Automated Unearthing of Dangerous Issue Reports

Shengyi Pan
Zhejiang University
Hangzhou, Zhejiang, China
shengyi.pan@zju.edu.cn

Jiayuan Zhou
Centre for Software Excellence, Huawei
Kingston, Ontario, Canada
jiayuan.zhou1@huawei.com

Filipe Roseiro Cogo
Centre for Software Excellence, Huawei
Kingston, Ontario, Canada
filipe.roseiro.cogo1@huawei.com

Xin Xia
Huawei
Hangzhou, Zhejiang, China
xin.xia@acm.org

Lingfeng Bao[*]
Zhejiang University
Hangzhou, Zhejiang, China
lingfengbao@zju.edu.cn

Xing Hu
Zhejiang University
Ningbo, Zhejiang, China
xinghu@zju.edu.cn

Shanping Li
Zhejiang University
Hangzhou, Zhejiang, China
shan@zju.edu.cn

Ahmed E. Hassan
Queen's University
Kingston, Ontario, Canada
ahmed@cs.queensu.ca

ABSTRACT

The coordinated vulnerability disclosure (CVD) process is commonly adopted for open source software (OSS) vulnerability management, which suggests to privately report the discovered vulnerabilities and keep relevant information secret until the official disclosure. However, in practice, due to various reasons (e.g., lacking security domain expertise or the sense of security management), many vulnerabilities are first reported via public issue reports (IRs) before its official disclosure. Such IRs are dangerous IRs, since attackers can take advantages of the leaked vulnerability information to launch zero-day attacks. It is crucial to identify such dangerous IRs at an early stage, such that OSS users can start the vulnerability remediation process earlier and OSS maintainers can timely manage the dangerous IRs. In this paper, we propose and evaluate a deep learning based approach, namely MemVul, to automatically identify dangerous IRs at the time they are reported. MemVul augments the neural networks with a memory component, which stores the external vulnerability knowledge from Common Weakness Enumeration (CWE). We rely on publicly accessible CVE-referred IRs (CIRs) to operationalize the concept of dangerous IR. We mine 3,937 CIRs distributed across 1,390 OSS projects hosted on GitHub. Evaluated under a practical scenario of high data imbalance, MemVul achieves the best trade-off between precision and recall among all baselines. In particular, the F1-score of MemVul (i.e., 0.49) improves the best performing baseline by 44%. For IRs that are predicted as CIRs but not reported to CVE, we conduct a user study to investigate their usefulness to OSS stakeholders. We observe that 82% (41 out of 50) of these IRs are security-related and 28 of them are suggested by security experts to be publicly disclosed, indicating MemVul is capable of identifying undisclosed dangerous IRs.

CCS CONCEPTS

• **Security and privacy → Software security engineering**.

KEYWORDS

Software Security, Vulnerability, Issue Report, Deep Learning

ACM Reference Format:
Shengyi Pan, Jiayuan Zhou, Filipe Roseiro Cogo, Xin Xia, Lingfeng Bao, Xing Hu, Shanping Li, and Ahmed E. Hassan. 2022. Automated Unearthing of Dangerous Issue Reports. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3540250.3549156

1 INTRODUCTION

Open source software (OSS) is widely used by the software industry [35], from small-scale projects to critical applications deployed in global infrastructures. Despite its benefits [64], the usage of OSS also incurs on security risks [41, 60, 71] that can lead to negative impacts to individuals and organizations [3]. To better manage OSS vulnerabilities and minimize the risk of exploits, the coordinated vulnerability disclosure (CVD) process [2, 46] is commonly adopted [4, 7, 22]. The CVD suggests to privately report the discovered vulnerabilities to OSS maintainers, so that relevant information is not publicly available until the maintainers are ready for the disclosure (e.g., a patch is available) [4].

In practice, however, due to various reasons (e.g., lacking security domain expertise [42] or the sense of security management [32]), many IRs that report vulnerabilities are first submitted to publicly accessible issue tracking systems (ITS), and then the associated vulnerabilities are officially disclosed. Such vulnerability IRs are dangerous IRs, as the time gap between the IR creation date and the vulnerability disclosure date provides a window of opportunity for attackers to develop and deploy exploits (e.g., zero-day

[*] Corresponding Author.

ESEC/FSE '22, November 14–18, 2022, Singapore, Singapore
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9413-0/22/11…$15.00
https://doi.org/10.1145/3540250.3549156

attacks [31]). In particular, attackers can take advantage of the leaked vulnerability information (e.g., the reproducible steps of an attack), which puts OSS users at a great disadvantage in defending against exploitation. For instance, many vulnerability IRs are first submitted to the GitHub ITS, then referred by a Common Vulnerabilities and Exposure (CVE) record that officially discloses the vulnerability [62, 77]. Such CVE-referred IRs (CIRs) are a type of typical dangerous IRs, as the leaked vulnerability information are earlier available with a median time of 13 days (see Section 2.3), and the reported vulnerability details provides an advantage for attackers (see Section 2.2 for a motivation example).

It is crucial for OSS users, especially downstream software vendors, to identify dangerous IRs at an early stage so that they can take an active role in the race against attackers (i.e., the race between patching and exploiting vulnerabilities). With the ability of early sensing dangerous IRs, vendors can start the remediation process earlier, instead of waiting for the official disclosure and then rushing to remediate. Since one project may rely on hundreds of OSS projects [35] and it takes too much work from OSS users to monitor every IR in the upstream OSS projects, an automated solution is necessary to identify dangerous IRs.

Furthermore, it is also important for ITS platforms to implement mechanisms to identify emerging dangerous IRs. ITS platforms, as a service provider, should take the responsibility of better managing vulnerability reports to comply with the CVD process, reducing the threat of dangerous IRs and consequently making the OSS ecosystem safer. For example, by identifying dangerous IRs in advance, ITS platforms can warn reporters before posting the IR and suggest them to privately report to the maintainers instead. Moreover, ITS platforms can automatically hide the existing dangerous IRs from the public channel, and flag them as priorities for maintainers.

The goal of our work is to provide a tool that can automatically unearth the emerging dangerous IRs at their early stage. Given the prevalent usage of CVE in OSS vulnerability management, and the popularity of GitHub ITS in OSS IR management, we operationalize dangerous IRs as CIRs. In this paper, we take a first step to explore the characteristics of CIRs and to propose an automated approach to identify them at their creation time. We first build a large-scale dataset consisting of 1,221,677 IRs from 1,390 OSS systems hosted on GitHub, with 3,937 CIRs in total. The number of CIRs in our dataset is larger and more up-to-date than those of the five datasets (351 in total, collected before 2015) as proposed by existing studies [62, 67, 78]. Moreover, the existing datasets are limited in project-scope, which hinders the construction and evaluation of a generic approach. The small proportion of CIRs (0.3%) agrees with real-world distributions and shows the challenge of unearthing dangerous IRs.

Our preliminary study shows that ❶ 98.5% of the CIRs are created before the disclosure of the corresponding CVE records, suggesting that early leakage of vulnerability information through a public IR is very common. ❷ CIRs account for only 0.3% of all IRs, indicating that there is a limited vulnerability knowledge for data-driven models. Moreover, all CIRs belong to 132 different CWE vulnerability types, which can have diverse causes, behaviours, and consequences. It is challenging to predict CIRs under such a highly imbalanced class distribution and diverse vulnerability types, particularly by using existing machine learning approaches that are adopted from similar tasks (e.g., triaging security bug reports [42, 62, 77, 78, 81]).

To cope with these challenges, we propose MemVul, a deep learning based approach that leverages language models [40] and a **mem**ory component to incorporate the external **vul**nerability knowledge from CWE (Common Weakness Enumeration): a curated classification schema for describing vulnerabilities. The memory component is designed to act as a vulnerability knowledge base during model inference. MemVul predicts whether an input IR is a CIR by matching the description of the IR against the knowledge base. To evaluate the effectiveness of MemVul, we conduct experiments under a practical scenario where the proportion of CIRs (0.3%) agrees with the real-world distribution. Evaluation results show that our approach achieves an F1-score of 0.49, which outperforms the best baseline by 44%. With an ablation study, we further verify the effectiveness of incorporating the memory component as a knowledge base of vulnerabilities. In summary, our paper makes the following contributions:

- We are the first to introduce the task of unearthing dangerous IRs at their early stage. We build a real-world dataset with larger and more-up-date security-related IR data, i.e., 3937 CIRs from 1,390 OSS systems. We consider IRs linked to CVE records as CIRs, and use them as a proxy to dangerous IRs.
- We propose MemVul, a deep learning based approach that uses language models and incorporates external vulnerability knowledge from CWE, improving the identification of CIRs.
- We evaluate MemVul under a practical scenario of extreme data imbalance. MemVul achieves an F1-score of 0.49, improving the best performing baseline by 44%.
- To foster future work and in line with good research practices, we provide a complete replication package of our approach and experiments [25].

2 MOTIVATION AND PRELIMINARIES

In this section, we first introduce the background of our paper (Section 2.1). Then, we provide a motivation example (Section 2.2). Finally, we conduct a preliminary study on CIRs (Section 2.3).

2.1 Background

Common Vulnerabilities and Exposure (CVE): CVE provides a standardized method to identify, define and catalog publicly disclosed software vulnerabilities [9]. Once a vulnerability is discovered, developers can request a CVE ID from the CVE Numbering Authority (e.g., MITRE Corporation). The ID being reserved is regarded as the initial state of a CVE record. However, the record is not publicly available until its publication to the CVE list, when the minimum required details are prepared. Except for the CVE ID, each CVE record includes a brief and project-oriented description of the vulnerability and the related references (i.e., a list of URLs). When constructing our dataset, we use the URLs from the references to identify the corresponding GitHub CIRs.

Common Weakness Enumeration (CWE): CWE serves as a common language for discussing and describing weaknesses [10]. Each CWE entry represents a single vulnerability type, providing detailed information including the common causes, behaviours, and consequences. CWE entries are organized in a tree-like structure of multiple levels of abstraction.

Table 1: An example of vulnerability that was publicly posted as a GitHub IR, prior to its disclosure.

Report date: Apr 24, 2019
Issue title: LDAP connector does not verify TLS certificates
Labels: bug, ldap, triaged
Issue body:

Expected Behavior: Graylog should verify the LDAP server certificate chain up to a trusted root, and refuse the connection when the certificate chain cannot be verified.

Current Behavior: Graylog accepts LDAP server certificates whose root certificate is not in any trust store. This presents a vulnerability for man-in-the-middle attacks.

Steps to Reproduce (for bugs):

- 1. Navigate to the LDAP / Active Directory configuration page.

......

Context: We run Graylog in a Docker container, using the official Docker image. Our LDAP server is on a different network. We would be better protected against man-in-the-middle attacks if Graylog verifies the LDAP server certificate.

Your Environment:

<Environment settings>

Disclosure date: Jul. 17, 2020
CVE ID: CVE-2020-15813
CVE description (key information):

Graylog before 3.3.3 lacks SSL Certificate Validation for LDAP servers. ...... Therefore, any attacker with the ability to intercept network traffic between a Graylog server and an LDAP server is able to redirect traffic to a different LDAP server ...... , effectively bypassing Graylog’s authentication mechanism.

CVSS v2.0 exploitability score: 8.6
CVSS v2.0 severity ratings: Medium

Note that the detailed environment setting is replaced with a placeholder.

National Vulnerability Database (NVD): NVD [24] is one of the most popular security advisories [77]. NVD fully syncs with CVE list, i.e., once a CVE record is published, it will appear in NVD immediately. Upon the information included in CVE records, NVD provides enhanced vulnerability information including the severity score rankings, the exploitation scores, and the weakness type (CWE). NVD uses CWE as a categorization mechanism to differentiate CVEs according to their vulnerability type. NVD is a widely used public source for new OSS vulnerabilities.

2.2 Motivation Example

Table 1 shows an example of vulnerability that was publicly posted as a GitHub IR [16] prior to its disclosure as a CVE record [55] (CVE-2020-15813). CVE-2020-15813 is an “Improper Certificate Validation” type vulnerability with an exploitability score of 8.6 and a medium level of severity rating. This vulnerability allows an attacker to bypass the authentication mechanism of graylog2/graylog2-server project, which is a widely used (6,000+ GitHub stars) OSS log management system. The vulnerability was disclosed via NVD on Jul. 17, 2020, while it was first reported and publicly available 450 days earlier (Apr. 24, 2019) on GitHub ITS.

This CIR was dangerous as it leaked sensitive vulnerability information, giving hackers a chance to launch attacks when the general public was unprepared or even unaware. Moreover, the CIR described the detailed information of the vulnerability, including the expected behavior, current behavior, and reproduction steps. These information provided attackers with advantages in creating a functioning exploit. According to CVD, such vulnerability information should not be publicly available until the official disclosure when the general public can start the remediation process. However, due to the lack of security domain expertise or the sense of security management, the end-user directly reported the vulnerability to the public ITS, leading to a dangerous issue report. Even worse, the maintainers let the CIR be publicly available for more than one year before the NVD disclosure, leaving potential attackers with a large time window to exploit the vulnerability.

Table 2: Number of IRs that contain vulnerability patterns.

Issue report type	#Issue reports	
	Contain vuln. patterns	No vuln. patterns
CVE-referred IR	3,271	666
Not CVE-referred IR	189,545	1,028,195

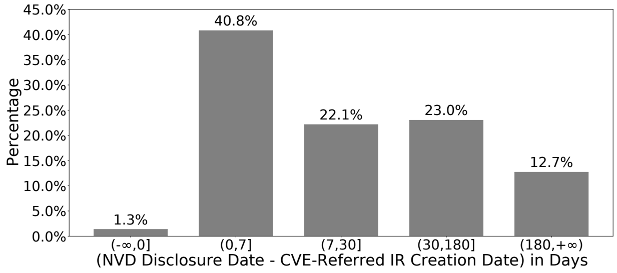

Figure 1: The distribution of delta days between the creation date of CIRs and the NVD disclosure date of the corresponding vulnerabilities.

It is crucial to identify such CIRs at an early stage. So that OSS users can take an active role in the race against potential attackers. Specifically, OSS users can sense the threat and take remediations in time, instead of waiting for the NVD disclosure and then hurry to remediate. Also, ITSs can warn reporters before posting a possible CIR, and help OSS maintainers to handle them appropriately by flagging CIRs as priorities and restricting the access to the IR from the public. With such mechanisms, ITS platforms can provide OSS stakeholders a safer ecosystem.

2.3 Preliminary Study

We build a large-scale dataset (see Section 4.2) for understanding the characteristics of CIRs and to evaluate our approach, which consists of 1,221,677 GitHub IRs from 1,390 GitHub projects. We consider IRs that are referred by CVE records as CIRs (3,937 in total), and the remaining IRs as not CVE-referred IRs (NCIRs). In this section, we perform a preliminary study using the collected dataset to show the importance and challenges of identifying and flagging potential CIRs at their creation time.

The majority (98.7%) of CIRs are created before the corresponding NVD disclosure date, exposing OSS users to unperceived security risks. Figure 1 shows the distribution of the number of days between the CIR creation and the NVD disclosure. We observe that only 1.3% of the CIRs were created after the associated vulnerability was disclosed via NVD. This result shows that

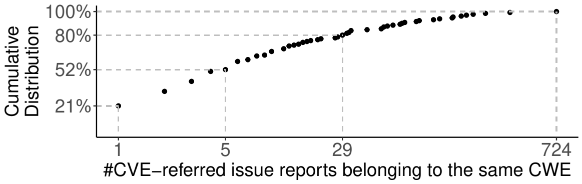

Figure 2: ECDF of #CIRs that belong to the same CWE.

most of the CIRs constitute the leakage of sensitive vulnerability information, providing attackers with advantages while exposing OSS users to huge security risks. For the remaining 98.7% (3,884 out of 3,937) of CIRs, the median time between the IR creation and the NVD disclosure is 13 days. The median exploitability score of these CIRs is 8.6/10 and 16.5% of the CIRs are rated as high severity vulnerabilities. Moreover, the average #Stars and #Forks of the OSS projects that these CIRs belong to are 3,895 and 1,024, respectively. These results suggest that CIRs are associated with potentially impactful vulnerabilities. Moreover, we estimate that 39.9% (1,570/3,937) of the CIRs in our dataset contain the steps of an attack. We manually analyzed a random sample of the CIRs and derived a set of keywords to identify CIRs that contain attack steps (i.e., "steps to reproduce", "steps to replicate", "proof-of-concept", "proof of concept", and "poc"). To validate our approach, we manually analyzed 50 random CIRs that contain at least one of our keywords and verified that all describe attack steps. Since some steps are described in the implicit way (e.g., "what did you do"), our result provides the lower bound estimation. Thus, instead of leaving a time window for attackers to launch zero-day attacks using the leaked vulnerability information in the CIR, **it is crucial to identify CIRs at an early stage so that the OSS users can start the remediation process earlier, and thus closing the dangerous timegap.**

Token-level keyword-based approaches are commonly used in the task of triaging security bug reports [43, 56, 62, 81]. Prior work [81] proposed regular expressions to identify security-related IRs that contain vulnerability-related patterns (i.e., 55 security keywords). For example, an IR is classified as security-related if it contains words such as "xss" (i.e., Cross-site scripting) and "malicious". We apply the same regular expression on the title and body of IRs in our dataset to match those containing vulnerability patterns. Table 2 shows the results of matching. We observe that 666 (17%) CIRs do not contain strong vulnerability patterns while 189,545 NCIRs also contain strong vulnerability patterns. Directly applying this keyword-based matching approach will result in an extremely low precision of 1.7%. **This result indicates that it is challenging to leverage token-level vulnerability-pattern-based approaches to identify CIRs with high precision and recall.**

CIRs are typically scarce among all IRs of OSS projects. In our dataset (Section 4.2), the number of NCIRs (1,217,740) significantly outweighs the number of CIRs (3,937). The data imbalance and limited amount of CIR data prevent data-driven models from effectively learning the associated patterns with such CIRs. Furthermore, the 3,937 CIRs in our dataset belong to 132 different CWE categories. Figure 2 shows the empirical cumulative distribution function (ECDF) of the number of CIRs that belong to the same CWE category. We observe that 80% of the CWE categories have less than 30 CIRs, and more than half of the CWE categories have only 5 CIRs. Moreover, the causes, behaviours, and consequences can vary significantly across different CWE categories (i.e., the vulnerability type it represents). For example, CWE-787 [13] regards the buffer overflow vulnerability, while CWE-502 [12] regards the lack of verification in deserialization of untrusted data. **Given the scattered and insufficient knowledge of different types of vulnerabilities, it is difficult for data-driven models to learn effective knowledge about diverse vulnerability types.**

To cope with the aforementioned challenges, we introduce the external vulnerability knowledge from CWE to enhance the internal knowledge learned by our model. We elaborate on the details of our approach in Section 3.

3 APPROACH

In this section, we introduce our approach so-called MemVul. We first introduce the overall architecture for CIR prediction. Then, we describe the details of model training and inference.

3.1 Architecture of the Memory Network

Existing approaches for triaging security related IRs [42, 62, 67, 78] typically train an end-to-end model using the textual description of the IR as input and return a classified IR as output. The input textual description is encoded as a vector of terms, and the model associates weights that describe security related IRs. However, as discussed in our preliminary study (see Section 2.3), the textual descriptions of CIRs that belong to different CWE categories can differ since the vulnerability types, as well as its causes and consequences are different. In addition, the CIRs in our dataset (3,884 in total) belong to 131 different CWE categories, and 81% of the categories have fewer than 30 samples. Hence, training a classifier that directly maps the textual description of an IR to its class (i.e., a binary classification of whether the IR will be referred by a CVE) will fail to learn effective knowledge of most vulnerability types, and will likely suffer from overfitting due to the limited CIR data.

To tackle this challenge, we introduce MemVul (Figure 3), which augments the neural network with an *external memory* [70, 74]. The memory component acts as an external knowledge base, storing the knowledge of different types of vulnerabilities refined and summarized in the CWE entries. By explicitly incorporating the external memory, the model is directly supplied with the knowledge of different vulnerabilities instead of learning from the data only. Moreover, to facilitate training, the model only needs to focus on matching the description of an input IR against the vulnerability types stored in the external memory. Figure 3 presents the overview of MemVul, which includes the following three components:

External Memory. The external memory component stores information of multiple CWE categories, providing domain knowledge about vulnerabilities to the model. The motivation to store the CWE information in the external memory is that the CWEs are well-organized, curated by security experts, and contain a common language for identifying and describing all types of vulnerabilities [10]. Different from the low-level and project-oriented CVE description, vulnerability knowledge presented in CWE is high-level and beyond the specific projects. The external memory is composed of multiple *anchors*, with each anchor storing related information

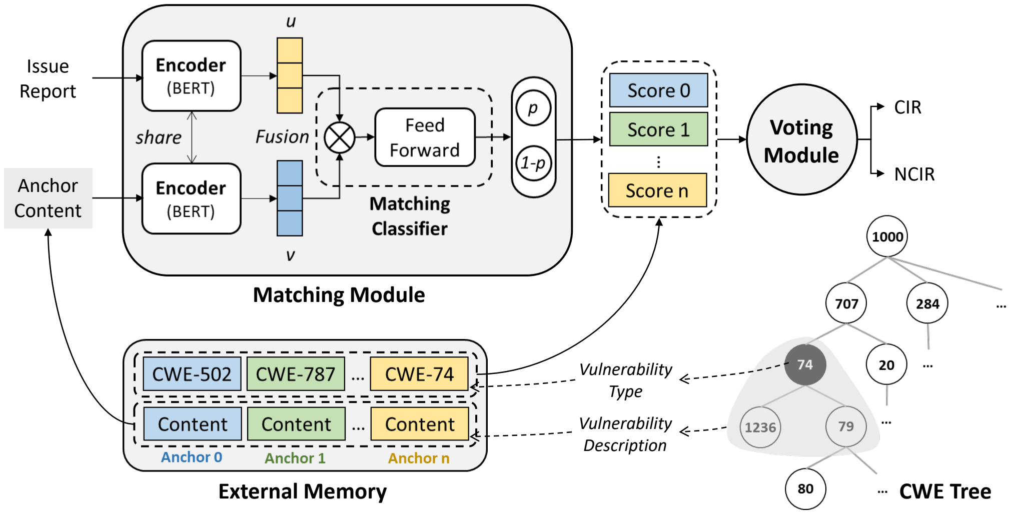

Figure 3: The overview of MemVul.

Table 3: The CWE attributes used to build anchors.

Attribute	Description
Name	The core behaviour of the weakness
Description	Summary of how the weakness is made, e.g., the intended behaviour, the mistake
Extended Description	Additional explanation for the weakness, e.g., why the weakness is a concern
Common Consequences	Typical negative security impacts of the weakness if exploited by an attacker
Related Weaknesses	Relationships with other CWEs, e.g., parent/child CWEs, similar CWEs

to the vulnerability associated with an individual CWE category. It acts as an external knowledge base, which will be accessed by the matching module during inference (Section 3.3). Specifically, we build an anchor for each CWE category that has at least one corresponding CIR in our dataset. By doing so, we introduce the corresponding vulnerability knowledge from CWE to enhance the internal knowledge learned by the model. We summarize the CWE attributes used to build anchors in Table 3. Specifically, we use the attribute *Related Weaknesses* to organize all CWE entries into the tree-like structure. We then merge the descriptions of the remaining four attributes from both the corresponding CWE entry and its direct child entries to generate the content for each anchor (Figure 3). When merging the information of CWE entries, we place CWE with higher abstraction levels (i.e., the parent entry) in front of those lower ones (i.e., the child entries). Thus we always ensure the common information first and add as many details as possible.

Matching Module. The input of the matching module is the target IR and an anchor from the external memory. The output is a normalized *matching score* indicating the extent to which the IR matches the anchor, i.e., the IR describes a vulnerability type presented by the anchor. Our implementation of the matching module uses the Siamese architecture [34] as shown in Figure 3. The Siamese architecture is widely adopted in various software engineering tasks for matching two information items [51, 66, 73, 79]. Besides, as a metric-based few-shot learning technique [72], it can help us tackle the imbalanced dataset challenge and make better use of the limited CIR data. Specifically, our Siamese architecture consists of following two components:

(1) Shared encoder, to convert the two inputs into feature vectors in hidden space. Considering that both *issue report* and *anchor content* are natural language descriptions, we leverage Bidirectional Encoder Representations from Transformers (BERT) [40] as the shared encoder. BERT is a multi-layer bidirectional transformer pretrained on large corpora, which achieves state-of-the-art performance on multiple NLP-related tasks. The embedding of `[CLS]` token is used as the representation of the input text. The *issue report* and *anchor content* are passed sequentially into the shared encoder to get feature vectors u and v, respectively.
(2) Matching classifier, to determine whether the two feature vectors match, i.e., present the same vulnerability information. We utilize a fully connected feed-forward module as the classifier. The cosine-similarity or the Euclidean distance is not applicable with vectors from BERT encoder [63], which lie in a high dimensional nonlinear space. The input of the feed-forward module is a joint feature vector $(u, v, |u - v|)$, providing the information of both vectors. Niels *et al.* [63] investigated the impact of different concatenation methods on the performance of Siamese architecture in classification tasks, and concluded that the element-wise difference $|u - v|$ is of vital importance.

Voting Module. The input of the voting module is the vulnerability type presented by each anchor (i.e., the associated CWE category) and the respective matching score with the target IR. The output is the classification result of whether the IR is a CIR or not. We discussed that each anchor in the external memory corresponds to a vulnerability type identified in CWE. Hence, the list of matching scores represents the probabilities of the IR describing each type of vulnerabilities. If the highest matching score is above a specified threshold, meaning a matched vulnerability type is successfully retrieved, the IR is predicted as a positive sample. Otherwise, it is predicted as a negative sample since all the candidate vulnerability types stored in the external memory fail to match.

3.2 Training Matching Module

The goal of training the matching module is to learn an encoder for extracting vulnerability-related features, jointly with a classifier for matching feature vectors. The training process takes two steps:

Further Pretraining BERT Encoder. BERT is pre-trained on BooksCorpus and English Wikipedia [40]. The pre-trained model can be leveraged to address different downstream tasks by finetuning on task-specific datasets. In our study, however, instead of directly finetuning a pre-trained BERT model, we first leverage the Masked Language Modeling (MLM), a commonly adopted pretraining task [40, 51], to further pre-train BERT using our collected IR data. The MLM task is performed in an unsupervised fashion by randomly masking some input tokens and training the model to recover the masked tokens based on the context. The motivation to further pre-train the BERT model is that, unlike the corpus used to pre-train the original BERT, IRs typically contain many technical expressions and software artifacts (e.g., code snippets). In addition, we have a large corpus available, consisting of 1,221,677 IRs, which is opportunistic to conduct a further pre-train.

Training the Siamese Architecture. We formulate the training of the Siamese architecture shown in Figure 3 as a binary classification task. The model is required to predict whether the input IR matches the paired anchor or not. We follow existing works [51, 63] to train the Siamese Network using cross-entropy as the loss function. In specific, we adjust the loss with the temperature parameter τ to help the model better benefit from hard negative samples [38]. Hard negative samples are *mismatch* pairs that easily deceive the model to make opposite predictions. Lin et al. [51] reported that these samples are crucial to both model performance and convergence speed under a similar task setting (i.e., matching issues with commits).

When building training samples for the Siamese architecture, we pair each IR (including both CIR and NCIR) in our dataset with k ($k = 16$ in our implementation) anchors. In this way, we make full use of the limited CIRs by converting n samples into $k \times n$ training sample pairs. It is essential to expose NCIRs to multiple anchors during the training since, ideally, NCIRs should not match any of the anchors during the inference. Specifically, for each NCIR, we pair it with k randomly sampled anchors from the built external memories. These training sample pairs are labeled as *mismatched*. For each CIR, we pair it with 1) the exact anchor corresponding to the type of its associated vulnerability, 2) the descriptions of CVEs belonging to the same vulnerability type. These training sample pairs are labeled as *matched*, which is meant to make the model learn which pair of descriptions relate to the same vulnerability type. Since CIR and CVE associated with the same CWE are semantically similar (i.e., describe the same vulnerability type), the incorporation of CVE descriptions increases the size of training pairs and introduces diversity, which prevents model from overfitting.

A challenge in model training is the highly imbalanced dataset with only 0.3% of the samples being CIRs. Different from the common offline strategy (e.g., SMOTE [37]), where a balanced dataset is sampled first and used during the entire training process, we adopt an online negative sampling strategy to handle the imbalanced dataset [44, 51]. A new dataset is constructed at the beginning of each training epoch with all CIRs and re-sampled NCIRs. The sample pairs used to train the Siamese architecture are also re-generated. Compared with the offline strategy, the online negative sampling strategy makes maximum use of the massive NCIR data by continually exposing the model to previously unseen negative samples. This sampling approach helps the model to avoid overfitting and benefit from hard negative samples.

3.3 Model Inference

The inference process can be generally regarded as a query, with the input IR as the key and the vulnerability types presented by the anchors as the values. We first calculate the matching scores between an IR and each anchor using the matching module. Then, we utilize the voting module to perform the retrieval. Successfully finding a matched vulnerability type (i.e., the highest matching score is above the threshold) classifies the input IR as a positive sample and vice versa. Our approach simulates the process of a software developer trying to use CWE to identify a described weakness in an IR. For efficiency, the feature vectors representing the anchors are first encoded using BERT (Figure 3) and then reused during the entire inference.

4 EXPERIMENT

In this section, we first describe our two research questions. Then, we introduce our data collection and preparation procedure. Finally, we describe our experiment setting, and present the results.

4.1 Research Questions

We answer the following research questions with our experiment:

RQ1: How effective is MemVul in identifying CIR? With this RQ, we evaluate the effectiveness of MemVul in identifying CIRs of OSS projects that are never seen during the training phase. The goal of MemVul is to help OSS users (especially software vendors who use various OSS projects in their products) sense the emerging threats and take earlier remediations, as well as helping ITS to identify dangerous IRs to better manage the vulnerability disclosure process. This requires our approach to be generalizable to IRs of various OSS projects. The most related work with ours are those on prediction of security bug reports (SBRs) [36, 42, 62, 67, 77, 78]. Therefore, we investigate the effectiveness of MemVul using several approaches from the SBR prediction as baselines.

RQ2: How effective are the key designs of our approach? In this RQ, we conduct an ablation study to verify the effectiveness of incorporating memory component and other two designs of MemVul. Therefore, we experiment with three variants (i.e., MemVul-m, MemVul-p and MemVul-o), each lacking one key design of MemVul. All other settings (both model and training) are kept exactly the same as in MemVul. The main design introduces the vulnerability knowledge from CWE using the external memory (see Section 3). We evaluate the validity of this design in tackling the challenge of learning effective vulnerability knowledge, which is caused by the differences between vulnerability types and the limited CIR data. We use the same BERT encoder (also further pretrained) that is used in the matching module (Figure 3), attached with classification layers (a fully connected feed-forward module and a softmax layer), to build a model that directly maps an input IR to its label. We refer to this variant as MemVul-m, which can be regarded as a plain MemVul without the external **m**emory. MemVul-m is trained using the online negative sampling strategy.

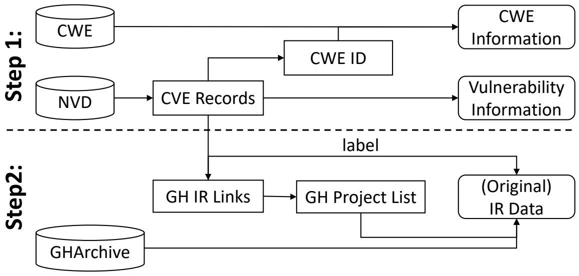

Figure 4: Overview of data collection process.

We further verify the effectiveness of the following two training strategies (Section 3.2): 1) Further pretraining of BERT encoder. We train a MemVul without the further **p**retraining, and refer to this variant as MemVul-p. 2) Online negative sampling strategy. We use a previously sampled dataset to train MemVul instead of the **o**nline strategy, and refer this variant as MemVul-o.

4.2 Data Collection

In the construction of our dataset, we utilize both GitHub IR information from a set of OSS and vulnerability information from NVD and CWE. We rely on NVD and CWE to first collect vulnerability-relevant information (e.g., GitHub IR links, vulnerability description, and CWE categories). Then, we further collect the original IR information of OSS that contain any of the vulnerability IRs we collected. The overview of our data collection process is shown in Figure 4.

Step 1: Collecting Vulnerability-Relevant Information. We first collect all CVE records from NVD (on Dec. 19, 2020) and filter out those that do not have references of GitHub IRs. We extract vulnerability information (e.g., CVE description, CWE ID and publish date) from these records. Then based on the extracted CWE IDs, we further collect the corresponding CWE information.

Step 2: Collecting Original Issue Report Information. We extract GitHub IR links from the CVE records collected in Step 1 and collect the associated IR data with the extracted links. Since our goal is to identify CIRs at the time they are created, we collect original (i.e., when first created) IR data (e.g., title, body and labels) from GHArchive [15], as such original data is not recorded by GitHub. We first get the GitHub project information of the extracted IR links. Then, we collect all IRs of these projects from GHArchive. Since GHArchive starts archiving the original information of IRs from 2015, the earliest IRs in our dataset were created on Jan. 1st, 2015. We consider the IRs that are in the extracted GitHub IR links (i.e., linked by a CVE record) as CIRs, and the remaining IRs as NCIRs.

As a result, in total, we collect 3,937 vulnerability IRs from 1,390 GitHub projects. The average #Stars and #Forks of the GitHub projects are 3,895 and 1,024 respectively, showing that the associated vulnerabilities could impact a large amount of OSS users. Together with remaining IRs (NCIRs) from these OSS projects, our dataset consists of 1,221,677 IRs in total. We extracted all available CIRs from full CVE records. Then, we collected all IRs from the relevant repositories without sampling the NCIRs. Hence, we consider our dataset retains the real-world distribution. Compared with the dataset (351 security bug reports in total, collected before 2015) used in the similar studies of triaging security bug reports [1, 59, 62, 67, 78], our collected security-related data (i.e., CIRs) are larger and more up-to-date. Moreover, these existing datasets are small and limited in project-scope, while ours contain IRs of 1,390 GitHub OSS across various development domains.

4.3 Data Preparation

We use the dataset built in Section 4.2 for model training and evaluation. Our data preparation follows two steps:

Step 1: Data Cleaning. We exclude the following IRs from the experiments: 1) CIRs whose creation time is later than the disclosure time of the corresponding CVE records. These CIRs (53 in total) do not constitute the leakage of sensitive vulnerability information, since the reported vulnerabilities have already been officially disclosed. Instead, they are used for documentation and disclosure purposes [33]. We then filter out 30 invalid projects (i.e., do not contain any CIRs) caused by the removal of these CIRs. 2) IRs with missing title and body. These samples are considered as noisy data. Finally, our dataset consists of 1,195,202 issue reports (3,884 CIRs and 1,191,318 NCIRs) from 1,360 projects.

Step 2: Data Preprocessing. We combine the original title and body (the very first comment) of the IR as the input of the model since both of them provide useful information. The title usually presents a summary of key information (e.g., the type of the vulnerability), while the details are described in the body. Besides, the IRs are usually very noisy, containing lots of special tokens (e.g., code snippets, URLs, and version numbers). To clean the IRs, we replace these tokens with specific tags (e.g., CODETAG, URLTAG, and NUMBERTAG) using regular expressions.

4.4 Experiment Setting

The experimental environment is a server with 4 NVIDIA GTX 3090 GPUs, Intel Xeon Gold 6226R CPU, running Ubuntu OS.

Testing Scenario. Our testing scenario simulates the situation where the users want to apply the trained algorithm to identify potential CIRs of new projects. This application scenario requires the learned knowledge of the model to be generalizable as the testing IRs are from projects that are previously unseen during the training. Cross-project is a commonly adopted testing scenario to evaluate the model's generalizability regarding different projects [45, 58]. We randomly sample 10% projects and use the IRs from these projects as the testing set. We again perform the same project-wise split to the remaining IRs, and use IRs from 10% projects as the validation set and remaining ones as the training set. Table 4 presents the details of the dataset used in the experiment. We retain the real-world distribution of CIRs (0.3% of all IRs) in evaluation, with highly imbalanced testing set and training set.

Implementation Details. We use the pre-trained BERT model from HuggingFace Transformer library [8]. In the further pretraining of BERT encoder (Section 3.2), we perform the MLM task on our collected IR data for 50 epochs. The outputs of the BERT encoder are pooled and further passed through a nonlinear projection header as suggested by [38]. We get two 512-dimensional feature vectors for the input IR and the anchor, respectively. The concatenation of two vectors, together with their element-wise differences are used as the input to the final classification layers. We use cross-entropy as the loss function, and set the temperature parameter to 0.1 following [38].

During the training of the matching module, we apply dropout [68] with the drop rate set to 0.1. We use AdamW [52]

Table 4: Description of dataset in cross-project scenario.

	#CIR	#NCIR	#Projects
Training Set	3,175	969,570	1,102
Validation Set	306	103,273	122
Testing Set	403	118,475	136

as the optimizer. The learning rate (lr) is set to $2e^{-5}$ for the BERT encoder as suggested for finetuning in [40], and $1e^{-4}$ for other modules. The learning rate linearly warm-ups over the first 10,000 steps (roughly three epochs) and decays in the left steps. Especially, in the online negative sampling, we sample NCIRs three times the number of CIRs. Thus, we make training pairs for each epoch in a 3:1 ratio of *mismatched* pairs to *matched* pairs, which is considered as an optimal ratio for training Siamese networks [57].

Baselines. We adopt the following baselines in our experiments:

- **Random Guess.** For each input IR, RG predicts randomly whether it is a CIR or not.
- **Simple Text Classification Approaches.** In the latest work of security bug prediction, Wu *et al.* [78] reported that simple text classification methods outperformed the specially designed approaches in previous studies [62, 67] on the clean dataset (i.e., all labels are correct). We directly use the code shared by the authors, since they perform a customized preprocessing for text tokenization and dimension reduction [78]. The five text classification approaches are Random Forest (RF), Naive Bayes (NB), Linear Regression (LR), Multilayer Perceptron (MLP), and K-Nearest Neighbor (KNN).
- **TextCNN [48].** Recently, the TextCNN has been widely applied in software engineering studies [47, 66] regarding text classification and achieved state-of-the-art results. We include TextCNN as a neural network baseline in our experiments.

Evaluation Metrics. To evaluate the performance of identifying potential CIRs, we use the metrics including Precision, Recall and F1-score. These three metrics are commonly adopted in the software engineering tasks, including security bug prediction [62, 67, 78]. Precision is the ratio of the IRs that are correctly classified as CIR to the total predictions made for CIR. Recall is the ratio of the IRs that are correctly classified as CIR to the total CIR in the ground truth. F1-score is the harmonic mean of precision and recall.

Besides, we also consider two threshold-independent metrics, i.e., AUROC (area under the Receiver Operating Characteristics curve) and AUPRC (area under the Precision-Recall curve), as the measure of the discriminatory power of the classifiers that is not affected by specific threshold values. We also adopt AUPRC, because the AUROC may not be indicative in our case where the data is extremely imbalanced, i.e. the number of negatives cases outweighs the number of positives significantly [39, 65, 69, 75].

4.5 Experiment Results

RQ1: Performance of CIR Identification. The performance comparison between our approach and the baselines for CIR identification are presented in Table 5. The best results are highlighted in **bold**. The precision of RG is extremely low, showing the challenge of this task, i.e., the scarcity of CIRs (0.3% in our dataset). The five customized simple text classification approaches behave very differently. RF, KNN, and LR achieve relatively high precision and low recall, while NB is the opposite. MLP finds a balance between precision and recall, thus achieving the best F1-score among all baselines. Similar to NB, the neural baseline, i.e., CNN, also suffers from a low precision (0.18).

Table 5: The performance comparisons between our approach and baselines for CIR identification.

Approach	Precision	Recall	F1-score	AUROC	AUPRC
RG	0.003	0.50	0.01	0.50	0.003
RF	**0.54**	0.10	0.16	0.92	0.35
NB	0.16	**0.81**	0.26	0.95	0.23
LR	0.43	0.24	0.31	0.94	0.31
MLP	0.34	0.34	0.34	0.92	0.25
KNN	0.40	0.10	0.17	0.66	0.10
CNN	0.18	0.73	0.28	0.97	0.27
MemVul	0.38	0.70	**0.49**	**0.98**	**0.39**

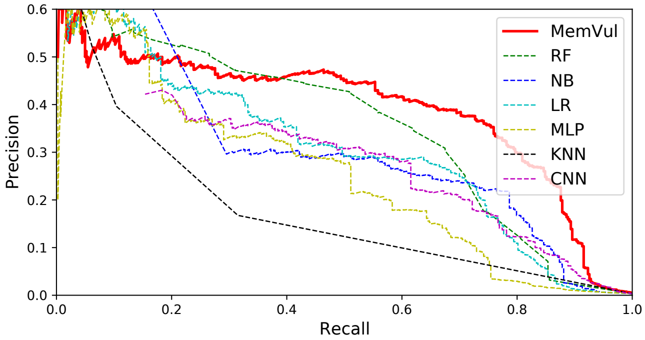

Figure 5: Precision-recall curves of MemVul and baselines.

Consider a real application scenario where a software vendor wants to monitor the emerging CIRs of OSS used by their products. Recall is important as the vendor is certainly not willing to miss threats that may cause huge losses. However, precision also needs to be considered. Since our approach monitors hundreds of OSS ITSs, detecting numerous emerging IRs, a low precision will undermine users' confidence and eventually cause the approach to be abandoned [53]. This is also the case when our approach is adopted by ITSs to monitor dangerous IRs for better management of vulnerability disclosure process. Comparing MemVul with RF, although the precision drops, recall is improved significantly from 0.10 to 0.70. For NB, although it achieves a higher recall, it sacrifices too much precision (only 0.16). When compared with MLP, which also achieves a balance between precision and recall, MemVul beats it in all metrics. Hence, we argue that our approach is better applicable for a real scenario, compared with the baselines. Since recall and precision are both important, we use F1-score as the most important evaluation metric to avoid bias, which evaluates if an increase in Precision (Recall) outweighs a reduction in Recall (Precision) [80]. Prior work [78] also stated that F1-score is the most important metric for triaging security-related IRs in practice. MemVul achieves the best performance in terms of F1-score (0.49), improving the

Table 6: The performance comparisons in ablation study.

Approach	Precision	Recall	F1-score	AUROC	AUPRC
MemVul-m	0.29	**0.85**	0.43	**0.98**	0.36
MemVul-p	0.31	0.55	0.40	**0.98**	0.31
MemVul-o	0.30	0.75	0.43	**0.98**	0.25
MemVul	**0.38**	0.70	**0.49**	**0.98**	**0.39**

performance of MLP, the one with the highest F1-score among the baselines, by 44%.

Besides, considering that precision, recall and F1-score are threshold-sensitive, we adopt AUROC and AUPRC to compare the discriminatory power (independent of specific thresholds) between our approach and baselines. MemVul achieves the best AUROC (0.98), indicating that our approach has a higher probability to rank a randomly chosen CIR higher than a randomly chosen NCIR. Different from AUROC whose baseline (i.e., random guess) is always going to be 0.5, the baseline for AUPRC is task-specific and equal to the fraction of positive cases [65]. In our task, due to the extreme data imbalance, the AUPRC of RG is much lower than the AUROC (0.003 compared with 0.5). MemVul achieves the best AUPRC of 0.39, which is satisfying given the challenge of this task. We also present the precision-recall of our approach and baselines in Figure 5. MemVul achieves the best precision at recall above 0.4. For a given recall of 0.8 (for software vendors who try to avoid missing the threats), MemVul achieves a precision of 0.30, which improves the best performing baseline (0.18) by 66.7%.

RQ-1: MemVul *achieves the best trade-off between precision and recall among all baselines. The F1-score of* MemVul *(i.e., 0.49) improves the best performing baseline by 44%.* MemVul *also achieves the best performance on threshold-independent metrics.*

RQ2: Effectiveness of Key Designs of Our Approach. In this RQ, we compare the performance of MemVul with three variants (i.e., MemVul-m, MemVul-p and MemVul-o) to verify the effectiveness of three key designs of our approach (i.e., the external memory, further pretraining of BERT and online negative sampling). Each of the three variants lacks one design compared with MemVul, while keeping all other settings exactly the same (see Section 4.1 for more details). The comparison results are presented in Table 6. The best results are highlighted in **bold**. MemVul boosts the F1-score of MemVul-m with a relative improvement of 14%. Specifically, compared with MemVul, the recall of MemVul-m is higher, but the precision drops by 24%. For threshold-independent metrics, MemVul also achieves a higher AUPRC. These results verify the effectiveness of our critical design of MemVul, i.e., incorporating the external vulnerability knowledge from CWE using a memory component. Moreover, by manually checking the testing results, we find that MemVul-m is more easily to be confounded by the *security cross words*, i.e., the same security related keywords used in both CIRs and NCIRs. Peters *et al.* [62] first introduced this term in the security bug prediction task and argued these words could mislead the model. We also point out in Section 2.3 that a large amount of NCIRs contain strong vulnerability patterns (Table 2) when applying a keyword-based approach. One possible explanation of MemVul-m being more easily deceived is that it only learns common signals (words like "memory" and "buffer") for all types of vulnerabilities. We discuss in Section 2.3 that the differences between vulnerability types (e.g., causes, behaviours, and consequences) can make it hard to effectively learn the knowledge of each type. For Memvul, however, this challenge is alleviated by introducing the external vulnerability knowledge from CWE.

For the effectiveness of the two training strategies, both of them contribute to a better performance in terms of F1-score and AUPRC. The further pretraining strategy makes the BERT encoder better adapt to the characteristics of IR data, bringing an 22.5% and 25.8% improvement in F1-score and AUPRC, respectively. The online negative sampling strategy, in another way, improves the overall performance by exposing the model to more NCIRs.

RQ-2: *The key design of incorporating the external memory improves the performance. The two training strategies (i.e., online negative sampling and further pretraining) are also effective.*

5 DISCUSSION

In this section, we discuss the results of our RQs and practical applications of our approach.

5.1 Mis-predicted CVE-Referred Issue Reports

When manually analyzing the experiment results, we observe that some false positives (i.e., NCIRs that are identified as CIRs by MemVul) actually describe security information, but the associated vulnerabilities have not been reported to the CVE. We conjecture that OSS users and maintainers can both benefit from knowing such NCIRs. Thus, we conduct a user study to investigate the usefulness of such NCIRs identified by MemVul, as well as the possible reasons why they are not referred by CVE records.

Experiment Tasks. We create tasks using the top 50 false positives (distributed among 28 OSS) with the highest probabilities of being predicted as CIRs by MemVul. For each FP, we ask four questions:

- Q1: Does this IR describe security information? (if answer is no, respondents would skip Q2-Q4)
- Q2: Which one of the following CWE category is related to the vulnerable issue reported in the IR (i.e., the CWE category can help you better understand the reported vulnerability)?
- Q3: Should the maintainer deal with this IR with higher priority (e.g., move the IR out of the public channel to minimize the impact and patch in priority)?
- Q4: Is it suggested to public disclose the vulnerable issue reported in the IR (e.g., publish a CVE record)?

For Q1, we aim to verify whether these false positives (FPs) are security-related as suggested by MemVul. For Q2, since MemVul naturally supports to track the specific matched CWE categories regarding an input IR (see Section 3.3), we aim to explore the usefulness of providing this additional information (i.e., explanation for making the prediction) in helping users better understand the vulnerability. Specifically, we provide five candidate CWE categories with the highest predicted matching scores, together with their titles, descriptions and URLs. Additionally, Q3 and Q4 aim to examine the value of these security-related but not CVE-referred IRs for OSS maintainers and users, respectively. Specially, with Q4, we also try to explore the possible reasons of not publicly disclosing these security-related IRs via CVE.

Participants. We invite five security experts from a prominent IT company with 5 to 7 years of experience in software security as our participants. Each of them is asked to finish an experiment task including 10 IRs. Our user study evaluates the usefulness of MemVul from the perspective of OSS users, i.e., helping them sense the new vulnerabilities to take early remediation.

Results. In Q1, except for 8 invalid IRs (i.e., with insufficient information or non-English words), only 1 FP is not security-related, while the remaining 41 FPs are verified by experts as security-related. In Q2, experts confirm that for these 41 FPs, they can retrieve the relevant CWE categories within the candidates we provided in most cases (29 for top 1, 32 for top 3 and 35 for top 5). The results of these two questions verify the effectiveness of MemVul for identifying IRs with security information.

In Q3, we observe that 40 security-related FPs are considered to have higher priority except for the one that was posted by the maintainer himself for disclosure purposes [6].

In Q4, we receive 6 answers as *unable to decide*. For these cases, with insufficient information provided in IRs and the limited knowledge of the projects, experts are not able to evaluate the impact of the reported IRs. In the remaining 35 FPs, 28 FPs are suggested to be publicly disclosed by experts and the other 7 FPs are not suggested to. To explore the reason why these security-related IRs are not publicly disclosed, we manually check these 35 FPs and the corresponding responds from security experts. For the 7 not-suggest-disclose FPs, we observe the following 3 possible reasons: 1) IRs were reported to wrong repositories [18, 19], 2) the reported vulnerability is not caused by the project but the operating environment [20] (e.g., compiler or dependencies), and 3) the maintainer decides that the issue is out of their scope [21] or not critical [5, 23]. For the 28 suggest-disclose FPs, we observe that 5 of them already have relevant IRs referred by CVE, hence, one potential reason is to avoid duplicated CVE reports. For example, a summarized disclosure statement [27] instead of the original issue [28] is reported and referred by the CVE-2020-15183 [54]. For the remaining 23 suggest-disclose FPs [6, 14], the reason remains unknown. One possible explanation is that these FPs may not as severe as the proper CVE-referred IRs, as the developers generally favour reports of higher-value vulnerabilities [50]. We argue that it is also beneficial to alert OSS users of this category of FPs at their preliminary stage.

We encourage future work to study the potential motivations that drive OSS maintainers to disclose certain vulnerabilities (which are believed to have a higher priority) via CVE.

5.2 Practical Applications

MemVul aims to alleviate the danger caused by the leakage of vulnerability information. We discuss the potential applications of MemVul from two aspects:

Software Vendors. MemVul promotes a better vulnerability management for software vendors. Specifically, software vendors can deploy MemVul to automatically monitor the emerging dangerous IRs of OSS projects used in their products. Beyond the current practice of solely relying on the published vulnerability information on NVD, MemVul enables vendors with the ability of early sensing new vulnerabilities at their preliminary stage. Thus, the vendors can actively perceive the risks and take timely remediations (e.g., temporary mitigation).

ITS Platforms. MemVul helps ITS to become safer by facilitating a better management of vulnerability disclosure process, i.e., helping practitioners better comply to the CVD process. ITS can integrate MemVul as a feature that would warn users before posting a potential dangerous IR and ask them to reconsider posting it, email the details directly to the maintainers instead. Moreover, users may ignore the project's security policy and insist on posting the IR publicly, e.g., irresponsibly adopt the full disclosure to get the vulnerability fixed in time, which could bother the maintainers [26]. Thus, ITS can also utilize MemVul to automatically hide the new dangerous IRs from the public and flag them for project maintainers as priorities. Reporters will be informed that these IRs have been moved to the private channel, where they can further discuss with maintainers. If maintainers take no further actions, the IR will be moved to public channel after a certain time. It is also important that ITS integrating MemVul to implement a feature through which both users and maintainers can preset a disclosure schedule (e.g., after 90 days [11, 17]), which controls the trade-off between preventing the vulnerability leakage and potential side-effects of delaying the fixes. We recommend ITS (e.g., GitHub and JIRA) that lack support of CVD process to add support for private issues and integrate MemVul to facilitate better management of dangerous IRs. ITS like Bugzilla should also consider adopting our dangerous IR detector.

5.3 Time Efficiency

Time efficiency is important when the model is deployed in production. The inference time of MemVul on the entire testset (118,878 IRs in total) using single GPU (Nvidia GTX 3090) is 322s. The average inference time per IR is less than 0.0027s.

5.4 Portability

In terms of portability, MemVul has three characteristics that are useful to practitioners:

Project Portability. 1) We leverage the vulnerability knowledge from CWE, which is high level and beyond the specific projects. Specifically, the CWE descriptions used to build the anchors is project agnostic and serve as a common language for describing weaknesses. 2) We use BERT as our encoder, which learns over any type of textual input semantically.

ITS Portability. Although MemVul is trained and evaluated on IRs from GitHub, it is ITS agnostic. MemVul is compatible with CIRs from other ITSs (e.g., Bugzilla), since it is only based on the textual descriptions of IRs but not ITS-specific features. For ITSs without a link to CVE, developers can either directly use MemVul or fine-tune MemVul using their own vulnerability database.

Vulnerability Database Portability. Other vulnerability databases, such as VulDB [29], collect extra vulnerabilities and are incorporated with SCA (Software Component Analysis). By further fine-tuning MemVul on other public or self-built vulnerability databases, MemVul can specialize for customized proxy of dangerous IRs.

6 THREATS TO VALIDITY

Internal Validity. Threats to internal validity relate to the experiment bias and errors. One threat is that the maximum sequence length input to the BERT encoder is set to 256, which may cause the loss of certain information when the IR is lengthy. This can put MemVul at a disadvantage in comparisons, since the baselines take the entire sequence as the input. Another threat is the results of Q3 and Q4 in the user study (Section 5.1) could be objective as everyone may hold different standards regarding the best practice of vulnerability disclosure. Also, the knowledge gap between security experts and the maintainer towards specific OSS projects can also affect the judgements. Moreover, there is possible bias when sampling the projects for test set under cross-project setting.

External Validity. Threats to external validity relate to the generalizability of our approach. One threat is that we use the CVE-referred GitHub IRs as the proxy of dangerous IRs. There may exist other possible proxies, for example, a different ITS (e.g., Bugzilla) and a different vulnerability database (e.g., VulDB). Future research should study different proxy of dangerous IRs to determine whether our approach is generalizable enough. Another threat relates to the choice of participants in user study. MemVul has two target customers, i.e., OSS users (especially companies that use OSSs) and maintainers (see Section 5.2). Our user study only evaluates the usefulness of MemVul from the perspective of users by inviting security experts from a prominent IT company as participants (see Section 5.1). We plan to evaluate from the perspective of maintainers in future work.

7 RELATED WORK

Prior research have investigated the importance of triaging security bug reports (SBRs) from the project ITS, so the maintainer can address them in priority. These works employ *text mining* to identify SBRs. Gegick et al. [42] utilized the TF-IDF (term-frequency inverse document frequency) weighting schema to build a statistical model for distinguishing SBRs from non-security bug reports (NSBRs). Wijayasekara et al. [76, 77] extracted syntatical information (i.e., security keywords) from the bug descriptions, and used it to generate feature vectors for Naive Bayes classifiers. Kudjo et al. [49] proposed replacing the TF-IDF schema with the TF-IGM (term-frequency inverse gravity moment) to improve the performance of text mining models. In turn, Zhou and Sharma [81] proposed the usage of a stacking approach to build an ensemble of learners to increase the model performance.

In an attempt to reduce the presence of false positives due to class unbalancing, Peters et al. [62] proposed FARSEC, a framework that removes from training data NSBRs that contain security crosswords (i.e., security related keywords that are associated with both SBRs and NSBRs). Shu et al. [67] proposed a hyperparameter optimization approach that improves the recall of FARSEC, compared to "off-the-shelf" hyperparameters. Wu et al. [78] manually corrected the labels of the datasets used by Peters et al. and Shu et al. They explored the impact of label correctness on the performance of text mining models, and found that simple text classification models yield better performances than approaches proposed in former two studies. Patrick et al. [56] pointed out the importance of augmenting standard security keywords with project-specific security vocabularies, which is proved to boost classifier performance. Particularly, they also consider the software artifacts linked by CVE entries as security-related when conducting experiments. Furthermore, Oyetoyan et al. [61] studied the problem of using cross-project data to identify security issue reports. They found that incorporating cross-project security related keywords during model training is an effective way to help on the generalization of these models, and outperforms models that are trained with project-specific keywords.

Different from prior works that use term information (e.g., TF-IDF) or manually selected security keywords as features of a machine learning model, MemVul leverages a deep learning based language model (BERT) to encode the semantic information of CIRs. Besides, prior works solely rely on the vulnerability knowledge learned by the model, while MemVul incorporates the expert-refined knowledge from CWE using an external memory component. In addition, approaches proposed in prior works are mainly limited in project-scope, which are constructed and evaluated using small and project-specific datasets. These approaches could suffer significant performance reduction when applied to unseen projects [30, 42, 61]. We build a large and real-world dataset consisting of IRs of 1,390 GitHub OSS across various development domains, and evaluate the generalizability of the proposed approach under the cross-project scenario. Moreover, compared with SBR triage, we aim to alleviate the danger caused by vulnerability information leakage. MemVul is trained and evaluated with regards to the identification of CIRs, which relate to real vulnerabilities (i.e., with CVE records) rather than security concerns or discussions in SBR.

8 CONCLUSION AND FUTURE WORK

In this work, we take the first look at the dangerous IRs and study their characteristics. With using CIRs as a proxy of dangerous IRs, we build a large-scale dataset consisting of 1,221,677 IRs from 1,390 OSS systems, with 3,937 CIRs in total. We propose a memory-augmented network, namely MemVul, introducing the well-refined vulnerability knowledge from CWE to enhance the internal knowledge learned by the model. We keep the real-world distribution of CIR (0.3% of all IRs) in evaluation and adopt a cross-project setting. Evaluation results suggest that MemVul achieves an average F1-score of 0.49, which improves the best performing baseline by 44%. For threshold-independent metrics, MemVul also achieves the highest AUROC (0.98) and AUPRC (0.39). The ablation experiments further verify the effectiveness of incorporating the external memory as the vulnerability knowledge base.

In future work, we plan to provide further analysis and more insights to the identified CIRs by inferring the specific vulnerability type each CIR presents. Actually, the current MemVul architecture naturally supports to track the matched CWE entry against the input IR (see Section 3.3). Given the vulnerability type, we can better understand the identified CIR, assess its associated risks and recommend possible mitigation.

ACKNOWLEDGMENTS

This research/project is supported by the National Science Foundation of China (No. U20A20173, No. 6190234 and No. 62141222), the Fundamental Research Funds for the Central Universities (No. 226-2022-00064), and the Key Research and Development Program of Zhejiang Province (No.2021C01105).

REFERENCES

[1] 2011. Mining Challenge of MSR 2011. http://2011.msrconf.org/msr-challenge.html.

[2] 2018. ISO/IEC 29147:2018: Security techniques - Vulnerability disclosure. https://www.iso.org/standard/72311.html.

[3] 2019. Malicious remote code execution backdoor discovered in the popular bootstrap-sass Ruby gem. https://snyk.io/blog/malicious-remote-code-execution-backdoor-discovered-in-the-popular-bootstrap-sass-ruby-gem/.

[4] 2022. About Coordinated Disclosure of Security Vulnerabilities - GitHub Docs. https://docs.github.com/en/code-security/repository-security-advisories/about-coordinated-disclosure-of-security-vulnerabilities.

[5] 2022. Anchor-cms/Issue-1328. https://github.com/anchorcms/anchor-cms/issues/1328.

[6] 2022. Anchore-engine/Issue-36. https://github.com/anchore/anchore-engine/issues/36.

[7] 2022. ASF Project Security for Committers (apache.org). https://www.apache.org/security/committers.html.

[8] 2022. bert-base-uncased · Hugging Face. https://huggingface.co/bert-base-uncased.

[9] 2022. Common Vulnerabilities and Exposure. https://www.cve.org/.

[10] 2022. Common Weakness Enumeration. https://cwe.mitre.org/index.html.

[11] 2022. Coordinated Vulnerability Disclosure for Open Source Projects. https://github.blog/2022-02-09-coordinated-vulnerability-disclosure-cvd-open-source-projects/.

[12] 2022. CWE-505. https://cwe.mitre.org/data/definitions/505.html.

[13] 2022. CWE-787. https://cwe.mitre.org/data/definitions/787.html.

[14] 2022. Dolibarr/Issue-9079. https://github.com/dolibarr/dolibarr/issues/9079.

[15] 2022. GHArchive. https://www.gharchive.org/.

[16] 2022. Graylog2-server/Issue-5906. https://github.com/Graylog2/graylog2-server/issues/5906.

[17] 2022. How Google handles security vulnerabilities. https://about.google/appsecurity/.

[18] 2022. Leanote/Issue-694. https://github.com/leanote/leanote/issues/694.

[19] 2022. Leanote/Issue-695. https://github.com/leanote/leanote/issues/695.

[20] 2022. Libraw/Issue-196. https://github.com/libraw/libraw/issues/196.

[21] 2022. Libraw/Issue-3655. https://github.com/mruby/mruby/issues/3655.

[22] 2022. Microsoft's Approach to Coordinated Vulnerability Disclosure. https://www.microsoft.com/en-us/msrc/cvd.

[23] 2022. Mruby/Issue-4928. https://github.com/mruby/mruby/issues/4928.

[24] 2022. National Vulnerability Database. https://nvd.nist.gov/.

[25] 2022. Our replication package. https://github.com/panshengyi/MemVul.

[26] 2022. Riot/Issue-10753. https://github.com/riot-os/riot/issues/10753.

[27] 2022. Soycms Security Advisory. https://github.com/inunosinsi/soycms/security/advisories/GHSA-33q6-4xmp-2f48.

[28] 2022. Soycms/Issue-8. https://github.com/inunosinsi/soycms/issues/8.

[29] 2022. Vuldb. https://vuldb.com/?doc.sources.

[30] John Anvik, Lyndon Hiew, and Gail C Murphy. 2006. Who should fix this bug?. In *Proceedings of the 28th international conference on Software engineering*. 361–370.

[31] Leyla Bilge and Tudor Dumitraş. 2012. Before we knew it: an empirical study of zero-day attacks in the real world. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 833–844.

[32] Asaf Biton. 2020. *Responsible disclosure: the impact of vulnerability disclosure on open source security.* Technical Report. SNYK.

[33] Mehran Bozorgi, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. 2010. Beyond heuristics: learning to classify vulnerabilities and predict exploits. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. 105–114.

[34] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1994. Signature verification using a" siamese" time delay neural network. *Proceedings of the 7th Advances in neural information processing systems (NIPS)* (1994), 737–737.

[35] The Black Duck by Synopsys. 2018. Open Source Security and Risk Analysis. (2018).

[36] Indu Chawla and Sandeep K Singh. 2014. Automatic bug labeling using semantic information from LSI. In *Proceedings of 7th International Conference on Contemporary Computing (IC3)*. 376–381. https://doi.org/10.1109/IC3.2014.6897203

[37] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research (JAIR)* 16 (2002), 321–357.

[38] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *Proceedings of 38th International conference on machine learning (ICML)*. PMLR, 1597–1607.

[39] Jesse Davis and Mark Goadrich. 2006. The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd international conference on Machine learning*. 233–240.

[40] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 17th Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (HLT-NAACL)*. 4171–4186.

[41] Ruian Duan, Ashish Bijlani, Meng Xu, Taesoo Kim, and Wenke Lee. 2017. Identifying open-source license violation and 1-day security risk at large scale. In *Proceedings of the 17th ACM SIGSAC Conference on computer and communications security (CCS)*. 2169–2185.

[42] Michael Gegick, Pete Rotella, and Tao Xie. 2010. Identifying security bug reports via text mining: An industrial case study. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR)*. IEEE, 11–20.

[43] Katerina Goseva-Popstojanova and Jacob Tyo. 2018. Identification of Security Related Bug Reports via Text Mining Using Supervised and Unsupervised Classification. In *Proceedings of 18th IEEE International Conference on Software Quality, Reliability and Security (QRS)*. 344–355.

[44] Jin Guo, Jinghui Cheng, and Jane Cleland-Huang. 2017. Semantically enhanced software traceability using deep learning techniques. In *Proceedings of 39th IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 3–14.

[45] Seyedrebvar Hosseini, Burak Turhan, and Dimuthu Gunarathna. 2017. A systematic literature review and meta-analysis on cross project defect prediction. *IEEE Transactions on Software Engineering* 45, 2 (2017), 111–147.

[46] Allen D Householder, Garret Wassermann, Art Manion, and Chris King. 2017. *The cert guide to coordinated vulnerability disclosure.* Technical Report. Carnegie-Mellon Univ Pittsburgh Pa Pittsburgh United States.

[47] Qiao Huang, Xin Xia, David Lo, and Gail C Murphy. 2018. Automating intention mining. *IEEE Transactions on Software Engineering (TSE)* 46, 10 (2018), 1098–1119.

[48] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of 19th Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 1746–1751. https://doi.org/10.3115/v1/D14-1181

[49] Patrick Kwaku Kudjo, Jinfu Chen, Minmin Zhou, Solomon Mensah, and Rubing Huang. 2019. Improving the Accuracy of Vulnerability Report Classification Using Term Frequency-Inverse Gravity Moment. In *Proceedings of 19th International Conference on Software Quality, Reliability and Security (QRS)*. 248–259.

[50] Frank Li and Vern Paxson. 2017. A large-scale empirical study of security patches. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2201–2215.

[51] Jinfeng Lin, Yalin Liu, Qingkai Zeng, Meng Jiang, and Jane Cleland-Huang. 2021. Traceability transformed: Generating more accurate links with pre-trained BERT models. In *Proceedings of 43rd IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 324–335.

[52] Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* (2017).

[53] André N Meyer, Thomas Fritz, Gail C Murphy, and Thomas Zimmermann. 2014. Software developers' perceptions of productivity. In *Proceedings of 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*. 19–29.

[54] MITRE. 2022. CVE-2020-15183. https://nvd.nist.gov/vuln/detail/CVE-2020-15183.

[55] MITRE. 2022. CVE-2020-15813. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-15813.

[56] Patrick Morrison, Tosin Daniel Oyetoyan, and Laurie Williams. 2018. Identifying Security Issues in Software Development: Are Keywords Enough?. In *Proceedings of the 40th International Conference on Software Engineering (ICSE)*. 426–427.

[57] Paul Neculoiu, Maarten Versteegh, and Mihai Rotaru. 2016. Learning text similarity with siamese recurrent networks. In *Proceedings of the 1st Workshop on Representation Learning for NLP*. 148–157.

[58] Chao Ni, Xin Xia, David Lo, Xiang Chen, and Qing Gu. 2020. Revisiting supervised and unsupervised methods for effort-aware cross-project defect prediction. *IEEE Transactions on Software Engineering* (2020).

[59] Masao Ohira, Yutaro Kashiwa, Yosuke Yamatani, Hayato Yoshiyuki, Yoshiya Maeda, Nachai Limsettho, Keisuke Fujino, Hideaki Hata, Akinori Ihara, and Kenichi Matsumoto. 2015. A Dataset of High Impact Bugs: Manually-Classified Issue Reports. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. 518–521. https://doi.org/10.1109/MSR.2015.78

[60] Marc Ohm, Henrik Plate, Arnold Sykosch, and Michael Meier. 2020. Backstabber's knife collection: A review of open source software supply chain attacks. In *Proceedings of the 17th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*. Springer, 23–43.

[61] Tosin Daniel Oyetoyan and Patrick Morrison. 2021. An improved text classification modelling approach to identify security messages in heterogeneous projects. *Software Quality Journal* 29, 2 (2021), 509–553.

[62] Fayola Peters, Thein Than Tun, Yijun Yu, and Bashar Nuseibeh. 2017. Text filtering and ranking for security bug report prediction. *IEEE Transactions on Software Engineering (TSE)* 45, 6 (2017), 615–631.

[63] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).

[64] C Ruffin and Christof Ebert. 2004. Using open source software in product development: A primer. *IEEE software* 21, 1 (2004), 82–86.

[65] Takaya Saito and Marc Rehmsmeier. 2015. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PloS one* 10, 3 (2015), e0118432.

[66] Lin Shi, Mingzhe Xing, Mingyang Li, Yawen Wang, Shoubin Li, and Qing Wang. 2020. Detection of hidden feature requests from massive chat messages via deep siamese network. In *Proceesings of the 42nd IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 641–653.

[67] Rui Shu, Tianpei Xia, Jianfeng Chen, Laurie Williams, and Tim Menzies. 2021. How to Better Distinguish Security Bug Reports (Using Dual Hyperparameter Optimization). *Empirical Software Engineering (EMSE)* 26, 3 (2021), 53.

[68] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* 15, 1 (2014), 1929–1958.

[69] Andrea Stocco, Michael Weiss, Marco Calzana, and Paolo Tonella. 2020. Misbehaviour prediction for autonomous driving systems. In *Proceedings of the ACM/IEEE 42nd international conference on software engineering*. 359–371.

[70] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. 2016. Matching networks for one shot learning. *Advances in neural information processing systems* 29 (2016).

[71] Duc-Ly Vu, Ivan Pashchenko, Fabio Massacci, Henrik Plate, and Antonino Sabetta. 2020. Typosquatting and combosquatting attacks on the python ecosystem. In *Proceedings of the 5th IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 509–514.

[72] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. 2020. Generalizing from a few examples: A survey on few-shot learning. *ACM Computing Surveys (CSUR)* 53, 3 (2020), 1–34.

[73] David Wehr, Halley Fede, Eleanor Pence, Bo Zhang, Guilherme Ferreira, John Walczyk, and Joseph Hughes. 2019. Learning Semantic Vector Representations of Source Code via a Siamese Neural Network. *arXiv preprint arXiv:1904.11968* (2019).

[74] Jason Weston, Sumit Chopra, and Antoine Bordes. 2015. Memory networks. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 1–1.

[75] Robert White, Jens Krinke, and Raymond Tan. 2020. Establishing multilevel test-to-code traceability links. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 861–872.

[76] Dumidu Wijayasekara, Milos Manic, and Miles McQueen. 2014. Vulnerability identification and classification via text mining bug databases. In *Proceedings of the 40th Annual Conference of the IEEE Industrial Electronics Society (IES)*. 3612–3618.

[77] Dumidu Wijayasekara, Milos Manic, Jason L Wright, and Miles McQueen. 2012. Mining bug databases for unidentified software vulnerabilities. In *Proceedings of the 5th International conference on human system interactions (HSI)*. IEEE, 89–96.

[78] Xiaoxue Wu, Wei Zheng, Xin Xia, and David Lo. 2021. Data Quality Matters: A Case Study on Data Label Correctness for Security Bug Report Prediction. *IEEE Transactions on Software Engineering (TSE)* (2021).

[79] Yueming Wu, Deqing Zou, Shihan Dou, Siru Yang, Wei Yang, Feng Cheng, Hong Liang, and Hai Jin. 2020. SCDetector: software functional clone detection based on semantic tokens analysis. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 821–833.

[80] Xin Xia, David Lo, Emad Shihab, Xinyu Wang, and Bo Zhou. 2015. Automatic, high accuracy prediction of reopened bugs. *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)* 22, 1 (2015), 75–109.

[81] Yaqin Zhou and Asankhaya Sharma. 2017. Automated identification of security issues from commit messages and bug reports. In *Proceedings of the 11th joint meeting on foundations of software engineering (FSE)*. 914–919.