A Domain Knowledge-Guided Lightweight Approach for Security Bug Reports Prediction

Wei Zheng  
School of Software Engineering  
Northwestern Polytechnical University  
Xi’an, China  
wzheng@nwpu.edu.cn

Zheng Chen  
School of Software Engineering  
Northwestern Polytechnical University  
Xi’an, China  
www13152121137@163.com

Xiaoyue Wu  
College of Information Engineering  
Yangzhou University  
Yangzhou, China  
wuxiaoyue00@gmail.com

Weiqiang Fu  
School of Software Engineering  
Northwestern Polytechnical University  
Xi’an, China  
fuweiqiang00@gmail.com

Bowen Sun  
School of Software Engineering  
Northwestern Polytechnical University  
Xi’an, China  
bwsun 0611@163.com

Jingyuan Cheng  
School of Software Engineering  
Northwestern Polytechnical University  
Xi’an, China  
chengjingyuan000@163.com

Abstract—Security bug report (SBR) prediction has been increasingly investigated for eliminating security attack risks of software products. However, there is still much room for improving the performance of automatic SBR prediction. This work is inspired by the work of two recent studies proposed by Peters et al. and Wu et al., which are focused on SBR prediction and both published on the top tier journal TSE (Transactions on Software Engineering). The goal of this work is to improve the effectiveness of supervised machine learning-based SBR prediction with the help of software security domain knowledge. It first extracts software security domain knowledge from CWE (Common Weakness Enumeration) and CVE (Common Vulnerabilities Exposure), which are authoritative sources of software vulnerability. After that, the matrix of bug reports is generated based on the roots of security domain keywords. Large-scale experiments are conducted on a set of trustworthy datasets cleaned by Wu et al. The results show our domain knowledge-guided approach could improve the effectiveness of SBR prediction by 25% in terms of F1-score on average.

Keywords: software security, bug report prediction, domain knowledge, keywords root, keyBERT

I. INTRODUCTION

In software engineering, bug reports are submitted to bug tracking systems to record issues found in software products. To reduce security risks of software systems, security bug report (SBR) prediction has become an increasingly hot topic recently [1]–[7]. Most of these work use machine learning-based text mining approaches to achieve this since the major information of a bug report is described in the field Description with text format [1]. [8]–[11]. For example, Peters et al. [1] propose a security keywords-based data filtering approach named Farsec for SBR prediction. They extract security keywords from the Description content of the training set for each project and then generate matrices with these keywords for bug reports of each dataset. Shu et al. [12] follow the work of Peters et al. [1], and apply hyperparameter optimization approach to optimize key parameters of the classification algorithms. Wu et al. [2] conduct a comprehensive empirical study on the impact of dataset label correctness for SBR prediction.

However, the effectiveness of previous studies is still not ideal for the production application. As shown in the work of Peters et al. [1], the best value of G-measure is 0.71 with a relatively small value of F1-score (0.30). We further explored the reasons that lead to the poor effectiveness of their approach and found the poor quality of security keywords should be one important reason.

CWE (Common Weakness Enumeration) is the authoritative community of international system security [13], [14]. It defines different software vulnerability categories with the subcategories of CWE-699 (Software Development). The security keywords that Farsec applied are extracted from the training set of each dataset. Figure 1 shows the top 100 security keywords they extracted for Chromium. While looking at these words closer, many of them (e.g., the words highlighted in blue) seem to be security-irrelevant based on the vulnerability definition (i.e., the Title and Description) of CWEs.

This work is inspired by the work of Peters et al. [1], which is published on top tier journal TSE and focused on file, security, chrome, page, http, download, user, starred, person, notified, changes, may, see, url, site, bug, open, google, browser, like, windows, window, https, web, code, one, memory, firefox, function, tests, problem, seems, tab, also, version, use, would, using, view, used, make, users, chromium, crash, click, password, think, vulnerability, sure, browsers, link, attached, attacker, data, get, fix, const, content, something, safari, new, error, javascript, learning, malicious, please, could, risk, release, try, found, allow, expected, time, example, corruption, test, back, access, crashes, urls, int, without, know, versions, way, uses, cause, fail, want, system, still, files, arbitrary, html, details, ssl, need, loaded, might.
SBR prediction. This work aims to improve the effectiveness of machine learning-based SBR prediction with the help of software security domain knowledge. It first extracts software security domain knowledge from CWE [13] and CVE (Common Vulnerabilities Exposure) [15], which are authoritative sources of software vulnerability [14], [16], [17]. After that, the matrices of bug reports are generated based on the roots of domain knowledge keywords. Large-scale experiments are conducted on a set of trustworthy datasets cleaned by Wu et al. The results show: (1) security domain knowledge-guided approach performs well on SBR prediction with the best value 0.81 in terms of F1-score; (2) Both the security domain keywords of CWE and CVE improve the effectiveness of SBR prediction.

This paper makes the following contributions:

- To the best of our knowledge, we are the first to use software security domain knowledge of CWE and CVE to improve SBR prediction.
- We use the two most authoritative software vulnerability data repositories CWE and CVE, as the sources of domain knowledge extraction, which can guarantee the quality of domain knowledge we obtained.
- We conduct an intensive empirical study to evaluate the effectiveness and efficiency of our approach.

The remainder of this paper is organized as follows. Section II introduces the background of this study. Section III describes our experimental settings. Section V details our experimental results of each research question respectively. Section VI-B discuss the threats to the validity of our study. Section VII concludes the paper.

II. BACKGROUND AND RELATED WORK

A. Security bug report prediction

Many machine learning-based SBR prediction approaches have been proposed in recent years [1], [3], [4], [10], [12]. The earliest work is proposed by Gegick et al. [4]. They try to identify security bug reports based on keyword mining and conduct an empirical study on datasets of real-world projects in Cisco. It was found that their approach determined that reporters labelled SBRs as NSBRs in 78 percent of large Cisco software systems. After that, Wijayasekara et al. [5] conduct an analysis of exposed vulnerabilities in projects Linux kernel and MYSQL. They propose a vulnerability identification approach by extracting information from the Summary and Description of a bug report. First, segment description (i.e., the Title of a bug report) and long description are extracted from the bug report, and then feature vectors are generated by text mining. The classifier is then used to determine whether it is a normal bug (NSBR) or a vulnerability (SBR).

Recently, Peters et al. [1] present a noise filtering approach named FARSEC for SBR prediction. Before making predictions, the framework identifies and removes NSBRs that contain security keywords. Then five machine learning classification algorithms (i.e., Random Forest, Logistical Regression, Naive Bayes, K-nearest Neighbor, and Multiple Layer Perceptron) were used to classify SBRs from NSBRs. Experimental results show Farsec is effective for improving the Recall of SBR prediction. Be inspired by this, Jiang et al. [3] introduce a content-based noisy approach for SBR prediction named LTRWES. LTRWES uses ranking model BM25F and word embedding technology for SBR recognition. Later, LTRWES uses word embedding to turn the error report into vectors. Then machine learning algorithms are used to make predictions.

Shu et al. [12] follow the work of Peters et al., they apply hyperparametric optimization to classifiers to improve the work of Peters et al. For example, how many trees are used by the random forest classifier. At the same time, they also updated the data preprocessing method of Peters et al. by using SMOTE, a hybrid sampled data preprocessor, to alleviate the problem of class imbalance.

Wu et al. [2] conducts a comprehensive review of the label correctness of five publicly available SBR prediction datasets and found a lot of mislabeling in these datasets, which might mislead the research direction of SBR prediction. Their experiment showed that when using the clean data sets, even a simple text classification model outperforms the work of Peters et al. and Shu et a.

B. Work of Peters et al.

A recent work that inspired this study is proposed by Peters et al. [1]. To improve SBR prediction, Peters et al. [1] design a framework for filtering and ranking bug reports for reducing the noisy data from the training set. The process of Farsec contains three major steps: (1) Identifying security keywords and making data matrices. They first tokenize SBRs of a dataset to terms. Then, each term’s tf-idf (term frequency — inverse document frequency) value is calculated, and the top 100 terms with the highest tf-idf values are kept as security keywords. After that, they calculate the frequency of each security keyword in the Description of each bug report for both training set and testing set, and security-keywords matrices of the training set and testing set are generated, respectively. (2) Filtering NSBRs with security-related keywords. The purpose of filtering in Farsec is to remove NSBRs with security-related keywords. To achieve this, they designed seven different filters: farsec, farsescsq, farsectwo, clni, clnifarsec, clnifarsecssq, and clnifarsectwo. They apply each of these filters to the training set of each dataset. Therefore, seven new training sets are generated, and each of them is applied to fit the model independently. (3) Ranking bug reports. After predicting the SBRs, a list of ranked bug reports was generated based on ensemble learning. The actual SBRs in the prediction results appear closer to the top of the list.

Peters et al. show that the security-related keywords are of great importance to judge whether a bug report is security-related or not. This conclusion keeps in accordance with the work of Wu et al. [2].
C. Work of Wu et al.

Wu et al. [2] conduct a comprehensive review of the five publicly available SBR prediction datasets to correct the labels. To guarantee the correctness of their review result, they classify each SBR into at least one CWE category. There are more than 50 CWE categories are involved in their reviewed SBRs. They further group them from the 40 top-level CWE categories of software development. As a result, CWE-1218 (Memory buffer error), CWE-199 (Information management error), and CWE-465 (Pointer Issues) take the top three of the CWE categories with the largest number of SBRs.

They also summarize the patterns with the help of Chaparro et al. [18], [19]'s work. According to Chaparro et al. [18], the Description of a bug report should contain at least the observed behavior (OB) and the expected behavior (EB) in text. The most common patterns of OB and EB can be summarized as below:

- **OB**: ([subject]) [negative aux. verb] [verb] [complement]. Here, [negative aux. verb] ∈ {are not, can not, does not, did not, etc.}. For example, [Searching with a tab in the omnibar does not] [work with google main search...](from Chromium Issue 71).
- **EB**: [subject] should/shall (not) [complement]. For example, [It should not [need local administrative access to complete the install...](from Chromium Issue 119).

In the case of the identified SBRs of the five datasets, there are some security-related keywords appear in the [subject] and [verb]/[complement]. Furthermore, these keywords are also commonly used words in the definition (Name and/or Description) of CWE categories. This sheds light on the design of our security domain knowledge-guided SBR prediction approach.

### III. METHODOLOGY

Figure 2 shows the process of our domain knowledge-based SBR prediction approach. It includes three phases: first, we identify security domain keywords from the top 25 most dangerous CWE categories and CVE entries. After that, the keywords matrices are generated for each dataset by calculating their frequency in the description of each bug report; finally, each matrix is split into the training set and testing set for model fitting and performance evaluation.

A. Phase I: Domain keywords extraction.

1) **Sources of domain keywords**: The rarity of SBRs and the sparseness of security-related keywords in the text description of bug reports would limit the accuracy and comprehensiveness of the extracted security keywords. To improve the diversity of domain knowledge, we extract the security domain keywords from two authoritative data repositories.

**CWE top 25 most dangerous categories**: CWE is a community-developed list of common software weaknesses that might result in systems being vulnerable to attack if left unaddressed [13]. We use the top-25 most dangerous vulnerability types reported by CWE [20] in 2020 as our domain knowledge sources. These CWE top 25 most dangerous categories show the most widespread and critical programming errors that can lead to serious software vulnerabilities [21].

Table 1 lists the ID and Name of the 2020 top 25 most dangerous CWE categories ranked by their prevalence and severity. To conduct a deeper exploration of the security-related domain knowledge, we use the Name, Description and Extended Description of each CWE as sources of our common security domain keywords extraction. Figure 3 shows an example of the Description and Extended Description of CWE with CWE-416 (Use-After-Free). These fields describe each CWE concisely and provide us a high-quality source for security keywords extraction.

**Project-specific security domain knowledge from CVE.** In addition to the general security requirements shared among products, each software also has its specific security requirement. CVE entries are ground-truth vulnerabilities found from the production environment of each project. Therefore, in this study, we collect product-specific security domain knowledge from history CVE entries to augment the security keywords list extracted from CWE.

The target projects of this study include Chromium, Ambari, Camel, Derby, and Wicket, which are five open-source projects widely applied in SBR prediction [1]–[3], [12]. Therefore, we collect their CVE entries and extract project-specific security keywords from the Description of them.

2) **Keywords roots extraction method**: We develop our domain keywords roots extraction approach based on an open-source tool named keyBERT [22], [23]. KeyBERT is a method developed based on BERT (Bidirectional Encoder Representations from Transformers) [24]. It is a quick and easy method for creating keywords without training from scratch.

To improve the effectiveness of the approach, we optimize keyBERT by stemming the root of each word.

Algorithm 1 describes the process of our security domain keywords root extraction approach. The input of the algorithm includes (a) doc: the document (i.e., documents generated from CWE or CVE entries in this study) from which to extract keywords; (b) top_n: the number of keywords needs to return. The output is the roots of the top n keywords extracted for a document with their respective distances to the input document. It first vectorizes the doc with CountVectorizer() (lines 2 and 3) and gets the word list of the features as candidates (line 4). Then it constructs an embedding model for sentence transformation and encodes the doc and candidates (lines 5-8). The parameter verbose controls the verbosity of the process. After that, the cosine similarity, which measures the similarity between two vectors of inner product space, is used to find the most similar words to the document (lines 9-10). Finally, it uses the SnowballStemmer() of NLTK [25] to extract the root of each keyword (lines 11-16).

As described in the above subsection III-A1, we have four sub-sources for security domain keywords extraction, namely, the Name, Description, and Extended Description of the top 25 CWEs, and the CVEs of each project. We set top_n in Algorithm 1 as 100 for all the four sub-sources. Finally, the extracted keyword roots of CWEs are shown in Figure 4. The
number of keyword roots extracted from the Name is a bit less than 100 since the total words of the Names for the 25 CWE categories is less than 100 after removing stop words. When comes to extracting keyword roots from CVE entries, we collect the CVE entries from each project. To avoid overlap with the testing data of the target project, the Publish Date of the CVE entries we selected are out of the report time range of the dataset. The security keyword roots extracted from the CVE entries of four target projects are shown in Figure 5.

### B. Phase II: Domain keywords matrix generation

The bug reports are tokenized and given an integer id for each possible token (the white-space and punctuation are always used as token separators). The occurrences tokens in each bug report are counted as frequency; finally, TfidfVectorizer normalizes and weight with diminishing importance tokens that occur in most bug reports. Each token occurrence frequency (normalized or not) is treated as a feature. The vector of all the token frequencies for a given bug report is considered a multivariate sample.

The matrix generation involves creating data matrices from the root keywords. Each row represents a bug report in a dataset, and the columns represent the terms in the keywords root list. The process of matrix generation is described with Algorithms 2. The input includes root keywords of CWE ($K_{cwe}$) and CVE ($K_{cve}$), and the target bug report dataset ($B$). The output is the generated root keywords matrix ($B_{mx}$) target data. It still uses the Snowball Stemmer as stemmer (line 2) and the CountVectorizer of scikit-learn [26] as vectorizer (line 3). The root keywords (i.e., the roots of security domain

<table>
<thead>
<tr>
<th>Rank</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CWE-79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
</tr>
<tr>
<td>2</td>
<td>CWE-787</td>
<td>Out-of-bounds Write</td>
</tr>
<tr>
<td>3</td>
<td>CWE-20</td>
<td>Improper Input Validation</td>
</tr>
<tr>
<td>4</td>
<td>CWE-125</td>
<td>Out-of-bounds Read</td>
</tr>
<tr>
<td>5</td>
<td>CWE-119</td>
<td>Improper Restrictions of Operations within the Bounds of a Memory Buffer</td>
</tr>
<tr>
<td>6</td>
<td>CWE-89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
</tr>
<tr>
<td>7</td>
<td>CWE-200</td>
<td>Exposure of Sensitive Information to an Unauthorized Actor</td>
</tr>
<tr>
<td>8</td>
<td>CWE-416</td>
<td>Use After Free</td>
</tr>
<tr>
<td>9</td>
<td>CWE-352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>10</td>
<td>CWE-78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
</tr>
<tr>
<td>11</td>
<td>CWE-190</td>
<td>Integer Overflow or Wraparound</td>
</tr>
<tr>
<td>12</td>
<td>CWE-22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
</tr>
<tr>
<td>13</td>
<td>CWE-476</td>
<td>NULL Pointer Dereference</td>
</tr>
<tr>
<td>14</td>
<td>CWE-287</td>
<td>Improper Authentication</td>
</tr>
<tr>
<td>15</td>
<td>CWE-434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
</tr>
<tr>
<td>16</td>
<td>CWE-732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
</tr>
<tr>
<td>17</td>
<td>CWE-94</td>
<td>Improper Control of Generation of Code ('Code Injection')</td>
</tr>
<tr>
<td>18</td>
<td>CWE-522</td>
<td>Insufficiently Protected Credentials</td>
</tr>
<tr>
<td>19</td>
<td>CWE-611</td>
<td>Improper Restriction of XML External Entity Reference</td>
</tr>
<tr>
<td>20</td>
<td>CWE-798</td>
<td>Use of Hard-coded Credentials</td>
</tr>
<tr>
<td>21</td>
<td>CWE-502</td>
<td>Deserialization of Untrusted Data</td>
</tr>
<tr>
<td>22</td>
<td>CWE-269</td>
<td>Improper Privilege Management</td>
</tr>
<tr>
<td>23</td>
<td>CWE-400</td>
<td>Uncontrolled Resource Consumption</td>
</tr>
<tr>
<td>24</td>
<td>CWE-306</td>
<td>Missing Authentication for Critical Function</td>
</tr>
<tr>
<td>25</td>
<td>CWE-862</td>
<td>Missing Authorization</td>
</tr>
</tbody>
</table>
Algorithm 1: Process of keywords roots extraction with keyBERT.

Input: Document doc; top n keywords need to return top_n;

Output: rt_Keywords: the roots of the top n keywords extracted from document.

```
begin
  vectorizer ← CountVectorizer(vocabulary = keywords, min_df = 0, stop_words = frozenset());
  count ← vectorizer.fit([doc]);
  candidates ← count.get_feature_names();
  SentenceTransformerBackend(model).embedding_model←SentenceTransformer(model);
  self.model←SentenceTransformerBackend(model);
  doc_embedding←self.model.embedding_model.encode(doc, show_progress_bar=verbose);
  candidate_embedding←self.model.embedding_model.encode(candidates, show_progress_bar=verbose);
  distances ← cosine_similarity(doc_embedding, candidate_embeddings);
  keywords ← [(candidates[index], round(float(distances[0][index]), 4)) for index in
distances.argsort()[0][:-top_n][::-1];
  stemmer ← SnowballStemmer("english");
  rt_Keywords ← [];
  foreach kw ∈ keywords do
    rt_kw ← stemmer.stem(kw);
    rt_Keywords.append(rt_kw);
  end
  return rt_Keywords.
end
```

Algorithm 2: Keyword root matrix generation process for each bug report dataset.

Input: Keyword roots from CVE K_cwe, keyword roots from CVE entries of the project K_cwe;
Bug report set B

Output: Keyword roots matrix B_ml for a bug report dataset B.

```
begin
  stemmer ← SnowballStemmer("english");
  vectorizer ← CountVectorizer(vocabulary = keywords, min_df = 0, stop_words = frozenset());
  keywords ← K_cwe, K_cve or K_cwe ∪ K_cve;
  words ← clean and tokenize B;
  words ← Remove stopwords from words;
  rt_words ← [];
  foreach w ∈ words do
    rw ← stemmer.stem(w);
    rt_words.append(rw);
  end
  X ← vectorizer.fit_transform(rt_words);
  result ← pandas.DataFrame(data=X.toarray(),
columns=vectorizer.get_feature_names());
  B_ml ← pandas.concat([df_id, result], axis=1).
end
```

Fig. 3. An example of CWE Description and Extended Description.

CWE-416: Use After Free

<table>
<thead>
<tr>
<th>Description</th>
<th>The use of previously-freed memory can have any number of adverse consequences, ranging from the corruption of valid data to the execution of arbitrary code, depending on the instance and timing of the flaw. The simplest way data corruption may occur involves the system's reuse of the freed memory. Use-after-free errors have two common and sometimes overlapping causes: Confusion over which part of the program is responsible for freeing the memory. In this scenario, the memory in question is allocated to another pointer validly at some point after it has been freed. The original pointer to the freed memory is used again and points to somewhere within the new allocation. As the data is changed, it corrupts the validly used memory; this induces undefined behavior in the process. If the newly allocated data chances to hold a class, in C++ for example, various function pointers may be scattered within the heap data. If one of these function pointers is overwritten with an address to valid shellcode, execution of arbitrary code can be achieved.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extended Description</td>
<td>Referencing memory after it has been freed can cause a program to crash, use unexpected values, or execute code.</td>
</tr>
</tbody>
</table>

keywords) could be of one sub-source or a combination of different sub-sources (i.e., the Name of CWE, the Description of CWE, the Extended Description, and the Description of CVE entries of a project) (line 4). It first cleans and splits the target into words (line 5) and then removes stopwords (line 6). After that, it extracts the root of each word (lines 8-11). Finally, It generates a security keywords matrix for root words of each bug report with vectorizer (lines 12-14).

C. Phase III: Model Fitting and Performance Evaluation

In Phase II, the matrix for each dataset is generated with the roots of security domain keywords by calculating word frequency. Then, it is easy to fit a classification model with the labelled records of the matrix, and then predict the unlabelled target records or evaluate the performance of the classification
model with a testing set. Since SBR prediction is formulated as a binary classification problem \cite{1,2,12}, various classification approaches (e.g., traditional classification models like Random Forest, Naïve Bayes; and deep neural network-based approaches) can be applied for its scenario.

\section*{IV. Experiments Setup}

\subsection*{A. Baseline Approach}

We use the recent work of Peters et al. \cite{1}, which is the most similar to our approach, as the basis of our experimental study. We call it Farsec by following the name of its origin work.

It is a framework for filtering noisy data from NSBRs. It first extracts top X (the number 100 is used in their work) security-related keywords from the SBRs of training data. After that, the similarity between each bug report (in the training set) and these security keywords is calculated. They used seven different filters to filter out noisy data from the training set. To be simple, we use a filter \textit{farsectwo} in the baseline as it performs the best across the five datasets in their study.

\subsection*{B. Research Questions}

Our empirical study aims to answer the following two research questions:

\begin{itemize}
  \item \textbf{RQ1: To what extent can security domain knowledge improve the effectiveness of SBR prediction?}
  \item \textbf{RQ2: How does the source of domain keywords impact the effectiveness of SBR prediction?}
\end{itemize}

Effectiveness is one of the most important goals of machine learning-based data mining, and the same is true for SBR prediction. First, to improve the diversity of security domain knowledge, we combine the security keywords extracted from two different sources (i.e., CWE and CVE). Then, we run three classification RF to evaluate the effectiveness of our domain knowledge-based approach for SBR prediction on five publicly available datasets.

\begin{itemize}
  \item \textbf{RQ2: How does the source of domain keywords impact the effectiveness of SBR prediction?}
\end{itemize}

We raise RQ2 because we have two sources to extract security domain keywords, namely, CWE and CVE. Furthermore, to maximize the diversity of domain knowledge, we use

\begin{itemize}
  \item \textbf{RQ1: To what extent can security domain knowledge improve the effectiveness of SBR prediction?}
\end{itemize}

We answer RQ1 because we have one source to extract security domain keywords. On the other hand, we use two sources to extract security domain keywords, namely, CWE and CVE.
three different fields (i.e., Name, Description, and Extended Description) of CWE for keywords extraction. In other words, we have four sub-sources for security domain keywords extraction. To measure the impact of each source, we use the domain keywords extracted from each sub-source to construct a bug report matrix and then evaluate their effectiveness with classifier RF.

C. Dataset Distribution

We use five datasets (i.e., Chromium, Ambari, Camel, Derby, and Wicket) cleaned and shared by Wu et al. [2] to validate the effectiveness and efficiency of our data labelling framework. These datasets have corrected the mislabels and can be considered as ground-truth. The distribution of these datasets is shown in Table II.

<table>
<thead>
<tr>
<th>Dataset</th>
<th># BR</th>
<th># SBR</th>
<th>% SBR</th>
<th># NSBR</th>
<th>% NSBR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromium</td>
<td>41,940</td>
<td>808</td>
<td>1.93</td>
<td>41,132</td>
<td>98.07</td>
</tr>
<tr>
<td>Ambari</td>
<td>1,000</td>
<td>56</td>
<td>5.60</td>
<td>944</td>
<td>94.40</td>
</tr>
<tr>
<td>Camel</td>
<td>1,000</td>
<td>74</td>
<td>7.40</td>
<td>926</td>
<td>92.60</td>
</tr>
<tr>
<td>Derby</td>
<td>1,000</td>
<td>179</td>
<td>17.90</td>
<td>821</td>
<td>82.10</td>
</tr>
<tr>
<td>Wicket</td>
<td>1,000</td>
<td>47</td>
<td>4.70</td>
<td>953</td>
<td>95.30</td>
</tr>
</tbody>
</table>

To be objective, we follow the work of Peters et al. [1] for the split of the training set and testing set. Namely, we first sort each dataset chronologically; then, it is divided into two equal parts (i.e., 50% and 50%), and the first part is used as the training set, and the other part is the testing set.

D. Classification approach and performance indicator

Classifiers. We use Random Forest (RF) [27] as our classification model since it performs best in the work of Peters et al. [1] and Wu et al. [2]. RF is also a widely applied classification algorithms in the area of software data repository mining [28]–[31]. We use RF implemented in scikit-learn [26]. The key parameter n_estimators is set with value 200 based on the work of Shu et al. [12]. Furthermore, we use the approach SelectFromModel to reduce the dimensionality of the bug reports matrix [32], [33].

Performance indicators. We use all the five indicators applied by Peter et al. for performance evaluation of our approach, namely, Recall (i.e., pd in their work), pf (probability of false alarm), Precision, F1-score, and G-measure. However, during the experiment results analysis, we use F1-score as the major indicator for its justice [16], [34]–[36] In the case of SBR prediction, the possible prediction result of each bug report can be 1 (SBR) or 1 (NSBR). The confusion matrix of the prediction result can be described as below:

- 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 value of performance indicators Recall, pf, Precision, F1-score, and G-measure can be calculated as below.

\[
\text{Recall} = \frac{TP}{TP + FN} \quad (1)
\]

\[
pf = \frac{FP}{FP + TN} \quad (2)
\]

\[
\text{Precision} = \frac{TP}{TP + FP} \quad (3)
\]

\[
F1 – score = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}} \quad (4)
\]

\[
G – measure = \frac{2 \times \text{Recall} \times (1 - pf)}{\text{Recall} + (1 - pf)} \quad (5)
\]

V. RESULTS AND ANALYSIS

A. Response to RQ1

RQ1: To what extent can security domain knowledge improve the effectiveness of SBR prediction?

To answer RQ1, we perform RF with the datasets matrix generated with our domain knowledge approach and Farsec independently. To fully use the security domain knowledge we have, we first merge the security domain keyword roots obtained from the four sub-sources (i.e., Name of CWE, Description of CWE, Extended Description of CWE, and CVE entries of each project). After that, a matrix for each dataset is generated by calculating the frequency of each keyword root in the description of each bug report.

Table III shows the results of the two approaches across the five datasets (i.e., Chromium, Ambari, Camel, Derby, and Wicket). The row with a higher value in terms of major performance indicator F1-score is highlighted in bold. As it shows that our domain keywords-guided approach outperforms the baseline approach Farsec in 80% cases across the five datasets. Farsec achieves a slightly better value (0.28) than our approach (0.27). On average, our approach increases F1-score by 25% compared with Farsec. Moreover, the false positive rate (i.e., pf) of our approach is relatively small, with the value 0.02 on average.

Summary of RQ1

- Security domain knowledge-guided approach performs good on SBR prediction with the best value 0.81 in terms of F1-score.
- Comparing with the baseline Farsec, the F1-score could be increased by 25% on average across the five datasets.

B. Response to RQ2

RQ2: How does the source of domain keywords impact the effectiveness of SBR prediction?

To answer RQ2, we construct the security domain keyword roots matrices for each dataset in the following three ways:

- Constructing matrix with security domain keywords roots of each of the four sub-sources, namely, Name of CWE, Description of CWE, Extended Description of CWE, and
Approach

Our approach

- Combining security domain keywords roots of each of the three sub-sources of CWE with that of CVE. Then, we have three combinations for each dataset, we briefly name them as Name + CVE, Description + CVE, and Extended Description + CVE.

- Combining security domain keywords roots of all the four sub-sources. To be simple, we name it as CWE + CVE.

Finally, there are eight security domain keyword roots matrices generated for each dataset. We then perform RF on each of these matrices and calculate the performance indicators Recall, pf, Precision, F1-score and G-measure for each dataset.

Table IV shows the performance results of different keywords roots sources on each dataset. The row achieves the highest F1-score on each dataset is highlighted in bold. As it shows, the best F1-score reaches 0.81 achieved by source combinations Description + CWE and CWE + CVE on Chromium. Nevertheless, the source achieves the best F1-score on each dataset is not consistent (e.g., CWE + CVE achieves the best on Derby while Description achieves the best on Camel). However, there are indeed two sources outperform the others. Namely, the combination of all the four sub-sources (i.e., CWE + CVE) performs the best on Chromium, Ambari, and Derby; the sub-source description of CWE (i.e., Description) achieves the best F1-score on Ambari, Camel, and Wicket.

**Summary of RQ2**

- Both the security domain keywords of CWE and CVE contributes to the effectiveness of SBR prediction.
- For the three sub-sources (Name, Description, and Extended Description) of CWE, the Description contributes the most.
- Domain knowledge of CVE entries contributes the most on large-scale dataset Chromium.

### VI. Discussion

#### A. Limitations

As shown in Table III, the Recall values obtained by our proposed approach are lower than those obtained by the Farsec for four projects (i.e., Ambari, Camel, Derby, and Wicket). A major reason leads to this would be the class imbalance problem of SBR datasets. Farsec alleviated the class imbalance problem of the SBR datasets while filtering out the potentially noisy records from NSBRs in the training set. We believe our approach could be extended to incorporate class rebalance strategies.

Another limitation is the quality of the extracted domain knowledge. We have improved the quality of domain knowledge by extracting it from authoritative sources and using keyword roots. There are still some seemingly security-irrelevant words that exist (e.g., read, write, file, etc.). However, these words could be security-relevant when combining with other words. For example, the word read would appear when describing a *out-of-bounds read* (CWE-125) vulnerability. This kind of relationship could be better described with techniques like knowledge graph [14] in the future.

#### B. Threats to validity

**Internal Validity.** A threat to the internal validity of this study is the implementation correctness of the approach. Our approach is developed based on publicly-available keywords extraction approach keyBERT [22]. We double reviewed and verified the scripts to guarantee their correctness. Another threat to the internal validity is the sources for extracting security domain knowledge. To guarantee the quality of domain knowledge we obtained, we use the authoritative definition of software vulnerabilities from CWE to extract common security domain keywords. Furthermore, we use the history CVE entries, which are ground-truth security issues of production projects, as the source to extract project-specific security domain knowledge.

**External Validity.** Threats to external validity are concerned with the generality of our experiment results. We use five publicly available datasets of projects in different areas. Furthermore, these datasets are of different scale (Chromium has over

### Table III

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Approach</th>
<th>Recall</th>
<th>pf</th>
<th>Precision</th>
<th>F1-score</th>
<th>G-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromium</td>
<td>Our approach</td>
<td>0.74</td>
<td>0.00</td>
<td>0.91</td>
<td>0.81</td>
<td>0.85</td>
</tr>
<tr>
<td></td>
<td>Farsec</td>
<td>0.66</td>
<td>0.00</td>
<td>0.95</td>
<td>0.78</td>
<td>0.79</td>
</tr>
<tr>
<td>Ambari</td>
<td>Our approach</td>
<td>0.18</td>
<td>0.01</td>
<td>0.56</td>
<td>0.27</td>
<td>0.30</td>
</tr>
<tr>
<td></td>
<td>Farsec</td>
<td>0.50</td>
<td>0.07</td>
<td>0.19</td>
<td>0.28</td>
<td>0.65</td>
</tr>
<tr>
<td>Camel</td>
<td>Our approach</td>
<td>0.27</td>
<td>0.02</td>
<td>0.50</td>
<td>0.35</td>
<td>0.42</td>
</tr>
<tr>
<td></td>
<td>Farsec</td>
<td>0.33</td>
<td>0.12</td>
<td>0.21</td>
<td>0.26</td>
<td>0.38</td>
</tr>
<tr>
<td>Derby</td>
<td>Our approach</td>
<td>0.71</td>
<td>0.08</td>
<td>0.67</td>
<td>0.69</td>
<td>0.80</td>
</tr>
<tr>
<td></td>
<td>Farsec</td>
<td>0.84</td>
<td>0.54</td>
<td>0.27</td>
<td>0.41</td>
<td>0.60</td>
</tr>
<tr>
<td>Wicket</td>
<td>Our approach</td>
<td>0.42</td>
<td>0.00</td>
<td>0.91</td>
<td>0.57</td>
<td>0.59</td>
</tr>
<tr>
<td></td>
<td>Farsec</td>
<td>0.52</td>
<td>0.05</td>
<td>0.34</td>
<td>0.41</td>
<td>0.67</td>
</tr>
<tr>
<td>Average</td>
<td>Our approach</td>
<td>0.46</td>
<td>0.02</td>
<td>0.71</td>
<td>0.54</td>
<td>0.59</td>
</tr>
<tr>
<td></td>
<td>Farsec</td>
<td>0.57</td>
<td>0.16</td>
<td>0.40</td>
<td>0.43</td>
<td>0.64</td>
</tr>
</tbody>
</table>
40k records, while the other four has 1,000 records each). All these can guarantee the diversity of the experimental datasets. Thereby issue the generability of our approach. Nonetheless, we do not claim that our approach and experiment results can be generalized to all software analytics tasks. However, it would be applicable to supervised machine learning-based SBR prediction tasks.

VII. CONCLUSION

This work focuses on SBR prediction, which is an essential way of decreasing the security risks of software products. It is inspired by the work of Peters et al. [1] and Wu et al. [2]. Peters et al. provide the idea of using security keywords matrix to make the model training data more focused on security-related information. And Wu et al. give us the idea to extract security domain knowledge from authoritative data repositories like CWE and CVE, which would greatly improve the quality of the security keywords. We improve the approach by using keyBERT to enhance the quality of extracted keywords. Furthermore, we use the roots of keywords to make the security keywords matrix more meaningful. Finally, we evaluate our security domain knowledge-guided approach by comparing it with the baseline approach proposed by Peters et al. [1]. The results show it could improve the effectiveness of SBR prediction by 25% on average and reaches 0.81 in terms of F1-score in the best case.

ACKNOWLEDGMENT

This research was partially supported by the Key Laboratory of Advanced Perception and Intelligent Control of High-end Equipment, Ministry of Education (GDSC202006) and the 2021 Key R&D Program in Shaanxi Province (2021GY-041).

REFERENCES


