2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE) | 978-1-6654-2587-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/ISSRE52982.2021.00043

Automatically Identifying Bug Reports with Tactical Vulnerabilities by Deep Feature Learning

Wei Zheng[†], Manqing Zhang[†*], Hui Tang[†], Yuanfang Cai[‡], Xiang Chen[§], Xiaoxue Wu[¶],
and Abubakar Omari Abdallah Semasaba[†]
[†]School of Software, Northwestern Polytechnical University, China. wzheng@nwpu.edu.cn,
{zmqgeek, th1605285104}@gmail.com, abubakar.semasaba@hotmail.com
[‡]Department of Computer Science, Drexel University, USA. yfcai@cs.drexel.edu
[§]School of Information Science and Technology, Nantong University, China. xchencs@ntu.edu.cn
[¶]School of Information Engineering, Yangzhou University, China. xiaoxuewu@yzu.edu.cn

***Abstract*—Identifying and fixing bug reports with tactical vulnerabilities in a timely and accurate manner is essential to ensure the security of the software architecture. Manually identifying the bug reports with tactical vulnerabilities is labor-intensive and challenging. This paper presents *Itactivul*, an approach to automatically identify bug reports with *tactical vulnerabilities* and recommend their tactical categories to guide the fix. Unlike the existing security bug report prediction approach, we are the first attempt to use deep learning to mine discriminative tactical text features only from the vulnerability descriptions of the National Vulnerability Database (NVD) and apply them to identify bug reports with tactical vulnerabilities. We evaluate *Itactivul* on three bug reports datasets gathered from three large-scale open-source projects, including Chromium, PHP, and Thunderbird. The experimental results show that Itactivul outperforms baselines by an average of 8.88%, 13.58%, and 6.61% in the F1-score of three datasets, respectively. To improve the explainability of the features mined by Itactivul, we manually analyze the high-weight phrases extracted by using attention backtracking. The results show that Itactivul can mine key and potential tactical vulnerabilities text features.**

***Index Terms*—Tactical Vulnerability, Model explainability, Bug Report, Text mining**

I. Introduction

More and more attention has been paid to security incidents caused by architectural weaknesses [1]–[7]. Previous research [8] showed that weaknesses in software design account for about 50% of security problems. To ensure software security, architects need to consider security quality attributes at the design stage [9]–[11]. Security tactics [12] are suites of techniques, which have been widely used to address various security concerns [13]. Architects need to make decisions and choose proper security tactics to meet system security requirements.

Despite the application of proper security tactics at the beginning of the project, the security of the architecture might erode with the evolution of the software or incorrect implementation in the code [14]–[16]. Mirakhorli and Cleland-Huang's study [17] showed that due to developers' lack of security coding experience, the improper implementation or deterioration of security tactics in the development and maintenance process could lead to severe *tactical vulnerability* [18]. Security engineers need to manually identify security bug reports (e.g., tactical vulnerability) in the bug-tracking system, and fix vulnerabilities in a timely and effective manner, thereby improving software security assurance [19]. However, this task is labor-intensive and time-consuming [20].

To automate identify security bug report for reducing human efforts, some mining-based models have been proposed [20]–[25]. However, these efforts only identify security reports and do not classify different types of vulnerabilities to determine the priority of repair. Tactical vulnerabilities are more subtle and sophisticated than code-level vulnerabilities (such as buffer overflows [26]). More importantly, tactical vulnerabilities related to the architecture usually have a broader impact, which makes them more difficult to fix and requires architectural knowledge. For example, *Authorize Actors*' tactical weakness in Intel's processor chips [27] forced a significant redesign of Linux and Windows. Therefore, it is necessary to separately identify these system-level tactical vulnerabilities so that they can be identified and fixed more effectively.

Observations and Insights. In this study, we want to explore an automated approach to identify tactical vulnerabilities and their categories in bug reports to solve this problem. This approach can help developers fix them in a timely and accurate manner. However, only a few bug reports are related to security tactics, and it is difficult to mine their features automatically. Inspired by the study of Bhuiyan et al. [28], it is possible to learn and mine vulnerability text features from the NVD[1] vulnerability descriptions. We use deep learning to mine the text features of tactical vulnerabilities in NVD vulnerability descriptions automatically. Then we apply the trained model to identify bug reports with tactical vulnerabilities. To the best of our knowledge, there is no prior work focusing on automatically identifying bug reports with tactical vulnerabilities.

Mining architecture security tactical features from the NVD vulnerability descriptions automatically is a challenging task. First, the proportion of different types of tactical vulnerability description data varies significantly. This results in a class imbalance issue for the tactical vulnerability feature mining task. Second, since the vocabulary of the NVD vulnerability de-

[*] Manqing Zhang is the corresponding author.

[1] https://nvd.nist.gov/vuln/

2332-6549/21/$31.00 ©2021 IEEE
DOI 10.1109/ISSRE52982.2021.00043

scriptions is different from the bug reports, out-of-vocabulary (OOV) words are widespread in the bug reports. Therefore, OOV words of the new bug report cannot be appropriately mapped to pre-trained word embeddings. Third, the neural network is a black-box model, which uses metrics (such as F1-score) to determine the effectiveness of text feature mining. However, there is no practical explanation about the specific characteristics of the automatically mined text features.

Our solution. This paper presents Itactivul, an effective approach to automatically identify bug reports with *tactical vulnerabilities* by using deep learning. First, Itactivul leverages attention-based Bi-directional Long Short Term Memory (BiLSTM) to automatically capture text features from the NVD vulnerability description. In the training process, we designed a weighted focal loss function to deal with the class imbalanced problem. At the same time, to solve the representation of OOV words, we adopt Fasttext [29], [30] to train word embeddings. Then, we extract the attention weight to highlight the prominent text feature phrases in the input data that help identify tactical vulnerabilities. The prominent key phrases provide an intuitive explanation for the text features automatically mined by Itactivul. After the model construction process, the trained model can be applied to identify bug reports with tactical vulnerabilities.

In our empirical study, we first collected 91,122 vulnerability descriptions in the NVD. By leveraging Santos et al.'s Common Architectural Weaknesses Enumeration (CAWE) [31] as the foundation, we labeled these data to different tactical vulnerability classes. The gathered NVD dataset is applied to the training model to realize in-depth tactical vulnerability text feature mining. After that, we evaluate the performance of Itactivul and the baselines on the bug report datasets of three projects (i.e., Chromium, PHP, and Thunderbird). The evaluation results show that (1) Itactivul outperforms baselines in terms of all metrics *Accuracy, Precision, Recall, and F1-score*. (2) The *Attention, Fasttext embedding, and Focal loss function* are effective and helpful for boosting the effectiveness of Itactivul. (3) After manually analyzing the high-weight phrases extracted by attention backtracking, we find that the majority of key phrases learned by Itactivul are related to the classification of tactical vulnerabilities.

We summarize the contributions of our study as follows:

- We propose Itactivul, an approach to automatically identify bug reports with different tactical vulnerabilities. Itactivul can learn the tactical vulnerability characteristics from the NVD vulnerability descriptions and alleviate the OOV word problem and the class imbalance problem.
- We designed an attention backtracking method to extract and highlight prominent text feature phrases in the input data, which intuitively explains the effectiveness of automatically mined text features.
- We evaluated our approach on three bug report datasets gathered from three large-scale open-source projects (i.e., Chromium, PHP, and Thunderbird). The evaluation results show that our approach significantly outperforms the baselines. Our gathered dataset and the scripts of our study are publicly available[2].

The remainder of this paper is organized as follows. Section II describes the background of our study. Section III presents our approach in detail. Section IV presents our experimental settings. Section V analyzes our experimental results of each research question respectively. Section VI discusses the performance of various hidden layers of our method and analyzes the potential threats to the validity of our study. After a brief review of related work in Section VII, we conclude this paper and point out potential future directions in Section VIII.

II. Background

A. Tactical vulnerability

To build a security software system, software engineers have to implement specific security quality concerns in the design phase. Security tactics are used to ensure the confidentiality, integrity, and availability of the software system to achieve system security and resistance to attacks. Specifically, different security tactics are designed to meet four types of security quality attributes [32]: detect attacks (e.g., Detect Intrusion), resist attacks (e.g., Limit Exposure), react to attacks (e.g., Revoke Access), and recover from attacks (e.g., Service Restoration).

However, the lack of experience of architects or developers may cause the lack, misuse, and improper implementation of architectural tactics. Tactical vulnerabilities are defined as vulnerabilities caused by the above-mentioned architectural tactics-related issues. Tactical vulnerabilities are mapped into three types according to their three reasons: Omission, Commission, and Realization [31]. For example, Ruby's paranoid2 package[3] has led to Realization-type tactical vulnerability (CVE-2019-13589[4], CVSS score: 9.8 CRITICAL) due to the improper implementation of Limit Exposure tactics. Specifically, Limit Exposure tactics minimize the attack surface by designing a system with the least number of entry points. However, paranoid2 package includes a code-execution backdoor because there is no strict restriction on the entry point of the third-party code during the implementation process.

B. Common Architectural Weaknesses Enumeration

In our study, the Common Architectural Weakness Enumeration (CAWE) refers to the classification catalog of security architecture weakness proposed by Santos et al. [31]. Their paper analyzed Common Weakness Enumeration (CWE) into different types of tactical weaknesses and common coding bugs through security tactical keyword search and manual analysis of CWE descriptions.

The CAWE catalog[5] is created by combining the CWE and security tactics, which describes the architecture weaknesses in the software system and provides mitigation techniques to

[2] https://github.com/zmqgeek/Itactivul
[3] https://rubygems.org/gems/paranoid2/
[4] https://nvd.nist.gov/vuln/detail/CVE-2019-13589
[5] https://cwe.mitre.org/data/definitions/1008.html

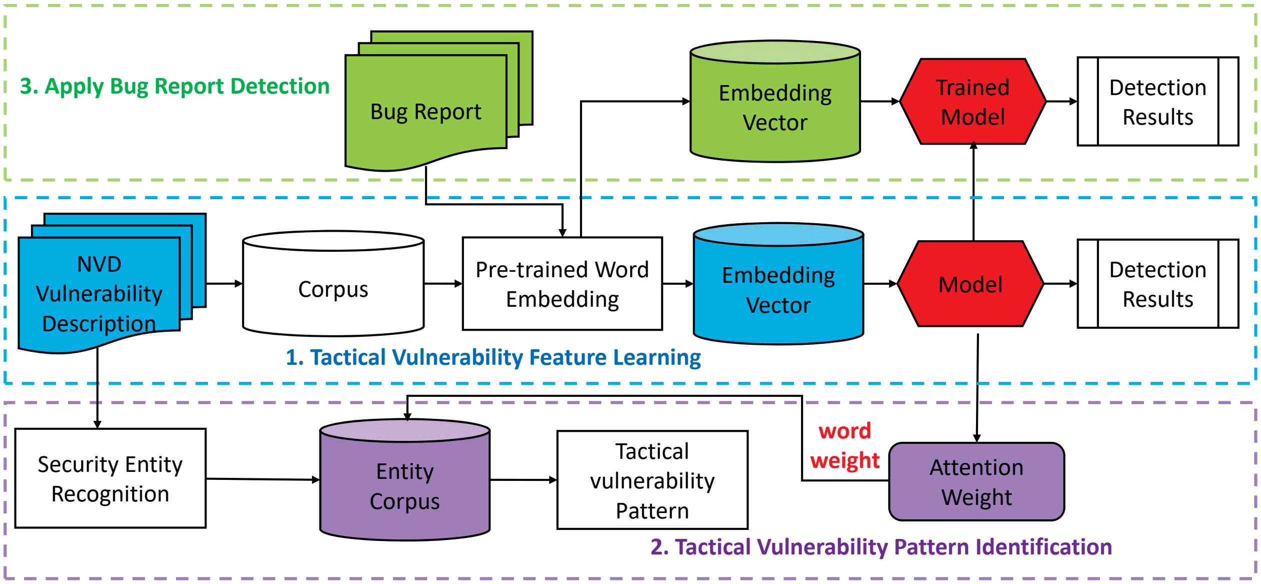

3. Apply Bug Report Detection
Bug Report
Embedding Vector
Trained Model
Detection Results
NVD Vulnerability Description
Corpus
Pre-trained Word Embedding
Embedding Vector
Model
Detection Results
1. Tactical Vulnerability Feature Learning
Security Entity Recognition
Entity Corpus
Tactical vulnerability Pattern
word weight
Attention Weight
2. Tactical Vulnerability Pattern Identification

Fig. 1. Overview of the proposed method.

address them. At present, the CAWE catalog is classified based on the impact of 12 security tactics and contains 224 weaknesses. Each CAWE entry consists of several parts, including impacted security tactics, text descriptions, weakness types, source code examples, mitigation techniques, and detection methods. Santos et al. [31] used a web-based interactive method to help architects and developers follow the guidelines of security tactics to identify potential weaknesses related to a specific security tactic.

III. Our Proposed Approach

In this section, we show an overview of our approach and technical details for each pahse.

A. Overview of Itactivul

Fig. 1 demonstrates the overall framework of Itactivul, which includes three phases: *Tactical Vulnerability Feature Learning, Tactical Vulnerability Pattern Identification, Apply Bug Report Detection.*

Phase 1: *Tactical Vulnerability Feature Learning*. We aim to learn tactical features in the vulnerability by attention-based BiLSTM in this phase. Specifically, We crawled and labeled NVD vulnerability descriptions as our feature learning data. Then, we built a corpus of NVD descriptions for learning domain-specific word embedding vectors. Following that, NVD data was fed into an attention-based BiLSTM to learn tactical vulnerability characteristics automatically. During the training process, we designed a focal loss function to solve the class imbalance problem.

Phase 2: *Tactical Vulnerability Pattern Identification.* We extracted the weight matrix of the attention layer of the model. The weight matrix reflects the contribution of different input words to the recognition result. Next, we used name entity recognition to extract phrases in vulnerability descriptions. Finally, we obtained the key tactical pattern features according to the word and phrase weight ranking.

Phase 3: *Bug Report Detection*. In this phase, the bug report includes a bug summary and bug description as the input of the trained model in Phase 1. Then, the bug report is mapped to a word embedding matrix based on the trained word embedding dictionary. The model will output two types of information as a detection report: the tactical vulnerability category and the attention weight of each word in the new bug report.

B. Tactical Vulnerability Feature Learning

We automatically learn the tactical vulnerability features in the vulnerability description using attention-based BiLSTM. This subsection introduces our word embedding method, attention-based BiLSTM architecture, and our designed focal loss function.

1) Word Embedding: Word embedding as the input of neural networks can help capture the grammatical and semantic information of sentences. An accurate and effective word embedding representation can significantly improve the performance of downstream deep learning tasks [33] [34]. For example, Sui et al. [33] proposed a value-flow-based precise code embedding method, which effectively improves the performance of code classification and code summarization tasks. In classification tasks, we often use Word2vec [35] to generate word vectors of words. However, Word2vec cannot generate word vectors for OOV words. Furthermore, some low-frequency words in the vulnerability description may have more significant meanings. To this end, we use FastText [36], [37] to generate our word vector dictionary. It can better generate word vectors of low-frequency words and OOV words by learning character-based embedding representation of subwords.

To learn domain-specific word embeddings, we use the NVD vulnerability description as the corpus for word vector training. For the generated corpus, we learn domain-specific word embeddings using FastText provided by open-source Python library gensim[6]. The output of FastText represents each word as a 300-dimensional vector. The vector of each dimension represents a specific feature value of the word. Finally, the pre-trained word vector is used to convert the input sentence into an embedding matrix in the embedding layer of the model.

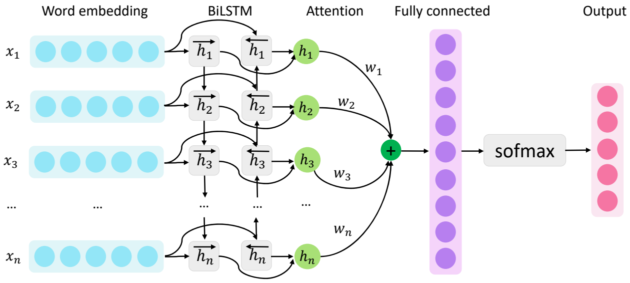


Fig. 2. The architecture of our model

2) The Architecture of Attention-Based BiLSTM: Fig. 2 shows the architecture of our Attention-based BiLSTM model. Our model takes as input an vulnerability description $X = [x_1, x_2, x_3, \cdots, x_n]$. Through the word embedding layer, each word x_i in the sentence is mapped into a fixed-length vector e_i. The mapping of the dictionary learned through the word embedding phases, we get an input matrix of $n \times 300$ (300 is the dimension of word embedding).

After the embedding layer, the word vector sequence is fed into the BiLSTM to achieve a higher-level sentence vector representation. The BiLSTM layer uses two LSTMs to learn sentence features from the forward and reverse directions, respectively. Two LSTMs respectively calculates the embedding vector to obtain the forward output vector $\overrightarrow{h_i}$ and the reverse output vector $\overleftarrow{h_i}$ of each word x_i. The word vector h_i of each word x_i finally output by BiLSTM is calculated as follows:

$$h_i = \left[\overrightarrow{h_i} \oplus \overleftarrow{h_i}\right] \tag{1}$$

where $\oplus$ represents the addition of each dimension of the vector.

Generally, not all words in a vulnerability contribute equally to the representation of tactical vulnerability, so we leverage the word attention mechanism to capture the distinguished influence feature word. The Attention layer considers the weights of different words, and finally calculates the BiLSTM output vector $H = [h_1, h_2, \cdots h_n]$ to form a dense vector. Formally, the feature vector r finally learned by the attention layer is calculated by:

$$M = \tanh(H) \tag{2}$$

$$w = \operatorname{softmax}\left(\alpha^T M\right) \tag{3}$$

$$r = Hw^T \tag{4}$$

where M represents the hidden representation of H obtained through one layer of MLP. Then we measure the importance of words by the similarity between M and the randomly initialized word-level context vector α. After that, by learning the parameter α^T, we get a normalized importance weight w through a $softmax$ function. Finally, the output sentence vector r of the attention layer is the weighted sum of H.

The attention layer is followed by two fully connected layers with different sizes of neurons. For the attention output vector fed into the first fully connected layer, we get the final vector h^* for classification through the activation function $tanh$.

$$h^* = \tanh(r) \tag{5}$$

Finally, after the second fully connected layer, the output pseudo probability value is obtained through the activation function $softmax$, which represents the probability of each category predicted by the model.

$$\hat{p}(y \mid S) = \operatorname{softmax}\left(W^{(S)} h^* + b^{(S)}\right) \tag{6}$$

3) Focal loss function: In our work, different types of tactical vulnerability description data have a very imbalanced distribution. The normal cross-entropy loss function commonly used in multi-classification problems does not perform well in our tactical vulnerability classification tasks. To solve the class imbalanced problem, we can use the weighted cross-entropy loss function to guide the training process. However, the performance of this loss function cannot achieve satisfactory performance in our task.

To figure out the reason, we manually checked the difference between the pseudo-probability value output by the model and the real label. We find that there are many *Hard Examples* in the data. For example, our model output $C (C = 12)$ pseudo probability values 0.1, 0.2, 0.6, 0.6, **0.1**, 0.5, **0.7**, 0.3, 0.1, 0.6, 0.5, 0.4, where C is the number of classification categories. The classifier outputs the seventh category with the largest probability value of 0.7 as the prediction result, but the true label is the fifth category, and the probability value is 0.1. We can find that the probability of different classes in this example is not much different, and the predicted probability is small. The weighted cross-entropy loss function does not perform well on this kind of *Hard Examples*.

To solve the above problem, we extend the weighted cross-entropy loss function to the focal loss function. The focal loss function can effectively solve *Hard Examples* and unbalanced data. Following the study of Lin et al. [38], we define the focal loss function FL of our model as follows:

$$FL = -\alpha_t \left(1 - p_t\right)^\gamma \log\left(p_t\right) \tag{7}$$

$$\alpha_t = \frac{median\_freq}{freq(t)} \tag{8}$$

where t is the class label of the current sample, and p_t represents the probability value of class t predicted by the classifier. The index γ can better control the size of the weight,

[6] https://radimrehurek.com/gensim/

we use the common values $\gamma = 2$. *Hard Examples* can be effectively solved by adding $(1 - p_t)^\gamma$. At the same time, to solve the problem of sample category imbalance, we add the class weight α_t.

We use median frequency balance [39] as class weight α_t, where $median\_freq$ represents the median frequency of different classes on the training set, $freq(t)$ is the frequency of the current sample class t. This implies that the contribution of small samples in the training set to loss is enhanced, and a large number of samples is weakened while retaining the normal loss of the intermediate number of samples [40].

C. Tactical Vulnerability Pattern Identification

In this work, we use deep learning to automatically learn the tactical vulnerability characteristics in the vulnerability description. We are interested in understanding and explaining what features the model has learned. Based on this, we can summarize the pattern of tactical vulnerabilities to help developers better distinguish them. As shown in Fig. 2, for the attention layer, our model calculates the weight w_i of each word x_i in the output of the BiLSTM layer. The w calculated in the attention layer reflects the importance weight of each word x_i in the sentence X. w_i is the weight of the i-th word in the sentence X. The weight feature value w_i in the attention layer can be traced back to the input sentence X, therefore deriving the importance of each word x_i.

We apply the traceability of the input sentence X and the weight matrix w to extract the most important feature words for identifying tactical vulnerabilities. In this way, we get the word weights of all the vulnerability description sentences. However, certain key phrases have specific meanings, such as "denial of service". If we sort weight only in terms of words, the order of these phrases may be destroyed. To solve this problem, we extract security-related entities from unstructured vulnerability descriptions as the basis for pattern classification. We use Named Entity Recognition (NER) to extract the security information in the vulnerability description. Previous studies have shown that NER can be effectively used to extract security entities in the field of cyber security [41]–[45]. Since NER requires pre-labeled data, we apply the NVD entity labeling dataset open-sourced by Bridges et al. [46] as our training data. The corpus marked CVE vulnerability descriptions from 2010 to 2013, involving over 650,000 tokens and 13 security-related entity categories. We use the *Stanford NER*[7] to train a model for secure entity recognition on the corpus. Then, we apply the NLTK[8] python package calls the well-trained entity model to extract the security-related entities se from the vulnerability description.

Finally, we identify tactical vulnerability patterns based on the extracted entity information and word weights in sentences. Specifically, for a given vulnerability description X, we first get the weight w of all words. Then we get the weight se_w of all phrase entities in the sentence. se_w is the average weight of the phrase's constituent words. Formally, the security entity phrase weight se_w is calculated by:

$$se_w = \frac{w_{i+1} + \cdots + w_{i+m}}{m} \tag{9}$$

where security entity phrase consists of the (i+1)-th word to the (i+m)-th word of the sentence. w_{i+1} represents the weight of the (i+1)-th word, and m represents the number of words that make up the phrase.

We sort the security entity phrase weights se_w and the rest of the word weights w_i to get the tactical vulnerability pattern of the most important security entities phrase or words in the sentence.

D. Apply Bug Report Detection

In Phase 1, we got a well-trained model, which automatically learns the tactical vulnerability characteristics from the vulnerability description. We can apply the well-trained model to detect bug reports with tactical vulnerabilities.

Specifically, given a bug report, which includes bug summary and bug description, it is converted into an input embedding vector matrix by the word embedding dictionary of phase 1. The input matrix is fed to the well-trained BiLSTM model. Finally, the output layer gives the probability of each category, with the highest probability being the identified tactical vulnerability category. In addition, we also output the attention weight corresponding to each word to form a detection report to help developers understand it in more detail.

IV. Experimental Setup

In our empirical study, we aim to answer the following three research questions (RQs):

- RQ1: How effective is Itactivul in identifying bug reports with tactical vulnerabilities?
- RQ2: How effective are the main components of Itactivul for learning the text feature of tactical vulnerabilities?
- RQ3: What are the characteristics of tactical vulnerabilities learned by Itactivul?

A. Data Collection

We first crawled 136,050 CVE data from the NVD website. These CVE data were published from 1996 to December 31, 2019, covering popular open-source projects, such as Android, Flash Player, Leap, Firefox, Mysql, JRE, PHP, Chrome, Linux Kernel, and Wireshark. The data contains the CVE index (such as CVE-2018-18882), CWE ID, vulnerability descriptions, and reference links. Among the 136,050 CVEs, we extracted 91,122 items with specific CWE ID and used these data to train the models.

To evaluate the actual effect of Itactivul on the bug report and compare it with baselines, we constructed the first tactical bug report dataset. Benefit from the paper of Santos et al. [47], they empirically studied the tactical vulnerability of three large-scale open-source systems Chromium, PHP, and Thunderbird. Their dataset classifies bug reports into different tactical vulnerability categories, but only contain links to project bug reports. To this end, we crawled all bug reports

[7] https://nlp.stanford.edu/software/CRF-NER.shtml

[8] http://www.nltk.org/

(including bug summary and bug description) based on these links as our bug report evaluation dataset.

B. Data Preprocessing

We adopt the following processes to classify and preprocess the collected CVEs:

Data classification. CAWE categorizes the CWE ID corresponding to each tactical vulnerability. First, we counted all the CWE IDs corresponding to each tactic. Then, we classified the 91,122 NVD data into 11 tactical vulnerabilities and non-tactical vulnerabilities based on their CWE ID. The data distribution is shown in TABLE I. The NVD vulnerability data set is divided into training set validation set and test set at a ratio of 6:2:2 for training and evaluation of our approach. The bug reports of the three projects were divided equally for training and testing baselines.

TABLE I
DATA STATISTICS FROM NVD, CHROMIUM, PHP AND THUNDERBIRD

Num	Tactical	NVD	Chromium	PHP	Thunderbird
T1	Audit	194	2	0	0
T2	Authenticate Actors	2,576	9	0	4
T3	Authorize Actors	3,826	99	10	105
T4	Cross Cutting	54	0	0	0
T5	Encrypt Data	1,164	8	2	7
T6	Identify Actors	603	26	0	33
T7	Limit Access	529	14	4	5
T8	Limit Exposure	17	107	7	8
T9	Manage User Sessions	179	0	2	0
T10	Validate Inputs	33,534	151	37	189
T11	Verify Message Integrity	130	0	0	0
NT	Non-tactical	48,316	763	104	1,079
Total	12	91,122	1,179	166	1,430

Data processing. To filter out the templated and trivial information from the data, we use regular expressions to preprocess the NVD data. We identify and delete the following elements from the messages: (1) punctuations, (2) URL, (3) email addresses, (4) author information, e.g., "reported by", (5) acknowledgment, e.g., "Thanks to", (6) commit date, (7) fix references, e.g., "Fixes #25995". Finally, we use a widely-used NLP library NLTK to process the descriptions and save the results in a binary file. Since the machine learning model is standardized for the input text preprocessing, we also adopt the above preprocessing methods for bug reports.

C. Baselines

As this is the first work on identifying tactical bug reports, there are no corresponding baselines. For the bug report analysis task, there are some work on security bug report prediction [20]–[22], [48], which can be adapted to identify tactical bug reports. Therefore, we select two recent studies proposed by Peter et al. [21] and Shu et al. [22] as the baseline approach in this study.

Both approaches use Random Forest (RF), Naive Bayes (NB), KNearest Neighbor (KNN), Multilayer Perceptron (MLP), and Logistical Regression (LR) to train classifiers based on the textual description of bug reports. To be objective, similar to Wu et al.'s study [49], we directly use the source code shared by Peter et al. [21] and Shu et al. [22] to suit our tasks and keep all the settings of the parameters (i.e., key parameters, the default values, and the tuning range of each key parameter). Note that the length of the input text remains the same as our approach. At the same time, to mitigate bias of comparison with our method, We exploit the word embedding of our method to convert the bug report into a vector matrix as the input of these five classifiers.

D. Evaluation Metrics

Identifying bug reports with tactical vulnerabilities is a typical text classification task. We employ the commonly-used metrics to evaluate the effectiveness of the models: *Accuracy*, *Precision*, *Recall* and *F1-score*. The calculation of these metrics requires the classification result to be expressed as an $n*n$ *Confusion Matrix* M, where n denotes the number of classes. The rows i represent the true categories, and the columns j represent the predicted categories, where $i, j \in n$. The diagonal $M_{ij}\ (i = j)$ indicates the correct category of i-type tactics.

Accuracy represents the proportion of correctly classified samples to all samples. *Precision* measures the ratio of true-positive samples to the total prediction results. It indicates how many of the predicted positive samples are actually positive samples. *Recall* is an indicator of coverage. It measures the ratio of true-positive samples to the total number of samples that are actually tactical categories. *F1-score* is the most effective and the most important metric of model evaluation. It is defined as the harmonic average of Precision and Recall. Thus, *F1-score* can give the most reasonable evaluation results of the model performance. In general, *Precision* and *Recall* are usually inversely proportional, and both of them are difficult to balance. As a comprehensive evaluation metric, *F1-score* solves this problem well. Therefore, we choose *F1-score* as the main evaluation metric.

E. Model Configuration

We perform hyper-parameter optimization so that the machine learning-based model can perform better and learn better classification rules. To select the best classification model, we compare the loss value of the validation set during the iterative training process and then save the model when the loss value is lowest. We have also selected the best parameter values to ensure the effectiveness of classification.

For example, the length of tokens varies significantly in our dataset. It has a big gap between the shortest description (i.e., four tokens of CVE-2018-20596) and the longest description (i.e., 587 tokens of CVE-2019-6568). We observe that the early changes in max_len have a significant impact on accuracy. With the increase of max_len, the accuracy of models tends to be stable gradually. If max_len is too small, it will lose many important features, while it will run out of memory and affect the training speed when max_len is enormous. The best range of max_len is between 80 and 110. In addition, we find that in 95% of the cases, the length of tokens is less than

80, which is our basis for parameter setting. Note that other parameters of the classification model can be optimized in a similar pattern.

V. Experimental Results

A. RQ1: How effective is Itactivul in identifying bug reports with tactical vulnerabilities?

To demonstrate the effectiveness of Itactivul in identifying bug reports with tactical vulnerabilities, we evaluate our approach in three bug report datasets. These three datasets come from three open source projects Chromium, PHP, and Thunderbird bug report, which is described in Section IV-A. TABLE II presents the performance of Itactivul on three bug report datasets and compares it with the baselines. We can observe that Itactivul outperforms baselines in terms of all metrics *Accuracy, Precision, Recall, and F1-score*. The average improvements in terms of the three F1-score compared to the baselines are 8.88%, 13.58%, and 6.61% points, respectively. These results indicate that the tactical features automatically learned by Itactivul can be effectively applied to bug reports. Compared to the five baselines, Itactivul can capture the key text features more precisely to identify tactical vulnerabilities.

We also observe that there are significant differences in the evaluation performance of the three projects. The performance of Itactivul and baselines in Thunderbird bug reports is higher than Chromium and PHP bug reports. This finding shows that the baseline method might be sensitive to the training dataset. Therefore, existing approaches for security bug report prediction are limited by the size of the dataset and the single source of information (bug report related information), cannot perform well for bug reports with tactical vulnerabilities. The training dataset of Itactivul is NVD vulnerability descriptions and is not limited to bug reports for specific projects. Itactivul is the first attempt to solve this problem. Our results have shown that it is a promising approach and could be effective and practical for identifying bug reports with tactical vulnerabilities in real-world software projects.

Summary for RQ1: Itactivul outperforms the baselines in terms of all metrics *Accuracy, Precision, Recall, and F1-score* and hence is effective and practical for identifying bug reports with tactical vulnerabilities.

B. RQ2: How effective are the main components of Itactivul for learning the text feature of tactical vulnerabilities?

To assess the effectiveness of each component, we evaluate Itactivul using the NVD test dataset with 18,225 vulnerability descriptions and compare it with the combination of each component (combination baselines). The models used for evaluation are the best-trained models. TABLE III presents the evaluation results, and Itactivul is referred to as *BiLSTM+Att+Fast+FL*. The columns *Accuracy, Precision, Recall, and F1-score* show combination baselines and Itactivul performance in the four basic classification metrics.

We can observe that *Attenion, Fasttext embedding and Focal loss function* can achieve performance improvement on all four evaluation metrics. As shown in TABLE III, under the same model structure, adding *Attention* performs better. For example, *BiLSTM+Att+Fast* outperforms *BiLSTM+Fast* in terms of all metrics, which means adding *Attention* can learn tactical features more effectively. *Fasttext embedding and Focal loss function* are similar to the observations of *Attention*. This observations shows that adding *Attenion, Fasttext embedding and Focal loss function* helpful for boosting the effectiveness of our approach.

Compared with the combination baselines, Itactivul can identify more tactical categories. As shown in TABLE III, Itactivul can identify three types of tactics, T1, T9, and T11, which are not recognized by the baselines. The main component that causes the difference in the classification results comes from our loss function. Our approach extends the normal cross-entropy loss function to the focal loss function. This evaluation result shows that our loss function can effectively deal with the class imbalance, and is helpful for the learning of different types of tactical vulnerability features.

Furthermore, through TABLE III, we observe that two types of tactical vulnerabilities, T4 (Cross Cutting) and T8 (Limit Exposure), could not be identified in the test set. To figure out the reason, we inspect the data of these two types of tactical. We find that weaknesses in the *Cross Cutting* category are related to the design and architecture of multiple security tactics and how they affect a system. The vulnerability description text features in the *Cross Cutting* category overlap with other tactical strategies, making it difficult to abstract and effectively learn to distinguish these features significantly. For *Limit Exposure* tactical category, we find that its data is only 17 in the NVD data set. The highly scarce amount of *Limit Exposure* tactical category data makes it difficult for the multi-class model to learn its characteristics, especially since the training data is unbalanced.

Summary for RQ2: The *Attenion, Fasttext embedding, and Focal loss function* are effective and helpful for boosting the effectiveness of our approach. In addition, the loss function is helpful to deal with class imbalance.

C. RQ3: What are the characteristics of tactical vulnerabilities learned by Itactivul?

To understand which features the model has learned to distinguish different tactical vulnerabilities, we retroactively counted the weight of key phrases according to the weight of attention. TABLE IV lists a sample description of *Authenticate Actors* tactical vulnerability. As the vulnerability description shows, the vulnerability stems from the improper processing of authentication requests by Windows, resulting in a privilege escalation vulnerability in multiple versions of Windows servers. We can observe that the word **authentication** has the highest recognition feature weight (i.e., the darkest color). As we know, *Authenticate Actors tactical vulnerability* not only needs to be related to authorization, but also requires some specific trigger conditions. The trigger condition words **improperly** and **handles** are the feature weights of sentences

TABLE II
PERFORMANCE RESULTS OF THE BASELINE APPROACHES AND ITACTIVUL ON CHROMIUM, PHP, AND THUNDERBIRD BUG REPORTS

Approach	Chromium				PHP				Thunderbird			
	Accuracy	Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score
RF	0.6034	0.4466	0.6034	0.5065	0.6024	0.5246	0.6024	0.5057	0.7343	0.6047	0.7343	0.6485
NB	0.5881	0.3844	0.5881	0.4506	0.5542	0.4986	0.5542	0.5164	0.7217	0.5632	0.7217	0.6325
KNN	0.6186	0.4957	0.6186	0.5322	0.6145	0.5804	0.6145	0.5386	0.7287	0.5923	0.7287	0.6371
MLP	0.6051	0.3661	0.6051	0.4562	0.5783	0.3344	0.5783	0.4238	0.7287	0.5310	0.7287	0.6143
LR	0.6153	0.3785	0.6153	0.4687	0.6145	0.3776	0.6145	0.4677	0.7343	0.5391	0.7343	0.6218
Itactivul	**0.6616**	**0.5295**	**0.6616**	**0.5717**	**0.6667**	**0.5927**	**0.6667**	**0.6263**	**0.7448**	**0.6871**	**0.7448**	**0.6970**
Average	+5.55%	+11.63%	+5.55%	+8.88%	+7.39%	+12.95%	+7.39%	+13.58%	+1.52%	+12.10%	+1.52%	+6.61%

* RF, NB, KNN, MLP, and LR refer to five baseline methods. Average refers to the average difference between Itactivul and the baseline method.
* The best results are highlighted in bold.

TABLE III
EFFECTIVENESS OF EACH COMPONENT IN ITACTIVUL IN EVALUATION METRICS AND IDENTIFIABLE TACTICAL

Approach	Accuracy	Precision	Recall	F1-score	Identifiable Tactical	Non-Identifiable Tactical
BiLSTM+Word	0.8372	0.7997	0.8372	0.8165	T2, T3, T10, NT	T1, T4, T5, T6, T7, T8, T9, T11
BiLSTM+Fast	0.8445	0.8246	0.8445	0.8331	T2, T3, T5, T10, NT	T1, T4, T6, T7, T8, T9, T11
BiLSTM+Att+Word	0.8557	0.8503	0.8557	0.8496	T2, T3, T5, T6, T7, T10, NT	T1, T4, T8, T9, T11
BiLSTM+Att+Fast	0.8623	0.8554	0.8623	0.8541	T2, T3, T5, T6, T7, T10, NT	T1, T4, T8, T9, T11
BiLSTM+Att+Fast+FL	**0.8658**	**0.8623**	**0.8658**	**0.8618**	**T1**, T2, T3, T5, T6, T7, **T9**, T10, **T11**, NT	T4, T8

* Word, Fast, Att and FL refer to the Word2vec embedding, the Fasttext embedding, Attention and the Focal loss function, respectively.
* The best results are highlighted in bold.

second only to **authentication** words, indicating that the word weights learned by our model can effectively represent the tactical vulnerability pattern.

TABLE IV
EXAMPLE OF ITACTIVUL EXTRACTING ATTENTION WEIGHT VISUALIZATION

CVE ID: CVE-2019-0543
CWE ID: CWE-287 (Improper Authentication)
Tactical Type: Authenticate Actors
Vulnerability description: An elevation of privilege vulnerability exists when Windows improperly handles authentication requests , aka " Microsoft Windows Elevation of Privilege Vulnerability." This affects Windows 7, Windows Server 2012 R2, Windows RT 8.1, Windows Server 2008, Windows Server 2019, Windows Server 2012, Windows 8.1, Windows Server 2016, Windows Server 2008 R2, Windows 10, Windows 10 Servers.

* The darker the word, the higher the attention weight value.

0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

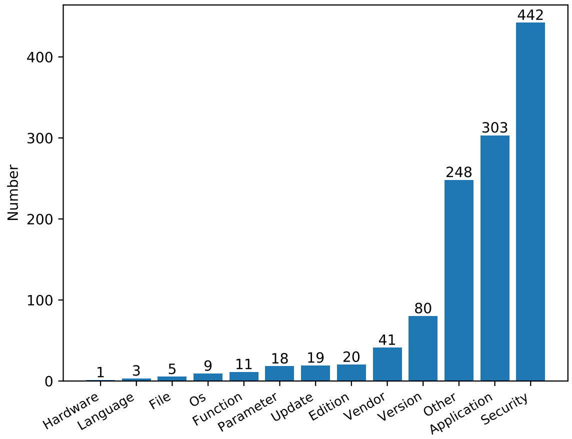


Fig. 3. Distribution of category of pattern key phrases. *Other* refer to key phrases that cannot be classified.

To obtain the pattern key phrases of each tactical category, we summarized the pattern phrases of different tactics according to the sum and ranking of the weight words. TABLE V presents the top 10 key phrases for each tactical vulnerability. We can observe that these key phrases are closely related to the triggering conditions of different levels of tactical vulnerability. For example, the key phrases of *Encrypt Data tactical vulnerability* are related to data generation (random), storage (stores), encryption (password, unencrypted, cleartext, encryption, cryptographic), and verification (credentials, signature, authentication). For *non-tactical vulnerability (NT)*, the key phrases are mainly about coding weakness phrases, such as buffer overflow, out-of-bounds, memory, etc. We expect that the summarized tactical pattern can be used to improve the performance of identifying bug reports with tactical vulnerabilities and deepen developers' awareness of tactical vulnerabilities.

In addition, to understand the characteristics of these patterns more comprehensively, we extracted the top 100 key phrases of each tactical vulnerability to analyze the distribution of these patterns. Fig. 3 shows the statistics of pattern key phrases category. In this classification process, 1200 features

TABLE V
TOP-10 CHARACTERISTICS OF TACTICAL VULNERABILITY PATTERN KEY WORD AND PHRASES

Num	Tactical	Pattern Key Phrase
T1	Audit	log, password, information, sensitive, authentication, credentials, allows, files, disclosure, exposure
T2	Authenticate Actors	authentication, bypass authentication, allows, password, hard-coded, remote attackers, bypass, improper, session, credentials
T3	Authorize Actors	upload, allows, permissions, authorization, vulnerability, access, permission, dll, incorrect, improper
T4	Cross Cutting	check, improper, exception, vulnerability, conditions, uncaught, error, exists, denial, allows
T5	Encrypt Data	credentials, unencrypted, password, cleartext, encryption, signature, stores, authentication, cryptographic, random
T6	Identify Actors	verify, certificate, x.509, ssl, validation, validate, tls, verification, allows, authentication
T7	Limit Access	xxe, entity, external, xml, vulnerability, injection, attack, deputy, error, allows
T8	Limit Exposure	inclusion, allows, insecure, file, vulnerability, flaw, uses, error, local, unintended
T9	Manage User Sessions	session, fixation, authentication, vulnerability, authorization, allows, improper, expiration, insufficient, validation
T10	Validate Inputs	xss, allows, vulnerability, cross-site scripting, csrf, via, sql injection vulnerability, remote attackers, inject, injection
T11	Verify Message Integrity	exception, vulnerability, uncaught, error, allows, check, exists, improper, validate, authentication
NT	Non-tactical	allows, buffer overflow, via, vulnerability, remote attackers, corruption, verify, out-of-bounds, memory, aka

were independently labeled by two Ph.D. students with rich security experience. Then they met to discuss the inconsistent marks and finally determined the classification results. The marked kappa coefficient is 0.87, which indicates that the initial classification is effective. We can see that security-relevant key phrases (442) have the most number, followed by key phrases related to the application (248) and version (80). The results of this pattern classification show that our approach can extract the key feature patterns (security-relevant) of tactical vulnerabilities. In addition, our approach can also understand the potential connection between vulnerabilities and external information (such as applications and versions) to extract detailed feature patterns.

Summary for RQ3: The tactical vulnerability pattern learned by Itactivul is effective and has tactical relevance. In addition, Itactivul can also identify potential tactical patterns.

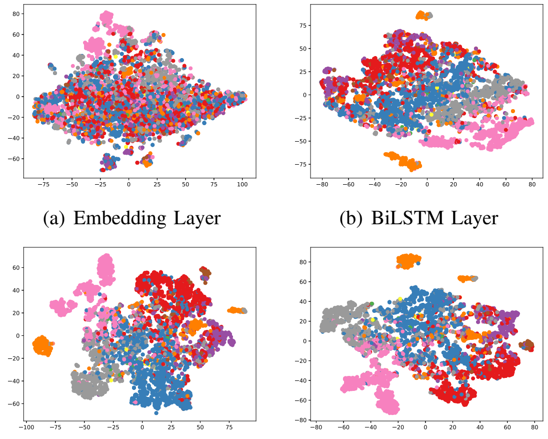
VI. DISCUSSION

A. How about the performance of various hidden layers of our method?

The neural network is a "black box", given the input and parameters, we can observe its output. In this work, we want to explore the learning performance of the hidden layer of our approach instead of blindly trusting the classification results. To this end, we use t-SNE to visualize the features of the hidden layers. This explainability can deepen our understanding and judgment of our model. t-SNE is a popular non-linear dimensionality reduction technology that can realize the visualization of high-dimensional data in low-dimensional feature spaces [50].

Specifically, we first extracted the feature vector output from various hidden layers, including embedding layer, BiLSTM layer, Attention Layer, and the penultimate fully connected layer. Then we use t-SNE to reduce the dimension of the high-dimensional feature vector learned by the model to achieve visualization. These feature vectors represent new feature representations learned by the various hidden layer. The clearer their separation in the feature space, the more effective the learned features.

Fig. 4 illustrates the results of t-SNE 2-dimensional visualization. We observe that FC layer embedding feature vectors is easier to distinguish than other hidden layers. We can also observe that as the hidden layer deepens (Fig. 4 (a) to Fig. 4 (d)), the class separation in the feature space becomes clearer. The results show that our approach can learn more discriminative tactical embeddings features. Different hidden layers make the model's ability to learn features better.



(a) Embedding Layer

(b) BiLSTM Layer

(c) Attention Layer

(d) FC Layer

Fig. 4. The t-SNE visualization of various layers word embeddings learned from NVD test dataset.

B. Threats to Validity

In this section, we analyze the main threats to the validity of our research.

Internal Validity. The threat of internal validity is related to our code implementation and parameter settings. To minimize this internal threat, we checked and tested our source code in detail and made full use of third-party implementations. Besides, we perform hyperparameter optimization and try different parameters so that we can get the best model. We have also publicly published our source code, datasets and detailed our experimental parameters in this study so that other researchers can replicate and extend our work.

External Validity. The threat to our experiment's external validity comes from the quality and generalization ability of the datasets. The training dataset we use is the vulnerability descriptions of the NVD website, which belongs to different types of vulnerabilities, covers a long time, and comes from different software projects and programming languages. However, these data only come from the NVD website. To verify the generalization ability of our proposed method in bug reports, we demonstrated whether our method could automatically classify Chromium, PHP, and Thunderbird bug reports.

Construct Validity. The threat of construct validity is related to the evaluation metrics used in our experiment. We use five popular performance metrics to evaluate the performance of our proposed method and the baseline model. In addition, in order to evaluate whether the architectural tactical vulnerability characteristics learned by our model from the vulnerability description can be applied to bug reports. We test the effectiveness of our method on the open source bug report datasets. These datasets come from publicly published papers, and the CWE corresponding to the security bug report is marked.

VII. RELATED WORK

In this section, we mainly analyze related studies to our work, such as software architecture for security, text mining-based security weakness detection.

A. Software Architectural Vulnerabilities

Some studies have conducted empirical research on architecture security vulnerabilities [47], [51]–[54], and Feng et al. [55] verified that architecture design vulnerabilities and security vulnerabilities are highly correlated through ten open-source projects, which laid the foundation for our in-depth understanding and development of automated tools. To ensure the security of the software architecture, Gepalakrishnan et al. [56] proposed an approach to recommend suitable architecture tactics based on the latent topical in source code.

The design, implementation, and modification of the architecture tactics will inevitably lead to architecture weakness. Santos et al. [4] proposed a method to map the CAWE-Models to the software architecture model, which can detect architecture weakness in the design phase of critical system. Mehdi and Jane [57] apply machine learning methods to discover architectural tactics in the code and monitor sensitive areas of the code to guide changes to prevent architecture deterioration. Keim et al. [45] use BERT to check whether the architectural tactics in the code are implemented correctly.

Different from the above research that actively detects the tactical vulnerabilities in the code. In this study, we focus on identifying bug reports with tactical vulnerabilities. Bug reports record tactical vulnerabilities that have been discovered during the testing and maintenance process. The tactical vulnerabilities in the bug report need to be identified and prioritized on time.

B. Text Mining-based Security Weakness Detection

As security issues have received more attention, researchers in software engineering have done a lot of research to use machine learning to automatically identify security-related text (such as bug reports, vulnerability descriptions, commits). Zhang et al. [58] explained the importance of bug report analysis and possible problems. Sabetta et al. [59] use machine learning to identify security-related commits in source code repositories automatically. Similarly, Peter et al. [21] use text filtering and sorting to detect security bug reports automatically. Shu et al. [22] improved the security bug report detection performance of Peter et al. [21] through hyperparameter optimization. Furthermore, Hogan et al. [60] studied the challenge of label vulnerability-contributing commits (VCCs) and proposed an approach to label VCCs according to the commits listed in CVE manually.

The mining and extraction of vulnerability description information have received a lot of attention. Han et al. [61] use the CNN model to mine the relationship between the vulnerability description and the security level. Palacio et al. [62] use vulnerability description learning word embedding for the identification of open source software security issues. Li et al. [63] show that text mining methods can be used to mine vulnerability features. Many existing studies have shown that different representation characteristics can effectively improve the performance of data-driven tasks, such as Android malware clustering [64] and code Summarization [65]. Therefore, it is valuable to explore whether the vulnerability description characteristics can be exploited for tactical vulnerability bug report identification.

For this study, Itactivul is a fine-grained classification for identifying bug reports with tactical vulnerabilities. Specifically, we trained a deep learning model to exploit the characteristics of architectural tactics described by vulnerabilities. Thus, the model is applied to identify bug reports with different tactical vulnerabilities.

VIII. CONCLUSION

In this paper, we propose Itactivul that uses deep learning to identify bug reports with tactical vulnerabilities automatically. First, we exploit attention-based BiLSTM to capture text features from the NVD vulnerability descriptions automatically. After that, we use attention weight to extract key phrases that are most relevant to tactical vulnerability. Manual analysis shows that the features learned by Itactivul are effective. Finally, we verify Itactivul identification ability on Chromium, PHP, and Thunderbird bug report datasets.

ACKNOWLEDGMENT

This work was supported in part by the Key Research and Development Program of Shaanxi 2021GY-041, in part by the Key Laboratory of Advanced Perception and Intelligent Control of High-end Equipment, Ministry of Education GDSC202006, in part by the seed Foundation of Innovation and Creation for Graduate Students in Northwestern Polytechnical University CX2020246.

References

[1] L. Braz, E. Fregnan, G. Çalikli, and A. Bacchelli, "Why don't developers detect improper input validation?'; drop table papers;–," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 499–511.

[2] A. Sejfia and N. Medvidović, "Strategies for pattern-based detection of architecturally-relevant software vulnerabilities," in *2020 IEEE International Conference on Software Architecture (ICSA)*. IEEE, 2020, pp. 92–102.

[3] J. C. Santos, A. Sejfia, T. Corrello, S. Gadenkanahalli, and M. Mirakhorli, "Achilles' heel of plug-and-play software architectures: a grounded theory based approach," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 671–682.

[4] J. C. Santos, S. Suloglu, J. Ye, and M. Mirakhorli, "Towards an automated approach for detecting architectural weaknesses in critical systems," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, 2020, pp. 250–253.

[5] J. C. Santos, K. Tarrit, A. Sejfia, M. Mirakhorli, and M. Galster, "An empirical study of tactical vulnerabilities," *Journal of Systems and Software*, vol. 149, pp. 263–284, 2019.

[6] X. Chen, Z. Yuan, Z. Cui, D. Zhang, and X. Ju, "Empirical studies on the impact of filter-based ranking feature selection on security vulnerability prediction," *IET Software*, 2020.

[7] X. Chen, Y. Zhao, Z. Cui, G. Meng, Y. Liu, and Z. Wang, "Large-scale empirical studies on effort-aware security vulnerability prediction methods," *IEEE Transactions on Reliability*, vol. 69, no. 1, pp. 70–87, 2019.

[8] G. McGraw, *Software Security: Building Security In*. Addison-Wesley Professional, 2006.

[9] K. Sahu, R. Shree, and R. Kumar, "Risk management perspective in sdlc," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 4, no. 3, 2014.

[10] R. Kumar, S. A. Khan, and R. A. Khan, "Revisiting software security risks," *Journal of Advances in Mathematics and Computer Science*, pp. 1–10, 2015.

[11] A. Agrawal, M. Alenezi, D. Pandey, R. Kumar, and R. A. Khan, "Usable-security assessment through a decision making procedure," *ICIC Express Letters*, vol. 10, no. 8, pp. 665–672, 2019.

[12] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Addison-Wesley Professional, 2003.

[13] M. Alenezi, A. K. Pandey, R. Verma, M. Faizan, S. Chandra, A. Agrawal, R. Kumar, and R. A. Khan, "Evaluating the impact of software security tactics: A design perspective."

[14] M. Mirakhorli, Y. Shin, J. Cleland-Huang, and M. Cinar, "A tactic-centric approach for automating traceability of quality concerns," in *2012 34th international conference on software engineering (ICSE)*. IEEE, 2012, pp. 639–649.

[15] J. Van Gurp, S. Brinkkemper, and J. Bosch, "Design preservation over subsequent releases of a software product: a case study of baan erp," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 17, no. 4, pp. 277–306, 2005.

[16] C. Izurieta and J. M. Bieman, "How software designs decay: A pilot study of pattern evolution," in *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. IEEE, 2007, pp. 449–451.

[17] M. Mirakhorli and J. Cleland-Huang, "Modifications, tweaks, and bug fixes in architectural tactics," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 377–380.

[18] J. C. Santos, A. Peruma, M. Mirakhorli, M. Galstery, J. V. Vidal, and A. Sejfia, "Understanding software vulnerabilities related to architectural security tactics: An empirical investigation of chromium, php and thunderbird," in *2017 IEEE International Conference on Software Architecture (ICSA)*. IEEE, 2017, pp. 69–78.

[19] S. C. Mayana Pereira, "Identifying security bug reports based solely on report titles and noisy data," https://docs.microsoft.com/en-us/security/engineering/identifying-security-bug-reports.

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

[21] F. Peters, T. T. Tun, Y. Yu, and B. Nuseibeh, "Text filtering and ranking for security bug report prediction," *IEEE Transactions on Software Engineering*, vol. 45, no. 6, pp. 615–631, 2019.

[22] R. Shu, T. Xia, L. Williams, and T. Menzies, "Better security bug report classification via hyperparameter optimization," *CoRR*, vol. abs/1905.06872, 2019. [Online]. Available: http://arxiv.org/abs/1905.06872

[23] X. Wu, W. Zheng, X. Chen, Y. Zhao, T. Yu, and D. Mu, "Improving high-impact bug report prediction with combination of interactive machine learning and active learning," *Information and Software Technology*, vol. 133, p. 106530, 2021.

[24] X. Wu, W. Zheng, X. Chen, F. Wang, and D. Mu, "Cve-assisted large-scale security bug report dataset construction method," *Journal of Systems and Software*, vol. 160, p. 110456, 2020.

[25] W. Zheng, J. Gao, X. Wu, F. Liu, Y. Xun, G. Liu, and X. Chen, "The impact factors on the performance of machine learning-based vulnerability detection: A comparative study," *Journal of Systems and Software*, vol. 168, p. 110659, 2020.

[26] I. C. for Secure Design, "Avoiding the top 10 software security design flaws," http://cybersecurity.ieee.org/center-for-secure-design/.

[27] C. Williams, "Kernel-memory-leaking intel processor design flaw forces linux, windows redesign," https://www.theregister.com/2018/01/02/intel_cpu_design_flaw/.

[28] F. A. Bhuiyan, R. Shakya, and A. Rahman, "Can we use software bug reports to identify vulnerability discovery strategies?" in *Proceedings of the 7th Symposium on Hot Topics in the Science of Security*, 2020, pp. 1–10.

[29] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[30] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *arXiv preprint arXiv:1607.01759*, 2016.

[31] J. C. S. Santos, K. Tarrit, and M. Mirakhorli, "A catalog of security architecture weaknesses," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, 2017, pp. 220–223.

[32] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed. Addison-Wesley Professional, 2012.

[33] Y. Sui, X. Cheng, G. Zhang, and H. Wang, "Flow2vec: Value-flow-based precise code embedding," *Proceedings of the ACM on Programming Languages*, vol. 4, no. OOPSLA, pp. 1–27, 2020.

[34] X. Chen, C. Chen, D. Zhang, and Z. Xing, "Sethesaurus: Wordnet in software engineering," *IEEE Transactions on Software Engineering*, 2019.

[35] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[36] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[37] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Association for Computational Linguistics, April 2017, pp. 427–431.

[38] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[39] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 2650–2658.

[40] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[41] H. Binyamini, R. Bitton, M. Inokuchi, T. Yagyu, Y. Elovici, and A. Shabtai, "An automated, end-to-end framework for modeling attacks from vulnerability descriptions," *arXiv preprint arXiv:2008.04377*, 2020.

[42] M. Tikhomirov, N. Loukachevitch, A. Sirotina, and B. Dobrov, "Using bert and augmentation in named entity recognition for cybersecurity domain," in *International Conference on Applications of Natural Language to Information Systems*. Springer, 2020, pp. 16–24.

[43] K. Simran, S. Sriram, R. Vinayakumar, and K. Soman, "Deep learning approach for intelligent named entity recognition of cyber security," in

International Symposium on Signal Processing and Intelligent Recognition Systems. Springer, 2019, pp. 163–172.

[44] H. Gasmi, A. Bouras, and J. Laval, "Lstm recurrent neural networks for cybersecurity named entity recognition," *ICSEA*, vol. 11, p. 2018, 2018.

[45] S. Dasgupta, A. Piplai, A. Kotal, A. Joshi *et al.*, "A comparative study of deep learning based named entity recognition algorithms for cybersecurity," in *4th International Workshop on Big Data Analytics for Cyber Intelligence and Defense, IEEE International Conference on Big Data*, 2020.

[46] R. A. Bridges, C. L. Jones, M. D. Iannacone, K. M. Testa, and J. R. Goodall, "Automatic labeling for entity extraction in cyber security," *arXiv preprint arXiv:1308.4941*, 2013.

[47] J. C. S. Santos, A. Peruma, M. Mirakhorli, M. Galstery, J. V. Vidal, and A. Sejfia, "Understanding software vulnerabilities related to architectural security tactics: An empirical investigation of chromium, php and thunderbird," in *2017 IEEE International Conference on Software Architecture (ICSA)*, 2017, pp. 69–78.

[48] T. Zhang, J. Chen, G. Yang, B. Lee, and X. Luo, "Towards more accurate severity prediction and fixer recommendation of software bugs," *Journal of Systems and Software*, vol. 117, pp. 166–184, 2016.

[49] X. Wu, W. Zheng, X. Xia, and D. Lo, "Data quality matters: A case study on data label correctness for security bug report prediction," *IEEE Transactions on Software Engineering*, 2021.

[50] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.

[51] D. Gonzalez, F. Alhenaki, and M. Mirakhorli, "Architectural security weaknesses in industrial control systems (ics) an empirical study based on disclosed software vulnerabilities," in *2019 IEEE International Conference on Software Architecture (ICSA)*, 2019, pp. 31–40.

[52] M. Mirakhorli, M. Galster, and L. Williams, "Understanding software security from design to deployment," *SIGSOFT Softw. Eng. Notes*, vol. 45, no. 2, p. 25–26, Apr. 2020. [Online]. Available: https://doi.org/10.1145/3385678.3385687

[53] A. Sejfia, "A pilot study on architecture and vulnerabilities: Lessons learned," in *2019 IEEE/ACM 2nd International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE)*, 2019, pp. 42–47.

[54] L. Sion, K. Tuma, R. Scandariato, K. Yskout, and W. Joosen, "Towards automated security design flaw detection," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW)*, 2019, pp. 49–56.

[55] Q. Feng, R. Kazman, Y. Cai, R. Mo, and L. Xiao, "Towards an architecture-centric approach to security analysis," in *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, 2016, pp. 221–230.

[56] R. Gopalakrishnan, P. Sharma, M. Mirakhorli, and M. Galster, "Can latent topics in source code predict missing architectural tactics?" in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, 2017, pp. 15–26.

[57] M. Mirakhorli and J. Cleland-Huang, "Detecting, tracing, and monitoring architectural tactics in code," *IEEE Transactions on Software Engineering*, vol. 42, no. 3, pp. 205–220, 2015.

[58] J. Zhang, X. Wang, D. Hao, B. Xie, L. Zhang, and H. Mei, "A survey on bug-report analysis," *Science China Information Sciences*, vol. 58, no. 2, pp. 1–24, 2015.

[59] A. Sabetta and M. Bezzi, "A practical approach to the automatic classification of security-relevant commits," in *2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2018*. IEEE Computer Society, 2018, pp. 579–582. [Online]. Available: https://doi.org/10.1109/ICSME.2018.00058

[60] K. Hogan, N. Warford, R. Morrison, D. Miller, S. Malone, and J. Purtilo, "The challenges of labeling vulnerability-contributing commits," in *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2019, pp. 270–275.

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

[62] D. N. Palacio, D. McCrystal, K. Moran, C. Bernal-Cárdenas, D. Poshyvanyk, and C. Shenefiel, "Learning to identify security-related issues using convolutional neural networks," in *2019 IEEE International conference on software maintenance and evolution (ICSME)*. IEEE, 2019, pp. 140–144.

[63] X. Li, J. Chen, Z. Lin, L. Zhang, Z. Wang, M. Zhou, and W. Xie, "A mining approach to obtain the software vulnerability characteristics," in *2017 Fifth International Conference on Advanced Cloud and Big Data (CBD)*. IEEE, 2017, pp. 296–301.

[64] Y. Zhang, Y. Sui, S. Pan, Z. Zheng, B. Ning, I. Tsang, and W. Zhou, "Familial clustering for weakly-labeled android malware using hybrid representation learning," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3401–3414, 2019.

[65] W. Wang, Y. Zhang, Y. Sui, Y. Wan, Z. Zhao, J. Wu, P. Yu, and G. Xu, "Reinforcement-learning-guided source code summarization via hierarchical attention," *IEEE Transactions on software Engineering*, 2020.