S., Dubash, N., et al., 2017. Towards accurate duplicate bug retrieval using deep learning techniques. In: 2017 IEEE International conference on software maintenance and evolution (ICSME). IEEE, pp. 115–124.
Dr wikipedia. https://en.wikipedia.org/wiki/Dimensionality_reduction.
Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, 2008. Liblinear: a library for large linear classification. J. Mach. Learn. Res. 9 (9), 1871–1874.
Fan, Y., Xia, X., Lo, D., Hassan, A.E., 2018. Chaff from the wheat: characterizing and determining valid bug reports. IEEE Trans. Softw. Eng. 1–30. Early Access
Fleiss, L., J., 1971. Measuring nominal scale agreement among many raters.. Psychol. Bull. 76 (5), 378–382.
Garcia, W., Benson, T., 2016. A first look at bugs in openstack. In: Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking, Irvine, California, USA, 12-12 December 2016. ACM, pp. 67–72. doi:10.1145/3010079.3010086.
Gegick, M., Rotella, P., Xie, T., 2010. Identifying security bug reports via text mining: An industrial case study. In: Proceedings of the 7th International Working Conference on Mining Software Repositories, Cape Town, South Africa, 2–3 May 2010. IEEE, pp. 11–20. doi:10.1109/MSR.2010.5463340.
Glanz, L., Schmidt, S., Wollny, S., Hermann, B., 2015. A vulnerability's lifetime: enhancing version information in CVE databases. In: Proceedings of the 15th International Conference on Knowledge Technologies and Data-driven Business, Graz, Austria, 21–23 October 2015, pp. 28:1–28:4. doi:10.1145/2809563.2809612.
Goseva-popstojanova, K., Tyo, J., Sizemore, B., 2018. Security Vulnerability Profiles of NASA Mission Software : Empirical Analysis of Security Related Bug Reports. In: Proceedings of the 28th International Symposium on Software Reliability Engineering, Toulouse, France, 23–26 October 2017. IEEE, pp. 1–11. doi:10.1109/ISSRE.2017.42.
Gunawi, H.S., Hao, M., Leesatapornwongsa, T., Patana-Anake, T., Do, T., Adityatama, J., Eliazar, K.J., Laksono, A., Lukman, J.F., Martin, V., 2014. What bugs live in the cloud? a study of 3000+ issues in cloud systems. In: Proceedings of the ACM Symposium on Cloud Computing Pages 1–14, Seattle, WA, USA, 03–05 November 2014. ACM doi:10.1145/2670979.2670986.
Jia, T., Li, Y., Yuan, X., Tang, H., Wu, Z., 2015. Characterizing and predicting bug assignment in openstack. In: Proceeding of the Second International Conference on Trustworthy Systems and Their Applications, Hualien, Taiwan, 8–9 July 2015. IEEE, pp. 16–23. doi:10.1109/TSA.2015.14.
Jira, 2018. https://www.atlassian.com/software/jira.
Junk, T., 1999. Confidence level computation for combining searches with small statistics. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 434 (2–3), 435–443.
Kamongi, P., Kavi, K., 2013. VULCAN: Vulnerability Assessment Framework for Cloud Computing. In: Proceedings of the 7th International Conference on Software Security and Reliability, Gaithersburg, MD, USA, 18–20 June 2013. IEEE doi:10.1109/SERE.2013.31.
OpenStack bug reports in LaunchPad, 2018. https://bugs.launchpad.net/openstack/.
Li, G., Hari, S.K.S., Sullivan, M., Tsai, T., Pattabiraman, K., Emer, J., Keckler, S.W., 2017. Understanding error propagation in deep learning neural network (dnn) accelerators and applications. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, p. 8.
Lindorfer, M., Neugschwandtner, M., Platzer, C., 2015. Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis. In: IEEE Computer Software and Applications Conference.. IEEE.
Linearsvc introduction in scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html.
Lopez, M.M., Kalita, J.K., 2017. Deep learning applied to nlp. CoRR abs/1703.03091.
Martin, R.A., 2003. Integrating your information security vulnerability management capabilities through industry standards (cve&oval). In: SMC'03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme-System Security and Assurance (Cat. No. 03CH37483), 2. IEEE, pp. 1528–1533.
Mining Software Repository Challenge, 2011. http://2011.msrconf.org/msr-challenge.html.
Musavi, P., Adams, B., Khomh, F., 2016. Experience report: An empirical study of api failures in openstack cloud environments. In: Proceedings of the 27th International Symposium on Software Reliability Engineering, Ottawa, ON, Canada, 23–27 October 2016. IEEE, pp. 424–434. doi:10.1109/ISSRE.2016.42.
Ng, A.Y., 2004. Feature selection, l1 vs. l2 regularization, and rotational invariance. In: Proceedings of the twenty-first international conference on Machine learning.
Ohira, M., Kashiwa, Y., Yamatani, Y., Yoshiyuki, H., Maeda, Y., Limsettho, N., Fujino, K., Hata, H., Ihara, A., Matsumoto, K., 2015. A dataset of high impact bugs: Manually-classified issue reports. In: Proceedings of the 12th Working Conference on Mining Software Repositories, Florence, Italy, 16–17 May 2015 doi:10.1109/MSR.2015.78.
Openstack user survey report, 2018. https://www.openstack.org/user-survey/2018-user-survey-report/.
Chaparro, O., Lu, J., Zampetti, F., Moreno, L., 2017. Detecting missing information in bug descriptions. In: Joint Meeting on Foundations of Software Engineering, pp. 396–407.
Pan, S.J., Yang, Q., 2009. A survey on transfer learning. IEEE Transactions on knowledge and data engineering 22 (10), 1345–1359.

Peters, F., Tun, T., Yu, Y., Nuseibeh, B., 2019. Text filtering and ranking for security bug report prediction. IEEE Trans. Softw. Eng. 45 (6), 615–631.
Pham, V., Dang, T., 2018. Cvexplorer: multidimensional visualization for common vulnerabilities and exposures. In: Proceedings of the International Conference on Big Data, Seattle, WA, USA, 10–13 December 2018. IEEE, pp. 1296–1301. doi:10.1109/BigData.2018.8622092.
Liu, B., Huo, W., Zhang, C., Li, W., Li, F., Piao, A., Zou, W., 2018. adiff: Cross-version binary code similarity detection with dnn. In: Proceeding of the International Conference on Automated Software Engineering. Corum, Montpellier, France, pp. 3–7.
Ren, X., Xing, Z., Xia, X., Lo, D., Wang, X., Grundy, J., 2019. Neural network-based detection of self-admitted technical debt: from performance to explainability. ACM Trans. Softw. Eng. Methodol. 28 (3), 15.
Romano, J., 2006. Appropriate statistics for ordinal level data: Should we really be using *t*-test and cohensd for evaluating group differences on the nsse and other surveys. Annual meeting of the Florida Association of Institutional Research..
Scap website. https://scap.nist.gov.
Scikit-learn. https://scikit-learn.org/.
Selectfrommodel. https://scikit-learn.org/stable/modules/classes.html.
Shepperd, M., Song, Q., Sun, Z., Mair, C., 2013. Data quality: some comments on the nasa software defect datasets. IEEE Trans. Softw. Eng. 39 (9), 1208–1215.
Shu, R., Xia, T., Williams, L., Menzies, T., 2019. Better Security Bug Report Classification via Hyperparameter Optimization. In: International Conference on Automated Software Engineering, 2019. IEEE/ACM, pp. 1–12. arXiv: 1905.06872.
Thomas Zimmermann Nachiappan Nagappan, L.A.W., 2010. Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista. In: Proceedings of the 3rd International Conference on Software Testing, Verification and Validation, Paris, France, April 7–9, 2010. IEEE, pp. 421–428. doi:10.1109/ICST.2010.32.
Tyo, J.P., 2018. Empirical Analysis and Automated Classification of Security Bug Reports. West Virginia University.
Openstack vulnerability management process, 2019. https://security.openstack.org/vmt-process.html.
Wan, Z., Xia, X., Hassan, A.E., Lo, D., Yin, J., Yang, X., 2018. Perceptions, expectations, and challenges in defect prediction. IEEE Trans. Softw. Eng..
Wijayasekara, D., Manic, M.W.J., 2012. Mining bug databases for unidentified software vulnerabilities. In: Proceedings of the 5th International Conference on Human System Interactions, erth, WA, Australia, 6–8 June 2012, pp. 89–96. doi:10.1109/HSI.2012.22.
Xiang, Y., Li, H., Wang, S., Chen, C.P., Xu, W., 2016. Debugging openstack problems using a state graph approach. In: Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems, Hong Kong, Hong Kong, 04–05 August 2016. ACM, p. 13. doi:10.1145/2967360.2967366.
Yang, X.-L., Lo, D., Xia, X., Huang, Q., Sun, J.-L., 2017. High-impact bug report identification with imbalanced learning strategies. J. Comput. Sci. Technol. 32 (1), 181–198.
Yu, F., Seff, A., Zhang, Y., Song, S., Funkhouser, T., Xiao, J., 2016. LSUN : Construction of a large-scale image dataset using deep learning with humans in the loop.
Yu, T., Wei, W., Xue, H., Hayes, J., 2019. Conpredictor: concurrency defect prediction in real-world applications. IEEE Trans. Softw. Eng. 45 (6), 558–575.
Zaman, S., Adams, B., Hassan, A.E., 2011. Security Versus Performance Bugs: A Case Study On Firefox. In: Proceedings of the Working Conference on Mining Software Repositories. IEEE, pp. 93–102.
Zheng, W., Feng, C., Yu, T., Yang, X., Wu, X., 2019. Towards understanding bugs in an open source cloud management stack: an empirical study of openstack software bugs. J. Syst. Softw. 151, 210–223.

Xiaoxue Wu is currently a Ph.D. Student with the Department of Cyberspace security, University of Northwestern Polytechnical. She received the B.E. degree in Information Management from University of Xi Dian, China, M.Sc. degree in Software Engineering from University of Northwestern Polytechnical, China. Her main research interests are security testing and big data analysis. She has published over 15 articles in journals, conferences, and book chapters.

Wei Zheng is currently an associate professor with the Department of Software and Microelectronics, University of Northwestern Polytechnical. He received the M.Sc degree in Computer Science Ph.D degree in Computer Software and Theory from University of Northwestern Polytechnical, China. His research focus is on cloud computing security software quality assurance, and multi-objective optimization. He is the author or coauthor of more than 40 research publications, which includes international journals of high impact or prestigious international conferences. Moreover, he has been a reviewer of several international journals and conferences.

Xiang Chen is currently an associate professor with the Department of Computer Science and Technology, Nan Tong University. He received the B.Sc. degree in the school of management from Xi'an Jiaotong University, China in 2002. Then he received the M.Sc., and Ph.D. degrees in computer software and theory from Nanjing University, China in 2008 and 2011 respectively.. He is interested in empirical software engineering, mining software repositories, software maintenance and software testing. In these areas, he has published over 60 papers in referred journals or conferences, such as IEEE Transactions on Software Engineering, Information and Software Technology, Journal of Systems and Software, ASE, ICSME and SANER.

Fang Wang is currently an associate professor with the Department of Computer Science, University of Northwestern Polytechnical. She received the M.Sc degree in Computer Science Ph.D degree in Computer Software and Theory from University of Northwestern Polytechnical, China. Her research focus is on Big data analysis and software testing. She is the author or coauthor of more than 30 research publications, which includes international journals of high impact or prestigious internaitonal conferences.

Dejun Mu is currently a professor with the Department of Cyberspace security, University of Northwestern Polytechnical, China. He received the M.Sc degree in Computer Science and Ph.D degree in Computer Software and Theory from University of Northwestern Polytechnical, China. His research focus is on cyber space security. He is the author or coauthor of more than 100 research publications, which includes international journals of high impact or prestigious internaitonal conferences. Moreover, he has been a reviewer of several international journals and conferences.