HERMES: Using Commit-Issue Linking to Detect Vulnerability-Fixing Commits

Giang Nguyen-Truong
Singapore Management University
Singapore
gtnguyen@smu.edu.sg

Hong Jin Kang
Singapore Management University
Singapore
hjkang.2018@phdcs.smu.edu.sg

David Lo
Singapore Management University
Singapore
davidlo@smu.edu.sg

Abhishek Sharma
Veracode
Singapore
absharma@veracode.com

Andrew E. Santosa
Veracode
Singapore
asantosa@veracode.com

Asankhaya Sharma
Veracode
Singapore
asharma@veracode.com

Ming Yi Ang
Veracode
Singapore
mang@veracode.com

***Abstract*—Software projects today rely on many third-party libraries, and therefore, are exposed to vulnerabilities in these libraries. When a library vulnerability is fixed, users are notified and advised to upgrade to a new version of the library. However, not all vulnerabilities are publicly disclosed, and users may not be aware of vulnerabilities that may affect their applications. Due to the above challenges, there is a need for techniques which can identify and alert users to *silent* fixes in libraries; commits that fix bugs with security implications that are not officially disclosed.**

We propose a machine learning approach to automatically identify vulnerability-fixing commits. Existing techniques consider only data within a commit, such as its commit message, which does not always have sufficiently discriminative information. To address this limitation, our approach incorporates the rich source of information from issue trackers. When a commit does not link to an issue, we use a commit-issue link recovery technique to infer the potential missing link. Our experiments are promising; incorporating information from issue trackers boosts the performance of a vulnerability-fixing commit classifier, improving over the strongest baseline by 11.1% on the entire dataset, which includes commits that do not link to an issue. On a subset of the data in which all commits explicitly link to an issue, our approach improves over the baseline by 12.5%.

***Index Terms*—vulnerability curation, silent fixes, commit classification, commit-issue link recovery**

I. Introduction

Modern software projects rely on a large number of third-party libraries. As a result, there is a wider attack surface as attackers may exploit vulnerabilities in a software project's library dependencies. For example, the Log4Shell [1], [2] vulnerability impacted the Log4j logging library, one of the most popular libraries in the Java ecosystem. The impact of the vulnerability was widespread, affecting over 35,000 packages in the Java ecosystem [3].

As vulnerabilities are discovered in library dependencies, they are publicly disclosed and assigned a CVE, and recorded in the National Vulnerability Database (NVD). Users of the libraries are notified and advised to update their dependencies to the latest versions of the libraries [4], [5]. In practice, however, there may be bugs with security-implications that are fixed without public disclosure. In the literature, these vulnerability fixes are referred to as *silent fixes* [4]. Silent fixes cause users of the libraries to be unaware that their application contains a vulnerability from its use of the library. Therefore, it is valuable to users of the libraries that they are kept informed of possible vulnerabilities that are not disclosed [4].

To address the problem of silent fixes, techniques have been proposed to automatically detect vulnerability-fixing commits, but these techniques often rely only on the information within each commit. For example, some techniques use just the commit message, and others use information from the code change. However, there is often a rich context for each commit, starting from a bug report on a JIRA or GitHub issue tracker or to the ensuing discussions in the issue tracker. Developers may discuss the appropriate solution to address a bug, provide stacktraces, or describe the consequences of a bug [6]–[8].

In this paper, we suggest that using commit-issue links may help in providing the necessary information to detect vulnerability-fixing commits. Software developers may explicitly link the commit to an issue, by indicating an issue number in the commit message. Instead of using just the commit to make a judgment (as to whether it is vulnerability fixing), we incorporate information from the issues linked from each commit to enrich the information used by a classifier.

However, studies have noted that developers may not always link their commits to issues [9]–[11]. While we empirically find a significant percentage of commits with messages that directly link to an issue, we find that over 60% of commits are unlinked, motivating a need to automatically link a relevant issue to unlinked commits. Using an existing commit-issue link recovery technique [11], we infer links between each unlinked commit and an issue that best matches it.

In this paper, we propose an approach, HERMES. The key insight of our approach is that the commit has information that enables traceability, allowing us to cross the boundary[1] of a commit, linking it to issues in the issue tracker. HERMES

[1]Much like how the Greek god, Hermes, can cross the boundaries between different realms

consists of three independent classifiers, each representing a different data source. Similar to previous work, two of these classifiers identify vulnerability-fixing commits based on their commit message and by the code change. The third classifier is an issue classifier which detects vulnerability-fixing commits based on the *issues that are linked from the commit*, which was either explicitly provided by the developer or inferred through the use of a commit-issue link recovery technique. Finally, we fuse the classifications from each model to make a final classification based on an ensemble model [12].

Experimentally, we have evaluated HERMES and Sabetta and Bezzi's approach [4] on 1132 vulnerability-fixing commit from SAP manually-curated dataset [13]. These commits come from 205 distinct open-source Java projects. Using the same procedure as Ponta et al. [13], we added 5,995 commits from these projects that are not security-relevant, with a manual review aided by scripts and pattern-matching, filtering out outliers from the dataset. We find that considering only 37% of the commits that are explicitly linked to issues, allows HERMES to achieve 12.5% improvement over a baseline tool [4]. On the entire dataset, with the use of commit-issue link recovery technique, HERMES outperforms existing tools by 11.1%, demonstrating that tracing commits to their issues increases the effectiveness of vulnerability-fixing commit classification.

In this paper, we make the following contributions:

- We show that enriching the commit data using issues that are linked from the commit can boost the performance of a vulnerability-fixing commit detector
- We demonstrate that when explicit links between commits and issues are missing, the use of simple commit-issue link recovery technique to infer potential links can still result in improvement to HERMES – despite the introduction of falsely inferred links.
- We perform a comprehensive analysis of the evaluation results, providing further insights about the relationship between commit-issue link recovery and vulnerability-fixing commit detection. Our replication package is publicly available [14].

The rest of the paper is organized as follows. Section II discusses a motivating example. Section III highlights relevant background that this study builds upon. Section IV describes HERMES. Section V compares HERMES against other tools that detect vulnerability-fixing commits. Section VI provides some discussion and analysis of our work, as well as the threats to validity. Section VII discusses related studies. Finally, Section VIII presents our conclusions and future directions.

II. MOTIVATING EXAMPLE

Figure 1 shows an example of a vulnerability-fixing commit, which consists of a code change (added and deleted code) and a commit message. The purpose of the commit may not be readily apparent by analyzing the code change itself. Moreover, the commit message is uninformative, containing only two words.

```
public <T> T fromString(String content, Class<T> classOfT) {
    try (StringReader reader = new StringReader(content)) {
        JAXBContext jaxbContext = JAXBContext.newInstance(classOfT);
        Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller

        return (T) jaxbUnmarshaller.unmarshal(reader);
    } catch (JAXBException e) {
        XMLInputFactory xmlInputFactory = XMLInputFactory.newFactory()
        xmlInputFactory.setProperty(XMLInputFactory.IS_SUPPORTING_EXTE
        xmlInputFactory.setProperty(XMLInputFactory.SUPPORT_DTD, true)
        XMLStreamReader xmlStreamReader = xmlInputFactory.createXMLStr

        Unmarshaller unmarshaller = jaxbContext.createUnmarshaller();

        return (T) unmarshaller.unmarshal(xmlStreamReader);
    } catch (JAXBException | XMLStreamException e) {
```

Fig. 1: A motivating example; the commit has little discriminative information. The commit message simply contains "Fix #486", and the code change does not indicate the purpose of the change.

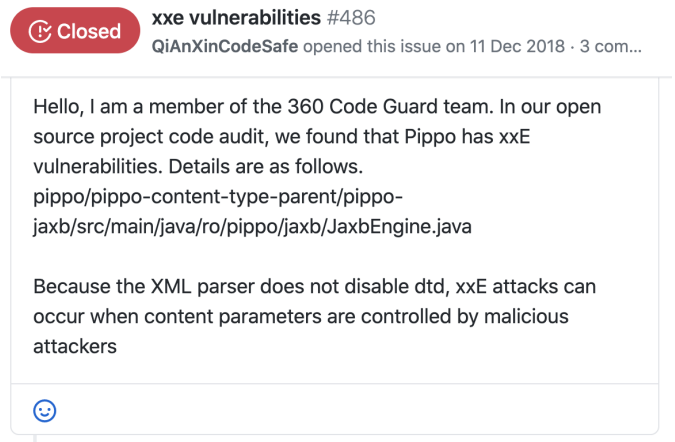

Fig. 2: The textual content of the linked issue, issue #486, of the commit from Figure 1. The issue description is highly discriminative, describing the conditions of possible attacks and their consequences.

On the other hand, by considering issue #468 linked from the commit, as shown in Figure 2, it is immediately evident that the code change is related to a vulnerability (the word "vulnerability" appears in the issue title). The issue description includes the reporter of the bug (a security team that reports vulnerabilities in various projects), the conditions under which the vulnerability can be exploited ("when content parameters are controlled by malicious attackers"), and the consequences of the vulnerability (XXE attack). Without obtaining the issue description based on the link from the commit message, a vulnerability-fixing commit detector does not consider this rich and discriminative information.

III. Background

A. Vulnerability-fixing commit classification

To detect vulnerability-fixing commits, some existing techniques take a machine learning approach for commit classification, treating the problem as a binary classification problem. Zhou and Sharma [5] convert commit messages into a vector representation (using word2vec), and use a classifier to detect vulnerability-fixing commits. Chen et al. [15] employ self-training, a semi-supervised machine learning technique, and use several features of various documents (e.g. commits) to identify vulnerability-related documents, but considers each document type *in isolation*.

Sabetta et al. [4] propose to use both the commit message and the content of the code change using a bag-of-words model, using two independent, base classifiers for the commit message and the code change. The output of the two base classifiers are combined using a simple mechanism: as long as one classifier identified the commit as security-relevant, the commit will be reported as security-relevant.

All of these existing techniques assume that the necessary information to make a correct classification is already present within the commit, in either the commit message, the code change, or both. In this study, we propose that to detect a vulnerability-fixing commit, information from other artifacts beyond a commit, such as the textual content of a corresponding issue from an issue tracker should be considered.

B. Commit-issue link recovery

While we suggest that using information from issue trackers will help, in practice, we empirically find that many commits are unlinked – this corroborates findings reported in prior studies [9], [10]. The proportion of unlinked commits have been reported to be about 35% to 40% [9], [10]. In the dataset of commits that we collected, we find that the problem is worse than what was reported in previous studies; nearly 63% of commits are unlinked. For unlinked commits, we use a technique from the literature of commit-issue link recovery to identify an issue that has the best fit with the contents of the commit, and link the issue and the commit.

In the literature of traceability research [11], [16]–[22], many techniques have been proposed to recover links between different software engineering artifacts, including between commits and issues. These techniques vary in the methods that they rely on, ranging from traditional heuristics [16]–[18], to learning-based techniques [11], [19], [20]. Recently, some sophisticated deep learning-based approaches have been proposed, including the use of transformer-based models to infer missing links [23].

In our study, we claim that the use of recovering missing commit-issue links improves our vulnerability-fixing commit classifier by considering another linked data source. To support this claim, we use a simple link recovery technique, FRLink [11], to recover missing links for the commits in the dataset. We selected FRLink for its simplicity and ease of implementation.

FRLink distinguishes between source code and non-source code changes. FRLink models the commits and issues by extracting code-related features (e.g. function names) and text features. The similarity between the commit and issue is determined by comparing the set of features associated with them. The higher the similarity, the more likely they are linked.

For a commit, code-related features are extracted from the commit message and from both source files and non-source code files in the code change. For an issue, code-related features are extracted from the title, description, and comments. First, the text is tokenized and stop words (extremely common words, e.g. ”the”, ”and”, ”is”) are removed. Code-related features are terms that appear in these texts, which match a list of regular expressions [24].

Only relevant files with a code change are used in extracting the code-related features. To determine the relevance of a source file to an issue, they first checked that the code terms in the issue appear in the file.

Next, text features are extracted. This is done through a similar process as the extraction of the code-related features, but without the use of regular expressions to match terms. For a commit, text features are only extracted from the commit message and non-source code documents.

To compute the similarity of a commit and an issue, Cosine similarity is used with the popular Term Frequency - Inverse Document Frequency (TF-IDF) weighted terms. TF-IDF is a measure of the relevance of a term to a document (e.g. an issue or commit), that considers a term to be less relevant if it is commonly used in many documents, and more relevant if it appears frequently in the given document.

Finally, FRLink [11] uses a similarity threshold, which is tuned over a ground-truth dataset. This threshold is tuned for Recall, such that FRLink infers enough correct links such that a predetermined proportion of the dataset is correctly linked.

Unlike their work, our objective is not to uncover accurate links, but to find related commits and issues in a big corpus of issues. We later investigate if there is a relationship between the accuracy of the link recovery process and the effectiveness of the vulnerability-fixing commit detector. A link could be inaccurate, but could allow HERMES to use information from an issue with sufficiently similar context (e.g. an issue that describe the same code location, or a similar type of bug) such that the additional, although not necessarily accurate, information can boost the effectiveness of the vulnerability-fixing commit detector.

Later, we perform more detailed experiments on this threshold to investigate the relationship between the accuracy of commit-issue link recovery and the performance of vulnerability-fixing commit classification.

Despite the relative simplicity of FRLink, we later show that it already substantially boosts the performance of HERMES, paving the way for future work that explores synergies between these two types of research.

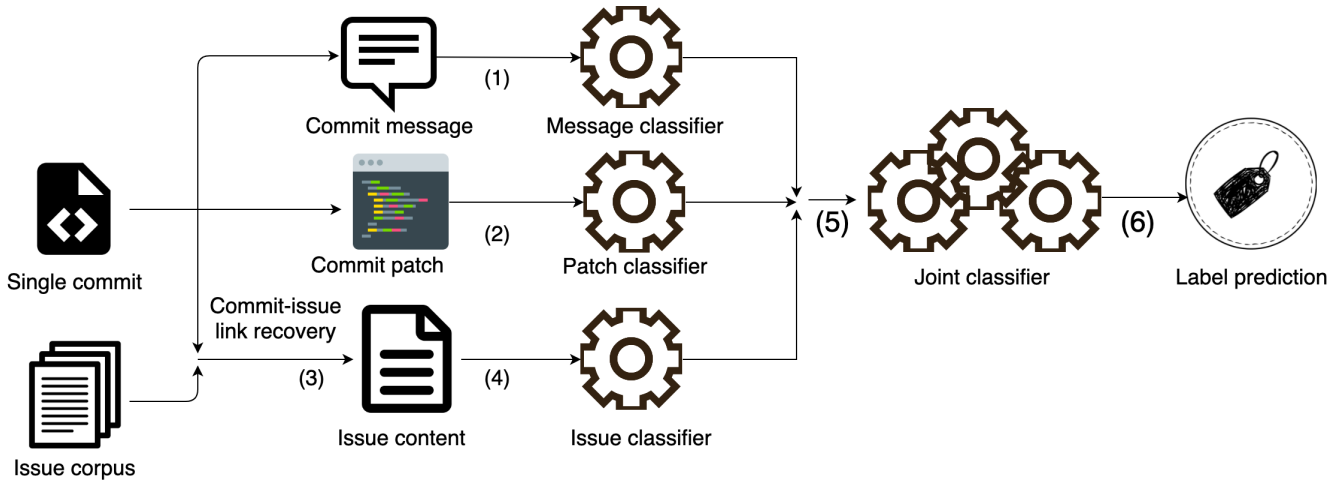

Fig. 3: Overview of HERMES

IV. APPROACH

A. Overview

Figure 3 presents a high-level design of our system. HERMES is a combination of independent modules: an issue linker, three base classifiers (commit message classifier, code change classifier, issue classifier), and a joint classifier. HERMES treats the problem as a binary classification task. It takes a commit as input, and provides a prediction of whether the commit is a vulnerability-fixing commit.

We obtain the commit message and code change from the input commit. Then, if the commit message references any issue on GitHub or a JIRA issue tracker, we directly follow the links and obtain the linked issues. For commits that are unlinked and do not have an explicit link to an issue, we infer a link to a single, most relevant issue from an issue corpus using a commit-issue link recovery technique.

The use of commit-issue link recovery enriches the dataset by providing a single issue that is relevant to the unlinked commit. When a relevant commit cannot be identified, we rely on a standard data imputation technique [25] to construct a placeholder value to input into the joint classifier. Hence, for each commit, we have three classes of information: its commit message, a code change, and the textual content of an issue related to the commit. We obtain features from each class of information, which have a different characteristic compared to the others. For each class of information, we train one classifier that captures the information content of one facet of the commit. Each classifier computes a probability that the commit is a vulnerability-fixing commit, based on the class of information considered. The outputs from the classifiers are input into a joint classifier, which performs data fusion to determine if the commit is vulnerability-fixing, considering all three classes of information.

B. Message classifier and code change classifier

Existing studies [4], [26] using machine learning on code commits consider the commit message and the code change of a commit as different classes of information about the commit. We build our work on the same view of a code comment, and in HERMES, the message classifier and code change classifier are two of the three independent classifiers.

The message classifier ((1) in Figure 3) and code change classifier (2) in Figure 3 take the commit message and code change of a commit as input, respectively. Adopting the same feature extraction method as Sabetta et al. [4], the commit message and code change in a commit are viewed as sequences of tokens. We adopt the same preprocessing steps. First, the raw documents are tokenized. Next, we normalized and stem the tokens using Porter Stemmer [27]. Non-alphanumeric characters are filtered to remove noise. We use a minimal TF-IDF threshold of 0.05 to remove infrequent tokens and tokens that contribute little information to our classifier, retaining only informative tokens. After preprocessing, the commit messages and code changes are represented as two set of features, using two different bag-of-words.

The two set of features are input to the corresponding classifiers (commit message and code change) to train two Support Vector Machine (SVM) models. As Sabetta et al. [4] used SVMs in their work, we use SVMs for a fair comparison. We use the two models to compute two probabilities that a commit is a vulnerability-fixing commit based on the commit message and the code change. These probabilities, along with the output of the issue classifier, are input to the joint classifier, which produces the output of HERMES.

C. Issue classifier

The main novelty of our approach is that we use issue information linked to the commits. First, we consider only issues explicitly linked to the commit message.

To leverage information contained in the linked issues, we have to identify the relevant issues that are indicated by a commit. To identify the right issue, we first manually identify the relevant issue tracker for all the projects in our dataset. Some of these projects use the GitHub issue tracker, while other projects use JIRA issue tracker. To identify the issue, we

look for references to issue numbers in the commit message. Table I shows the regular expressions that we use to match references to GitHub issues and Jira issue references in the commit messages. These regular expressions are based on GitHub's conventions for issue references [28], as well as a regular expression provided by the JIRA support [29]. After matching, we manually investigate the tokens that match the regular expressions, removing matches that are not references to issues, for example, "UTF-8". and checked that each match correctly corresponds to an issue on the project's issue tracker.

Next, we extract the features for representing the issue. As a commit can be linked to multiple issues, the text from every linked issue are concatenated to form a single document, from which the features for the issue classifier are extracted.

Similar to the preprocessing steps for the commit message classifier and code change classifier, the content of each issue are treated as a sequence of tokens. We performed tokenization, stop-word removal, non-alphanumeric token removal, and stemming. The issues are represented through a bag-of-words representation, which is input to a Support Vector Machine, which will produce a probability that the commit attached to the issue is a vulnerability-fixing commit ((3) in Figure 3.

D. Commit-issue link recovery

Only about a third of the commits in the dataset have an explicit link to an issue. This is a challenge of using issues to help commit classification. An approach that can work only on commits with explicit links to issues will not be generalizable – the majority of commits do not have explicit links. Therefore, while the issue classifier can be used, we also recover missing links using a commit-issue link recovery technique to provide more information ((4) in Figure 3).

To infer links for the unlinked commits, we look for tools from the literature of traceability research, which aims to construct links between different types of software artifacts including commits and issues. Among recent work, we find the work of Sun et al. [11], who propose FRLink.

Next, we construct a corpus of issues. To allows for a more realistic setting comparing commits and issues, we only obtain issues that were created in the same period as the commits in the dataset. We construct a corpus of over 290,000 issues from multiple projects with commits in the dataset.

Using FRLink, we recover commit-issue links for every unlinked commit. Each unlinked commit is linked to one issue that best matches its commit content. While commits with explicit links may reference multiple issues, we pick only one issue with the best match to link to the commit when we use FRLink. The reason for picking only one issue is that the link-recovery approach does not have a high accuracy (later discussed in detail in Section V), and we wish to avoid adding too much noise to the dataset. In practice, most commits with explicit links indicate only one corresponding issue.

We modified FRLink to allow commits to be linked to issues from a different projects. While we experimented with the default configuration of linking commits only to issues from the same repository, we found that many commits would be unlinked. About 55% of unlinked issues would remain unlinked after the link recovery process. While this would certainly lead to a drop in the accuracy of the commit-issue link recovery process, we later show (in Section V) that increased accuracy may not increase the effectiveness of HERMES. By allowing links between commits and issues of different projects, HERMES considers information from issues that match a similar context (e.g. a similar bug) as the commit.

After linking every commit to the issue with the highest similarity, we train the issue classifier using the complete set of data, including the issues with links that were recovered and the issues that were originally linked by the commit authors. It may be possible that there is some natural proportion of commits that correctly do not have any link to an issue (ad-hoc commits that contain refactoring, upgrading library versions, or preparing for new releases), therefore, FRLink can be configured to leave a portion of commit without any linked issue. This proportion is controlled by the configurable threshold in FRLink. By default, we set the threshold to 100%, which will cause all commits to be linked, if possible. We hypothesized that tuning this threshold may help in reducing noise from inaccurately recovered links. Later, in our experiments, we analyse the effect of tuning the threshold.

For a small minority of unlinked commits, there are no issues that are linked to them even after performing link recovery. This is because these commits are not similar to any issue at all. For these cases, we rely on the standard data imputation technique [25] of passing the joint classifier the mean value of the issue classifier's output. We do not run the issue classifier for these commits, and for the corresponding input to the joint model, we use the mean value output by the issue classifier for the other commits.

E. Joint classifier

By this point, we have three classifiers that each considers one class of information about the commit. Individually, each classifier may not have the necessary information to detect a vulnerability-fixing, but capture information about one facet of the commit. Similar to previous studies [4], we join the outputs of the three classifiers by using another classifier to make the final judgment for classification ((5) in Figure 3).

From the three classifiers, we obtain three probabilities that a commit is vulnerability-fixing based on the commit message, the code change, and the issue, respectively. To fuse these probabilities into the final prediction made by HERMES, we employ *stacking*, an ensemble machine learning technique that uses another classifier that learns to combine the output of the individual classifiers to produce the right prediction [12]. To do that, we combine the probabilities output by the individual classifiers with a logistic regression classifier, commonly used for stacking.

The input to the logistic regression classifier is the probabilities outputs by the three classifiers. The output of the logistic regression classifier is a prediction indicating if the commit is a vulnerability-fixing commit or not. This prediction is used as the output of HERMES ((6) in Figure 3).

TABLE I: Regular expression for matching issue's reference

Reference type	*Regular expression*	*Matching examples*
GitHub issue	`#[0-9]{1,20}`	"`#1234`", "`#552`"
JIRA issue	`((?!([A-Z0-9a-z]{1,10})-?$)[A-Z]{1}[A-Z0-9]+-\d+)`	"CAMEL-16527", "WW-4348", "STS-262"

TABLE II: Regular expressions for code terms matching

Type	*Example*	*Pattern*
C_Notation	OPT_DRIVER	`[A-Za-z]+[0-9]*_.*`
Qualified name	options.add	`[A-Za-z]+[0-9][\.].+`
CamelCase	OptionValidator	`[A-Za-z]+.*[A-Z]+.*`
Uppercase	XOR	`[A-Z0-9]+`
System variable	_cmdline	`/_+[A-Za-z0-9]+.+/`
Reference expression	std::env	`[a-zA-Z]+[:]{2,}.+`

V. Evaluation

Our experiments are driven by these research questions:

RQ1. How effective is HERMES for commits with explicit links (provided by commit authors)?

HERMES relies on the information content of issues on the issue tracker to improve the effectiveness of the commit classifier. This research question is concerned with the effectiveness of the commit classifier when using explicit commit-issue links indicated by the commit authors.

RQ2. How effective is HERMES on all commits when leveraging a commit-issue link recovery technique?

The goal of this research question is to assess the performance of the HERMES when we use issues that are from links recovered by FRLink.

RQ3. How much does noise in the inferred links between commit-issue affect HERMES?

Commit-issue link recovery techniques may recover inaccurate links. This introduces noise when classifying a commit. We investigate the effect of noise on HERMES's effectiveness.

A. Experimental Setting

We ran our experiment on SAP's manually-curated dataset [13] of fixes to vulnerabilities of open-source software. At the time of our experiment, we use 1,132 instances provided on the dataset. The dataset contains information about GitHub repositories, commit IDs of vulnerability-fixing commits, which are labeled as positive commits. To train a machine learning classifier, both examples of positives (vulnerability-fixing commits) and negative commits (non-vulnerability-fixing commits) are needed.

To obtain negative commits, we use the same augmentation process described by Sabetta et al. [4]. As vulnerability-fixing commits are rare compared to other types of commits (such as commits for developing new features or fixing other bugs), we sampled five random commits (which do not appear in the dataset) out of one positive commit, using manual analysis aided by scripts and pattern matching to filter outliers. These commits are obtained from the same repository and we treat them as negative cases. Following Sabetta et al. [4], we use an ad-hoc script to filter obvious outliers (extremely large, empty, or otherwise invalid commits). This gives us an imbalanced dataset, in which non-vulnerability-fixing commits outnumber vulnerability-fixing commits by roughly 5:1. Finally, the dataset contains 7,127 instances with 1,132 positive commits and 5,995 negative commits.

To obtain issue information, we retrieved them from two sources, which are issues trackers on GitHub and JIRA. For Github issues, we obtained their titles, bodies, and comments contents. For JIRA issues, we used their summaries, descriptions, and comments contents.

We used 10-fold cross validation in our evaluation. In 10-fold cross validation, HERMES is trained using 9 folds, or 90% of the data, and then evaluated on the remaining one fold. The evaluation metrics reported are the average of evaluation metrics computed on each fold.

In our experiments, in we set a minimum document frequency threshold of five, where terms that appear in fewer than less than five documents are removed.

Our experiments focus on evaluating the addition of issue information, therefore, as a baseline for our experiments, we compare our work against the approach by Sabetta et al. [4]. Our work builds on the work by Sabetta et al. [4], using a similar implementation of the commit message and code change classifier, with the key addition of the issue classifier.

B. Evaluation Metrics

F1 is used to evaluated HERMES, similar to numerous studies performing classification. A true positive (TP) is a vulnerability-fixing commit that is correctly detected, a false positive (FP) is a non-vulnerability-fixing commit that is incorrectly detected as vulnerability-fixing, and a false negative (FN) is a vulnerability-fixing commit that is not detected as such. Precision (P) and Recall (R) are computed as follows:

$$\mathrm{P} = \frac{\mathrm{TP}}{\mathrm{TP+FP}} \quad \mathrm{R} = \frac{\mathrm{TP}}{\mathrm{TP+FN}}$$

These two metrics capture different desirable aspects of a vulnerability-fixing commit detector. A high precision reduces the effort by security researchers in manually checking the predicted vulnerability-fixing commits. A low precision may hinder the adoption of Software Engineering tools as more human effort is required to interpret their output [30], [31]; in the context of detecting silent fixes, it is important that vulnerability-fixing commit detectors should reduce the number of commits that are manually analyzed by security researchers, who have to assess a large volume of commits from a growing number of open-source repositories. A high recall ensures that few vulnerability-fixing commits are missed. Finally, we report F1, known as the harmonic mean

TABLE III: Evaluation results on the explicitly-linked subset of data, i.e., the subset of the data with explicit links indicated by the commit author.

Model	*Precision*	*Recall*	*F1*
Sabetta et al. [4]	0.54	0.82	0.64
HERMES	0.80	0.67	**0.72**

of precision and recall to measure our classifier performance, calculated as follows:

$$F1 = \frac{2(P \times R)}{P + R}$$

where P and R indicate precision and recall respectively.

C. Experiment Results

1) Effectiveness of HERMES *on explicitly linked subset of the data:* To test the premise that including information from linked issues will help in vulnerability-fixing commit classification, we run experiments on a subset of the original dataset. In this subset, every commit is explicitly linked (by the commit authors) to at least one GitHub or JIRA issue. This subset contains 2,612 instances in which 433 instances are labeled as a vulnerability-fixing commit, and the other 2,179 instances are not.

In this initial investigation, we investigate only explicitly-linked commits in our dataset to train the issue classifier. The commit-issue link recovery technique was not used. This subset has 37% of the commits from the full dataset. The ratio of vulnerability-fixing commits to non-vulnerability-fixing commits is similar to the ratio in the original dataset.

The results of our experiments are shown in Table III. Overall, in terms of F1, HERMES improves over the approach of Sabetta et al. [4] by 12.5% ((0.72 - 0.64)/0.64). The improvements come from the higher precision of HERMES, improving over Sabetta et al. [4] from 0.54 to 0.80. Compared to Sabetta et al. [4], the use of HERMES produces fewer false alarms and leads to less human effort in inspecting commits, which is an important goal in light of the growing number of open-source libraries [32]. With explicitly-linked issues, we expect that the linked issues accurately describe the purpose of the commits. The strong experimental performance confirms our intuition that there is information in accurately linked issues that is not captured from the code change or commit message, and that this additional information improves the detection of vulnerability-fixing commits.

2) Effectiveness of commit-issue link recovery on vulnerability-fixing commit detection: Next, to assess the performance of the HERMES when we use issues from links recovered by FRLink, we automatically recovered likely links between unlinked commits to an issue from over 290,000 JIRA issues of repositories. We limit ourselves to issues that are collected from 2015. In our experiments, we first attempt to recover links for all of the unlinked commits. The dataset comprises 7,127 instances where each commit either contains at least one explicit link to an issue or a link recovered by our commit-issue link recovery technique, FRLink. Under this experimental setting, a commit would be linked to the most similar issue. As long as there is some degree of similarity to an issue, each commit would be linked to an issue. However, for 4 commits, the commit-issue link recovery technique failed to recover links as they were completely dissimilar to every issue in our issue corpus.

Table IV shows our experimental results, which is the effectiveness of HERMES on the entire dataset when we applied FRLink to connect unlinked commits with issues. Overall, HERMES has higher precision and improves over the baseline by 11.1% in terms of F1 ((0.70-0.63)/0.63).

TABLE IV: Evaluation result on all commits in the dataset

Model	*Precision*	*Recall*	*F1*
Sabetta et al. [4]	0.52	0.81	0.63
HERMES	0.74	0.66	**0.70**

This improvement was smaller than the improvement of HERMES over the baseline approach when evaluated only on explicitly-linked commits (from Section V-C1). This indicates that HERMES is sensitive to the quality of links between commits and issues. When there is a significant amount of inaccurate links between commits to issues, the performance of HERMES is negatively affected. Despite the decline in performance due to noise, HERMES still outperforms the baseline approach, which suggests that the use of commit-issue link recovery can help if we limit the recovery of links to only the commits and issues that are most similar.

Overall, our experimental results show that the use of techniques that recover commit-issue links helps detect vulnerability-fixing commits. This shows that even when developers do not explicitly link commits to issues, information from issues improves vulnerability-fixing commit detection.

3) Analysis of the effect of noise on HERMES*:* Next, we wish to investigate the relationship between the accuracy of the commit-issue link recovery and the effectiveness of HERMES.

First, we investigate the accuracy of commit-issue link recovery. To do so, we experiment on the subset of commits and issues that are explicitly linked by the commit author (e.g. the "Fix #438" commit message). Experimenting on the explicitly-linked subset of data, we run FRLink to determine if it is able to successfully recover the same link as the explicit link indicated by the commit author. FRLink could only recover 1,229 of the 2,250 explicit links, corresponding to an accuracy of 54.6%. We use the accuracy of FRLink on the explicitly-linked subset as an estimate of the accuracy of FRLink on the entire dataset.

Effect of the similarity threshold on commit-issue link recovery accuracy. When commit-issue link recovery is used, the links recovered may be inaccurate. This introduces noise into the vulnerability-fixing commit classification process. We investigate the effect of noise on the effectiveness of HERMES.

We evaluate the effect of tuning the similarity threshold in FRLink on its accuracy. Our goal is to determine if we can

TABLE V: FRLink's commit-issue link accuracy computed on the explicitly-linked subset of data, by varying its similarity threshold. We vary its similarity threshold to control for the proportion of unlinked commits that FRLink infer links for.

% unlinked commits linked by FRLink	*Accuracy*
10%	1.00
20%	0.98
30%	0.93
40%	0.87
50%	0.80
60%	0.74
70%	0.69
80%	0.64
90%	0.60
100%	0.55

control the accuracy of the commit-issue link recovery process by varying the similarity threshold.

We adjust the value of the similarity threshold such that the proportion of unlinked commits is above a given number. Note that FRLink will first infer links for the most similar commit-issue pairs, and infer links the least similar commit-issue pairs last (i.e., setting the similarity threshold for FRLink to infer links for 10% of the data indicates that 90% of the unlinked commits with the lowest similarity to any issue would remain unlinked). As earlier described, there may be a natural proportion of commits that are developed without an issue in mind (e.g. ad-hoc refactoring), and it would be inaccurate to link these commits to an issue.

We hypothesize that the more accurate the recovery of the missing links, the more effective HERMES will be. By increasing the similarity threshold, fewer inaccurate links would be introduced. This would limit the commit-issue link recovery process only to pairs of commit and issues with high similarity. We investigate the impact of modifying this threshold. Note that in Table IV, the value of this threshold was set for FRLink to infer links for 100% of unlinked data, i.e., FRLinks attempts to link all unlinked commits even if the best matching issue is a poor match for the commit.

Our experiments are summarized in Table V. As the threshold increases, the accuracy of the commit-issue link recovery process decreases. The higher the similarity threshold, the more accurate the recovered links. From our experiments, we conclude that it is unlikely that we can recover links for all commits at a high level of accuracy; however, by limiting the recovery of links using the similarity threshold, we can choose to only recover accurate links.

Effect of inaccurate links. Next, we assess the impact of inaccurate links on HERMES. On top of the explicitly-linked subset of the data, we add commits with links inferred by FRLink. We vary the similarity threshold and add the commits with inferred links to issues with similarity higher than the threshold. At low levels of the threshold, the added links are expected to be more accurate, as we have seen earlier in Table V. Our hypothesis is that HERMES is more effective when the commit-issue links are more accurate.

TABLE VI: Coverage and F1 of HERMES on the explicitly-linked data and on commits with recovered links, when varying FRLink's similarity threshold to control for the percentage of links inferred for unlinked commits

% unlinked commits linked	*Coverage (# commits)*	*F1*
0% unlinked commits linked	37% (2,612)	**0.72**
5% unlinked commits linked	40% (2,838)	0.70
10% unlinked commits linked	43% (3,064)	0.69
20% unlinked commits linked	49% (3,515)	0.68
30% unlinked commits linked	56% (3,967)	0.69
40% unlinked commits linked	62% (4,418)	0.68
50% unlinked commits linked	68% (4,870)	0.67

Table VI shows the results of the experiment. On one hand, HERMES performs best when only explicitly linked commit-issue links are used. On the other hand, this limits the data coverage of HERMES, as it cannot make accurate predictions for unlinked data, which is the majority of the dataset. Indeed, there is a tradeoff between accuracy and coverage. At its most accurate, HERMES covers only 37% of the dataset (the explicitly linked subset). To cover 68% of the dataset (by inferring links for 50% of the unlinked commits), the F1 of HERMES declines to 0.67. If we wish to use HERMES on as many commits as possible, we should allow the inference of commit-issue links for as many unlinked commits as possible.

To conclude, the performance of HERMES declines with noise introduced by commit-issue link recovery. HERMES has the best performance when using only explicitly-linked issues, but this restricts its application to only a minority of the dataset. Moreover, when increasing coverage by inferring links for more of the dataset, HERMES only faces a slight decline in performance but still has a high overall effectiveness.

VI. DISCUSSION

A. Limiting link inference to projects developed under similar development practices

While we have evaluated HERMES on a large number of different projects, we also wish to investigate the performance of HERMES if it is deployed on a set of projects that are more likely to be homogeneous in their software engineering practices. In an **additional experiment**, we investigate the performance of HERMES when both the commits and issues considered are limited to only projects that are under the Apache organization.

In this experiment, commit-issue link recovery is limited to be between commits and issues that are created under Apache projects, which use similar software development practices. This may aid the commit-issue link recovery process in creating more relevant links. We obtain 2,402 unlinked commits that were made in projects under the Apache foundation. For these commits, we inferred commit-issue links using FRLink, producing 2,383 commit-issue links. The recovery process failed to find a link to any issue for 19 commits, as they were completely dissimilar to every issue in the Apache projects. In total, this experimental setting uses 4,995 commits, of which

2,383 commits were initially unlinked and 2,616 commits had a developer-indicated, explicit link to an issue.

In this experimental setting, HERMES achieves an F1 of 0.74. This strong evaluation result is better than the results achieved on commits explicitly linked to issues (F1 of 0.72). This suggests that HERMES's performance may be related to the homogeneity of the software development practices.

B. Qualitative Analysis

In this section, we perform a qualitative analysis of the results of HERMES. Our objective is to understand the situations under which commit-issue linking aids HERMES in detecting vulnerability-fixing commits.

Figure 4 shows an example of a commit[2] which HERMES only detected as a vulnerability-fixing commit after it was linked to an issue in a different project. Figure 5 shows the inaccurately linked issue (XERCESJ-2701[3]). While some security-related tokens appeared in the commit, the code change and message classifier did not detect that the commit was vulnerability-fixing, since these terms may have appeared just as frequently in non-vulnerability-fixing commits. The commit in the VertX project matched an issue from the XercesJ project describing a similar functionality (parsing XML files) with the use of similar functions (i.e., the *newDocumentBuilder* function). The issue description described a possible consequence (an infinite loop) of the code used in the commit, which allowed the issue classifier to determine that the issue was related to a vulnerability.

Although the issue did not directly match the commit and they were inaccurately linked, they share a similar context. Moreover, the issue provided useful information about the functions used in the code, and described a possible consequence of the code that was changed by the commit. For HERMES to determine if the commit was vulnerability-related, information from the issue was essential.

Tradeoff between linking accuracy and coverage of commits. While modifying the threshold to only link a small number of commits leads to higher link accuracy, it also causes HERMES to lack additional information when making predictions for the rest of the unlinked commits. In other words, the similarity threshold used in commit-issue link recovery represents a trade-off in terms of the accuracy of HERMES on the linked commits and the proportion of total commits that HERMES can more accurately make predictions. Our experimental results indicate that there is only a minor decrease in F1 when we increase the coverage of HERMES by inferring commit-issue links.

Implications on vulnerability disclosure practices. An implication of our work is that library developers should consider the possibility of attackers using automatic traceability techniques to detect vulnerability fixes, even if developers try to limit the dissemination of the vulnerability by limiting discussion of the vulnerability and silently fixing it.

As such, our findings suggest that library developers should not hope that a silent fix goes undetected, and take steps to disclose the vulnerability to their clients to mitigate possible exploitation in client projects using the library.

Implications for future research. For detecting silent fixes, our findings indicate that information outside of a commit should be considered. While many studies use sophisticated machine learning techniques to classify commits, we find that we obtained good results by considering other practical sources of information. Our study suggests that other informative sources of data may be overlooked.

C. Threats to Validity

To mitigate threats to **internal validity**, which are concerned with mistakes in the implementation and analysis of HERMES, we have double-checked our source code and data. We selected a commit-issue link recovery technique that is simple to understand and implement, which helps us to avoid mistakes in our implementation. However, there may still be errors. To validate our approach, we have conducted experiments to understand the performance of HERMES in detail. Our source code and data are available in our replication package [14].

Another threat to validity is our reliance on the assumption that the commits are made to address issues from the issue tracker. In practice, ad-hoc commits may be made without considering any issue. This threat is mitigated as we have evaluated the effect of the accuracy of commit-issue link recovery on the effectiveness of HERMES in Section V.

To minimize threats to **construct validity**, we used the same standard evaluation metrics used in numerous studies in software engineering. We constructed the dataset in the same way as prior studies [4], and using the same process as prior work [4], [5] to clean the dataset.

To mitigate threats to **external validity**, the dataset used in our experiments contains a large number of commits from a range of projects. While we do not use the latest approaches proposed for commit-issue link recovery, we expect our findings to generalize as we use a representative tool that shares the main insight employed by most link recovery techniques: the more similar two artifacts are, the more likely that there is a link between them. In this work, we aimed for simplicity to make our approach interpretable to accurately identify the source of our improvements (the use of issues) [33]. Using the latest approaches is likely to increase the performance of HERMES, and we leave this for future work.

VII. RELATED WORK

Many studies have proposed various ways to analyze code commits. Some studies [34]–[39] analyze code changes, but do not classify commits. Yuan et al. [40] proposed to use the machine learning framework, Learning from Positive and Unlabelled examples [41], to detect bug fixing commits in the Linux kernel. In this work, we classify vulnerability-fixing commits in multiple projects.

VCCFinder [42] and DeepCVA [43] aim to identify suspicious commits. Our work is similar in that we assess commits

[2]https://github.com/vert-x3/vertx-web/commit/d814d22ade14bafec47c4447a4ba9bff090f05e8

[3]https://issues.apache.org/jira/browse/XERCESJ-1702

```
@Override
public RequestParameter isValid(String value) throws ValidationException {
  try {
-       DocumentBuilder parser = DocumentBuilderFactory.newInstance().newDocumentBuilder();
+       DocumentBuilder parser = createDocumentBuilderFactoryInstance().newDocumentBuilder();
       Document document = parser.parse(value);
       this.schemaValidator.validate(new DOMSource(document));
       return RequestParameter.create(document);
```

Fig. 4: Short snippet of the code change from a commit of VertX with the commit message "Create safe xml parsers"

Description

Hi,

When I'm trying to parse XML document using :

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

DocumentBuilder builder = factory.newDocumentBuilder();

builder.parse(File)

The parsing is stuck in infinite loop:

Fig. 5: A snippet of the issue (XERCESJ-1702) description that was most similar to the commit in Figure 4 during commit-issue link recovery. The link from the commit to XERCESJ-1702 allows HERMES to make the right classification despite the inaccuracy of the link.

to understand security risks, but while VCCFinder and DeepCVA focus on detecting commits introducing vulnerabilities, our work focuses on detecting vulnerability-fixing commits.

PatchNet [44] and DeepJIT [45] propose techniques to use deep learning on code commit classification, including detecting defect-introducing commits. CC2Vec [26] uses deep learning to learn vector representation of code changes based on their associated commit messages. Subsequent studies [46]–[48] show that simpler approaches can outperform Deep Learning in Just-in-Time defect prediction. Other researchers classify commits into maintenance and non-maintenance activities [49]. Researchers have also proposed methods of generating commit messages from code comments [35]–[39]. In these studies, the proposed techniques do not consider information beyond the commit, while we enrich the commit using issues from issue trackers.

Apart from detecting vulnerabilities from analyzing commits, methods of detecting vulnerabilities from other software development artifacts have been proposed. Chen et al. [15] and Ramsauer et al. [50] propose to identify vulnerabilities from other artifacts such as discussions on mailing lists.

Our work builds on research on software artifact traceability. This field of research is interested in studying and establishing connections between various software engineering artifacts, such as commits, documentation, and issue descriptions. Many techniques to infer links between software engineering artifacts have been proposed [11], [20], [23], [51]–[53]. Recent work focuses on using powerful machine learning models to generate links between artifacts [20], [23]. It has been noted that existing techniques are not adopted in practice due to their poor accuracy [23], a limitation that we faced in our work. As techniques recovering commit-issue links become more accurate, our proposed approach will be able to leverage these new innovations to become more effective.

VIII. Conclusion and Future Work

We suggest that using data within a commit may be insufficient to correctly classify it, and we show that commit-issue links help in vulnerability-fixing commit detection. We explore the use of a commit-issue link recovery technique to enrich a dataset previously used for assessing existing detectors of silent fixes. We find that it increases the effectiveness of the commit classifier. Our approach, HERMES, substantially improves over the state-of-the-art vulnerability-fixing commit classifier by over 15% on commits that explicitly link to issues. When the use of commit-issue link recovery is required, our technique improves over the baseline by over 7%. We analyse and understand the performance of HERMES, and highlight aspects that should be further investigated in the future.

The evaluation result we obtain from using a commit-issue link recovery technique is promising – HERMES opens up the possibility that new innovations in software traceability research could enhance vulnerability-fixing commit detection. In the future, we will evaluate our technique with the use of different commit-issue link recovery techniques. We will explore using other software engineering artifacts, such as discussions on mailing lists, to further enrich our dataset.

Acknowledgment

This project is supported by the National Research Foundation, Singapore and National University of Singapore through its National Satellite of Excellence in Trustworthy Software Systems (NSOE-TSS) office under the Trustworthy Computing for Secure Smart Nation Grant (TCSSNG) award no. NSOE-TSS2020-02. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and National University of Singapore (including its National Satellite of Excellence in Trustworthy Software Systems (NSOE-TSS) office).

REFERENCES

[1] "Log4shell: Rce 0-day exploit found in log4j 2, a popular java logging package," https://www.lunasec.io/docs/blog/log4j-zero-day.

[2] "Cve-2021-44228 – log4j 2 vulnerability analysis," https://www.randori.com/blog/cve-2021-44228.

[3] "Google security blog: Understanding the impact of apache log4j vulnerability," https://security.googleblog.com/2021/12/understanding-impact-of-apache-log4j.html.

[4] 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)*. IEEE, 2018, pp. 579–582.

[5] Y. Zhou and A. Sharma, "Automated identification of security issues from commit messages and bug reports," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (FSE)*, 2017, pp. 914–919.

[6] J. Liao, G. Yang, D. Kavaler, V. Filkov, and P. Devanbu, "Status, identity, and language: A study of issue discussions in github," *PloS one*, vol. 14, no. 6, p. e0215059, 2019.

[7] J. Tsay, L. Dabbish, and J. Herbsleb, "Let's talk about it: evaluating contributions through discussion in github," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 144–154.

[8] T. F. Bissyandé, D. Lo, L. Jiang, L. Réveillere, J. Klein, and Y. Le Traon, "Got issues? who cares about it? a large scale investigation of issue trackers from github," in *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2013, pp. 188–197.

[9] M. Rath, J. Rendall, J. L. Guo, J. Cleland-Huang, and P. Mäder, "Traceability in the wild: automatically augmenting incomplete trace links," in *Proceedings of the 40th International Conference on Software Engineering (ICSE)*, 2018, pp. 834–845.

[10] G. Schermann, M. Brandtner, S. Panichella, P. Leitner, and H. Gall, "Discovering loners and phantoms in commit and issue data," in *2015 IEEE 23rd International Conference on Program Comprehension (ICPC)*. IEEE, 2015, pp. 4–14.

[11] Y. Sun, Q. Wang, and Y. Yang, "Frlink: Improving the recovery of missing issue-commit links by revisiting file relevance," *Information and Software Technology (IST)*, vol. 84, pp. 33–47, 2017.

[12] D. H. Wolpert, "Stacked generalization," *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.

[13] S. E. Ponta, H. Plate, A. Sabetta, M. Bezzi, and C. Dangremont, "A manually-curated dataset of fixes to vulnerabilities of open-source software," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 383–387.

[14] "Our replication package," https://github.com/soarsmu/HERMES.

[15] Y. Chen, A. E. Santosa, A. M. Yi, A. Sharma, A. Sharma, and D. Lo, "A machine learning approach for vulnerability curation," in *Proceedings of the 17th International Conference on Mining Software Repositories (MSR)*, 2020, pp. 32–42.

[16] M. Fischer, M. Pinzger, and H. Gall, "Populating a release history database from version control and bug tracking systems," in *International Conference on Software Maintenance, 2003 (ICSM)*. IEEE, 2003, pp. 23–32.

[17] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1–5, 2005.

[18] A. Schröter, T. Zimmermann, and A. Zeller, "Predicting component failures at design time," in *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*, 2006, pp. 18–27.

[19] T.-D. B. Le, M. Linares-Vásquez, D. Lo, and D. Poshyvanyk, "Rclinker: Automated linking of issue reports and commits leveraging rich contextual information," in *2015 IEEE 23rd International Conference on Program Comprehension (ICPC)*. IEEE, 2015, pp. 36–47.

[20] H. Ruan, B. Chen, X. Peng, and W. Zhao, "DeepLink: Recovering issue-commit links based on deep learning," *Journal of Systems and Software (JSS)*, vol. 158, p. 110406, 2019.

[21] S. Wang and D. Lo, "Version history, similar report, and structure: Putting them together for improved bug localization," in *Proceedings of the 22nd International Conference on Program Comprehension (ICPC)*, 2014, pp. 53–63.

[22] J. Lin, Y. Liu, and J. Cleland-Huang, "Information retrieval versus deep learning approaches for generating traceability links in bilingual projects," *Empirical Software Engineering*, vol. 27, no. 1, pp. 1–33, 2022.

[23] J. Lin, Y. Liu, Q. Zeng, M. Jiang, and J. Cleland-Huang, "Traceability transformed: Generating more accurate links with pre-trained BERT models," in *International Conference on Software Engineering (ICSE)*, 2021.

[24] A. T. Nguyen, T. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "Multi-layered approach for recovering links between bug reports and fixes," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (FSE)*, 2012, pp. 1–11.

[25] R. J. Little and D. B. Rubin, *Statistical analysis with missing data*. John Wiley & Sons, 2019, vol. 793.

[26] T. Hoang, H. J. Kang, D. Lo, and J. Lawall, "CC2Vec: Distributed representations of code changes," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE)*, 2020, pp. 518–529.

[27] C. J. Van Rijsbergen, S. E. Robertson, and M. F. Porter, *New models in probabilistic information retrieval*. British Library Research and Development Department London, 1980, vol. 5587.

[28] "Github docs: Autolinked references and urls," https://docs.github.com/en/github/writing-on-github/autolinked-references-and-urls, accessed 30 April 2021.

[29] "Atlassian community: Regex pattern to match JIRA issue key," https://community.atlassian.com/t5/Bitbucket-questions/Regex-pattern-to-match-JIRA-issue-key/qaq-p/233319, accessed 30 April 2021.

[30] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 672–681.

[31] P. S. Kochhar, X. Xia, D. Lo, and S. Li, "Practitioners' expectations on automated fault localization," in *Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA)*, 2016, pp. 165–176.

[32] G. A. A. Prana, A. Sharma, L. K. Shar, D. Foo, A. E. Santosa, A. Sharma, and D. Lo, "Out of sight, out of mind? how vulnerable dependencies affect open-source projects," *Empirical Software Engineering (EMSE)*, vol. 26, no. 4, pp. 1–34, 2021.

[33] W. Fu and T. Menzies, "Easy over hard: A case study on deep learning," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (FSE)*, 2017, pp. 49–60.

[34] D. Jackson, D. A. Ladd *et al.*, "Semantic diff: A tool for summarizing the effects of modifications." in *Proceedings of the International Conference on Software Maintenance (ICSM)*, vol. 94. Citeseer, 1994, pp. 243–252.

[35] S. Jiang and C. McMillan, "Towards automatic generation of short summaries of commits," in *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. IEEE, 2017, pp. 320–323.

[36] Z. Liu, X. Xia, A. E. Hassan, D. Lo, Z. Xing, and X. Wang, "Neural-machine-translation-based commit message generation: how far are we?" in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE)*, 2018, pp. 373–384.

[37] S. Xu, Y. Yao, F. Xu, T. Gu, H. Tong, and J. Lu, "Commit message generation for source code changes," in *IJCAI*, 2019.

[38] M. Linares-Vásquez, L. F. Cortés-Coy, J. Aponte, and D. Poshyvanyk, "Changescribe: A tool for automatically generating commit messages," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE)*, vol. 2. IEEE, 2015, pp. 709–712.

[39] R. P. Buse and W. R. Weimer, "Automatically documenting program changes," in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2010, pp. 33–42.

[40] Y. Tian, J. Lawall, and D. Lo, "Identifying linux bug fixing patches," in *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 386–396.

[41] F. Denis, R. Gilleron, and F. Letouzey, "Learning from positive and unlabeled examples," *Theoretical Computer Science*, vol. 348, no. 1, pp. 70–83, 2005.

[42] H. Perl, S. Dechand, M. Smith, D. Arp, F. Yamaguchi, K. Rieck, S. Fahl, and Y. Acar, "VCCFinder: Finding potential vulnerabilities in open-source projects to assist code audits," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 426–437.

[43] T. H. Le, D. Hin, R. Croft, and M. A. Babar, "DeepCVA: Automated commit-level vulnerability assessment with deep multi-task learning," *2021 36th IEEE/ACM Automated Software Engineering Conference (ASE)*, 2021.

[44] T. Hoang, J. Lawall, R. J. Oentaryo, Y. Tian, and D. Lo, "PatchNet: a tool for deep patch classification," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2019, pp. 83–86.

[45] T. Hoang, H. K. Dam, Y. Kamei, D. Lo, and N. Ubayashi, "DeepJIT: an end-to-end deep learning framework for just-in-time defect prediction," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 34–45.

[46] C. Pornprasit and C. Tantithamthavorn, "JITLine: A simpler, better, faster, finer-grained Just-In-Time Defect Prediction," *International Conference on Mining Software Repositories (MSR)*, 2021.

[47] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," in *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, 2016, pp. 321–332.

[48] Z. Zeng, Y. Zhang, H. Zhang, and L. Zhang, "Deep just-in-time defect prediction: how far are we?" in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2021, pp. 427–438.

[49] A. Hindle, D. M. German, M. W. Godfrey, and R. C. Holt, "Automatic classication of large changes into maintenance categories," in *2009 IEEE 17th International Conference on Program Comprehension (ICPC)*. IEEE, 2009, pp. 30–39.

[50] R. Ramsauer, L. Bulwahn, D. Lohmann, and W. Mauerer, "The sound of silence: Mining security vulnerabilities from secret integration channels in open-source projects," in *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*, 2020, pp. 147–157.

[51] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "Relink: recovering links between bugs and changes," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (FSE)*, 2011, pp. 15–25.

[52] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering (TSE)*, vol. 28, no. 10, pp. 970–983, 2002.

[53] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Enhancing an artefact management system with traceability recovery features," in *20th IEEE International Conference on Software Maintenance, 2004 (ICSM)*. IEEE, 2004, pp. 306–315.