VulCurator: A Vulnerability-Fixing Commit Detector

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

Thanh Le-Cong
Singapore Management University
Singapore, Singapore
tlecong@smu.edu.sg

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

Xuan-Bach D. Le
University of Melbourne
Melbourne, Australia
bach.le@unimelb.edu.au

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

ABSTRACT

Open-source software (OSS) vulnerability management process is important nowadays, as the number of discovered OSS vulnerabilities is increasing over time. Monitoring vulnerability-fixing commits is a part of the standard process to prevent vulnerability exploitation. Manually detecting vulnerability-fixing commits is, however, time-consuming due to the possibly large number of commits to review. Recently, many techniques have been proposed to *automatically* detect vulnerability-fixing commits using machine learning. These solutions either: (1) did not use deep learning, or (2) use deep learning on only limited sources of information. This paper proposes VulCurator, a tool that leverages deep learning on richer sources of information, including commit messages, code changes and issue reports for vulnerability-fixing commit classification. Our experimental results show that VulCurator outperforms the state-of-the-art baselines up to 16.1% in terms of F1-score.

VulCurator tool is publicly available at https://github.com/ntgiang71096/VFDetector and https://zenodo.org/record/7034132#.Yw3MN-xBzDI, with a demo video at https://youtu.be/uMlFmWSJYOE.

CCS CONCEPTS

• **Computing methodologies** → **Supervised learning**; *Supervised learning by classification*; • **Security and privacy** → **Vulnerability management**.

KEYWORDS

Vulnerability-Fixing Commits, Deep Learning, BERT

ACM Reference Format:

Truong Giang Nguyen, Thanh Le-Cong, Hong Jin Kang, Xuan-Bach D. Le, and David Lo. 2022. VulCurator: A Vulnerability-Fixing Commit Detector. 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, 5 pages. https://doi.org/10.1145/3540250.3558936

ESEC/FSE '22, November 14–18, 2022, Singapore, Singapore

ACM ISBN 978-1-4503-9413-0/22/11…$15.00
https://doi.org/10.1145/3540250.3558936

1 INTRODUCTION

Open-source software (OSS) vulnerabilities can severely damage systems. An infamous example is the Equifax Data Breach[1], which led to millions of cases of identity theft. Another example is the Log4Shell[2] incident, which led to many vulnerable cloud services and applications. For vulnerability management, the information of vulnerabilities are collected in the Common Vulnerabilities and Exposures (CVE) [3] or National Vulnerability Database (NVD) [13]. OSS users can use vulnerability information such as vulnerable version(s) of a specific third-party library or how the vulnerability is fixed to make informed decisions, e.g., migrating the dependencies to invulnerable versions or patching their own client code.

Unfortunately, in practice, there is often a delay between the time a vulnerability is fixed and the time it is publicly disclosed [16], leading to a risk that OSS users are unaware of vulnerabilities in their applications. Therefore, OSS users would benefit from a tool that automatically detect security-relevant changes, i.e., vulnerability-fixing commits, that are not yet disclosed [16, 20].

Many existing techniques [2, 6, 16, 17, 19–21, 23] have recently proposed solutions for automatically identifying vulnerability-fixing commits. Several approaches [6, 17, 21, 24] use deep learning, but only consider only commit messages and code changes. Our recent work, HERMES [20], combines information from commit messages, code changes, and issue reports, however, uses Support Vector Machine (SVM). In this paper, we introduce VulCurator, a tool using a deep learning to detect vulnerability-fixing commits based on commit messages, code changes, and issue reports. Different from previous works, VulCurator leverages BERT-based models to represent both text-based and code-based information of a commit. Specifically, we use two RoBERTa [8] models for commit messages and issue reports respectively, and a CodeBERT [4] model for code changes. The output probabilities from the aforementioned classifiers are aggregated using a stacking ensemble to form the final output probability. Based on the output probability, VulCurator provides a list of commits ranked by their likelihood of being vulnerability-fixing commits.

To evaluate the performance of VulCurator, we conduct an empirical evaluation on two benchmarks, including the SAP dataset proposed by Sabetta et al. [16] and a newly collected dataset of TensorFlow vulnerabilities. While the former contains 1,132 vulnerability-fixing and 5,995 non-vulnerability-fixing commits written in Java

[1] https://nvd.nist.gov/vuln/detail/cve-2017-5638
[2] https://nvd.nist.gov/vuln/detail/CVE-2021-44228

and Python, the latter contains 290 vulnerability-fixing and 1,535 non-vulnerability-fixing commits from TensorFlow [1], a well-known deep learning framework. We compare VulCurator with two recently proposed approaches, HERMES [20], which uses Support Vector Machine classifiers using information from commit messages, code changes and issue reports, and VulFixMiner [21], a deep learning model classifying code changes from commits. Our experiments show that VulCurator outperforms HERMES by 16.1% and 8.5% on the SAP and TensorFlow dataset respectively, and VulCurator improves over VulFixMiner by 3.9% and 4.7%.

2 BACKGROUND AND RELATED WORK

Vulnerability-fixing commit classification. Vulnerability-fixing commit classification has been an active and challenging topic in software engineering research. Zhou et al. [23] use word2vec [10] to represent commit messages and forward it to a K-fold stacking model for classification. Zhou et al. [21] fine-tuned CodeBERT to transform code changes into embedding vectors and then use one-layer neural network to classify commits. Sabetta et al. [16] and Zhou et al. [24] proposed to train message classifier and code change classifier separately before combining them for commit classification. The former approach uses Support Vector Machine, while the latter uses LSTM and multi-layer CNN. Nguyen et al. recently proposed HERMES [20], which uses issue reports as a third source of information using an issue classifier and an issue linker. The issue linker maps commits without explicitly linked issues to best-matching issues.

BERT-based models. RoBERTa [8] is a multi-layer bidirectional Transformer model, which is trained on a large dataset of natural language. CodeBERT [4], a variant of RoBERTa, is trained on large-scale dataset consisting of bimodal data points which refer to natural language - programming language pair, and unimodal data points which refer to only programming language. Both RoBERTa and CodeBERT have shown to be effective in various tasks, including vulnerability-fixing classification [21, 24], type inference [5], program repair [9], program analysis [7] or defect prediction [22].

3 VULCURATOR ARCHITECTURE

Figure 1 provides an overview of VulCurator. Our tool takes as input a JSON file ① containing a list of commits with their messages, code changes and linked issues. Note that VulCurator allows commits without explicitly linked issues. In these cases, VulCurator leverages an issue linker ②, which is built based on an issue corpus ③ for mapping each commit to the most relevant issue in the corpus. Then, VulCurator feeds each type of commit information to their the corresponding classifiers, i.e. message classifier ④, patch classifier ⑤, or issue classifier ⑥. Each classifier produces a probability indicating the likelihood of a commit being a vulnerability-fixing commit. Then, the predicted probabilities from three classifiers are combined using stacking ensemble ⑦ to form the final probability.

Issue Linker. VulCurator first recovers commit-issue link for every commit without any corresponding issues as only a fraction of commits are explicitly linked to issue reports [18]. Particularly, similar from HERMES [20], VulCurator uses FRLink [18] to map each commit without any corresponding issues to its most

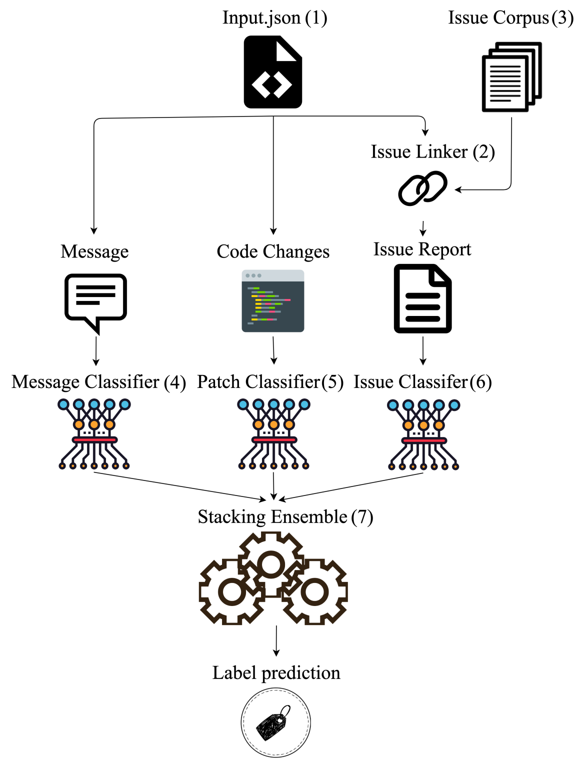


Figure 1: Overview of VulCurator

similar issue in the input data based on a pre-defined similarity function. The similarity function is calculated with respect to the Term Frequency-Inverse Document Frequency (TF-IDF) of natural language terms and code terms in commit message, code changes and issue content. The TF-IDF value of every word is calculated once using TfidfVectorizer[3] and stored locally using pickle[4] for the model inference phase. From the findings of prior work [20], the accuracy of commit-issue linking affects the classification performance. By limiting the issue linker's similarity threshold, only accurate links will be recovered.

Patch Classifier. We use the same approach as VulFixMiner [21] for the patch classifier of VulCurator. CodeBERT[5] is used as the core model. For code changes of each file, the added code and removed code version of code changes are extracted separately. The codes are tokenized using CodeBERT Tokenizer, and then formed as input for CodeBERT following the format below:

$$[CLS]\langle\text{rem-code}\rangle[SEP]\langle\text{added-code}\rangle[EOS] \qquad (1)$$

where $rem-code$ and $add-code$ are the sequence of tokens of the removed code and added code, respectively; [CLS], [SEP], [EOS] are special tokens given by CodeBERT, denoting the classification, separation and end of sequence token, respectively. The input will be forwarded to the CodeBERT to obtain an embedding vector, i.e.

[3] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
[4] https://scikit-learn.org/stable/model_persistence.html
[5] https://huggingface.co/microsoft/codebert-base

vector of numerical numbers, representing the semantic of code changes of each file. Finally, the embedding vectors are forwarded by an aggregator followed by a neural classifier to output the final probability for each commit.

Message Classifier. The message classifier leverages the multi-layer bidirectional Transformer model, RoBERTa [8]. Specifically, a commit message is tokenized into tokens using RobertaTokenizer and then forwarded into the base version of the Roberta model[6] and a softmax function to obtain the output probability.

Issue Classifier. Similar to the message classifier, the issue classifier also uses the base version of RoBERTa model. The model takes the commit issue's title and body as inputs, and outputs the predicted probability that the commit corresponding to the issue is for vulnerability-fixing.

Stacking Ensemble and Output Prediction. Given the output probabilities from the three aforementioned classifiers, VulCurator leverages a logistic regression model which acts as a stacking ensemble classifier to produce the final probability for each commit. Commits with a final probability larger than a threshold will be deemed as vulnerability-fixing commits. By default, the classification threshold is set as 0.5 but VulCurator allows users to adjust the threshold (see details in Section 4).

4 USAGE

4.1 Installation

User can either clone our GitHub [12] repository and install required dependencies or use our Docker image to run VulCurator [11]. For full customization of VulCurator, a user can follow the following steps.

4.2 Preparation

VulCurator contains a built-in Issue Linker and pre-trained Classifiers, which users can directly use. However, users can build their own Issue Linker and Classifiers following instructions below.

Issue Linker. User can customize issue corpus by providing a folder that contains files that store issue reports followed our pre-defined format (see details in our GitHub repository [12]), where each issue contains issue title, issue body, and issue comments (optional). Given the corpus, users can use their own Issue Linker by using the following commands:

```
python linker_builder.py -corpus_path <corpus_path>
```

VulCurator models. Users can also train new classifiers for VulCurator with their own dataset by using the following command:

```
python model_builder.py -data_path <path_to_data>
```

Note that the training dataset must follow a pre-defined format, which is provided on our GitHub repository [12].

4.3 Inference

VulCurator provides command line interface with two modes for end-users: *prediction* and *ranking*.

Input format. To use VulCurator, users need to prepare data following our pre-defined json format as below:

```
[
    {
        {
        "id": <commit_id>,
        "message": <commit_message>,
        "issue": {
            "title": <issue_title>,
            "body": <issue_body>,
            "comments" : [<list_of_comments]
        },
        "patch": [list_of_code_change]
    },
  ...
]
```

Prediction mode. In prediction mode, given the input of a dataset of commits, VulCurator returns a list of likely vulnerability fixing commits along with the confidence scores. Although VulCurator sets the classification threshold at 0.5 by default, VulCurator allows the threshold to be adjusted with the option `-threshold`. Users can use the following command to obtain the results:

```
python application.py  -mode       prediction
                        -input      <input_path>
                        -threshold  <threshold>
                        -output     <output_path>
```

Ranking mode. In ranking mode, users can input data following our format and VulCurator will output a list of commits sorted by the probability that the commits is vulnerability-fixing. Users can use the following commands:

```
python application.py   -mode     ranking
                        -input    <input_path>
                        -output   <output_path>
```

5 PERFORMANCE EVALUATION

In this section, we investigate the following research questions:

- *RQ1. How effective is VulCurator?*
- *RQ2. How much does each classifier contribute?*

5.1 Experimental Setting

5.1.1 Dataset. We empirically evaluate VulCurator using two datasets, the SAP dataset proposed by Sabetta et al. [16] and a newly prepared TensorFlow dataset. For each dataset, we use 80% data for training and the remaining 20% for testing.

SAP dataset: We evaluate our tool on the SAP dataset, which is widely used [16, 20]. The dataset contains vulnerability-fixing commits of widely used open-source projects manually-curated by SAP Security Research over a period of four years. Non-vulnerability-fixing commits are randomly sampled with a ratio of five non-vulnerability-fixing commits for one vulnerability-fixing commit from the same project. In total, the dataset has 1,132 vulnerability-fixing and 5,995 non-vulnerability-fixing commits, in which, 37% of the commits are explicitly linked to issues.

TensorFlow dataset: We introduce a new dataset with commits from TensorFlow, which is a well-known deep learning library. The purpose of the dataset is two-fold. First, with the increase of vulnerabilities in deep learning libraries in recent years, we would like to investigate whether VulCurator is also applicable in this

[6] https://huggingface.co/roberta-base

Table 1: F1 score of VulCurator and HERMES on SAP dataset. The number with the asterisk(*) denotes the result of VulFixMiner

Model	*Message*	*Issue*	*Patch*	*Ensemble*
HERMES	0.67	0.51	0.60	0.68
VulCurator	0.76	0.65	0.76*	**0.79**

Table 2: F1 score of VulCurator and HERMES on TensorFlow dataset. The number with the asterisk(*) denotes the result of VulFixMiner

Model	*Message*	*Issue*	*Patch*	*Ensemble*
HERMES	0.87	0.75	0.69	0.82
VulCurator	0.81	0.80	0.85*	**0.89**

domain. Second, we wish to avoid overfitting our experiments and tool design to the SAP dataset. To construct the dataset, we collect all vulnerability-fixing commits of TensorFlow, which are listed on National Vulnerability Database (NVD) [13] up until May 2022. We randomly sampled non-vulnerability-fixing commits from TensorFlow's repository using the same setting as Nguyen et al. [20] and Sabetta et al. [16]. As a result, our dataset contains 290 vulnerability-fixing and 1,535 non-vulnerability-fixing commits. In this dataset, no commit is explicitly linked to an issue.

5.1.2 Evaluation metrics. Similar to prior studies [2, 19, 20, 24], both precision and recall are important. Therefore, we use F1-score, which is the harmonic mean of precision and recall, to evaluate the effectiveness of VulCurator and HERMES.

In our task, 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. A false negative (FN) is a vulnerability-fixing commit that is not detected. Precision (P) and Recall (R) are computed as follows:

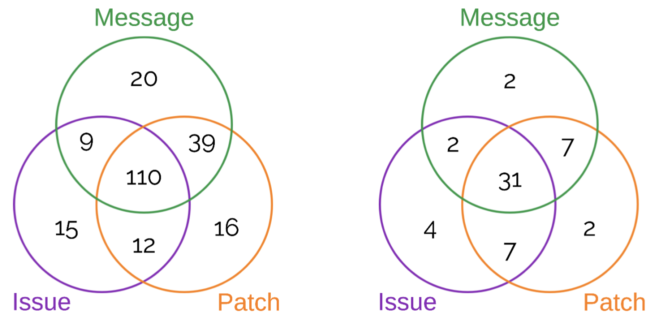
$$\mathrm{P} = \frac{\mathrm{TP}}{\mathrm{TP+FP}} \quad \mathrm{R} = \frac{\mathrm{TP}}{\mathrm{TP+FN}}$$

Then, the F1 score is calculated as follows:

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

5.2 Experimental Result

5.2.1 RQ1: Effectiveness. To answer this question, we train and test both VulCurator and HERMES on the two datasets. The experimental results are shown in Tables 1 and 2. On the SAP dataset, all VulCurator's base models and the whole model outperform HERMES's. Specifically, VulCurator's message, issue, patch classifiers and the whole model improve HERMES's counterparts by 13.4%, 27.4%, 26.7%, and 16.1% in terms of F1, respectively. On the TensorFlow dataset, while VulCurator's message classifier has a decrease of 6.9% in message classifier compared to HERMES, VulCurator issue classifier and patch classifier improves over HERMES by 6.7% and 23.2% respectively, leading to an overall 8.5% improvement over HERMES. The experiment results suggest that VulCurator benefits from the use of pre-trained deep learning models.



(a) SAP dataset   **(b) TensorFlow dataset**

Figure 2: Relationship between true positive cases predicted by three base classifiers of VulCurator

The patch classifier of VulCurator uses the same model as VulFixMiner [21]. The improvement in F1 of the ensemble model over the patch classifier alone (from 0.76 to 0.79 on SAP dataset and 0.85 to 0.89 on TensorFlow dataset) shows that combining multiple sources of information allows VulCurator to outperform VulFixMiner [21]. This result also validates the finding of Nguyen et al. [20] that using information from the issue tracker boosts classification performance.

5.2.2 RQ2: Ablation Study. We investigate if different sources of information capture different aspects of a commit. On the SAP dataset (Figure. 2a), out of 221 discovered vulnerability-fixing commits, there are 20, 15, and 16 commits that can only be exposed by message classifier, issue classifier, patch classifier, respectively. The similar finding is also found in TensorFlow (Figure. 2b). The experimental results show that each classifier helps detect unique vulnerability-fixing commits.

6 CONCLUSION AND FUTURE WORK

We present VulCurator, a tool for detecting vulnerability-fixing commits. VulCurator combines multiple sources of information such as commit messages, code changes, and issue reports in a deep learning model. In the future, to better support security researchers in monitoring commits, we plan to apply explainable AI techniques [14, 15] to provide explanations for each prediction.

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).

Xuan-Bach D. Le is supported by the Australian Government through the Australian Research Council's Discovery Early Career Researcher Award, project number DE220101057

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: a system for Large-Scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. 265–283. https://doi.org/10.5555/3026877.3026899

[2] Yang Chen, Andrew E Santosa, Ang Ming Yi, Abhishek Sharma, Asankhaya Sharma, and David Lo. 2020. A Machine Learning Approach for Vulnerability Curation. In *Proceedings of the 17th International Conference on Mining Software Repositories (MSR)*. 32–42. https://doi.org/10.1145/3379597.3387461

[3] The MITRE Corporation. 1999. Common Vulnerabilities and Exposures. https://cve.mitre.org.

[4] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*. 1536–1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139

[5] Milod Kazerounian, Jeffrey S Foster, and Bonan Min. 2021. SimTyper: sound type inference for Ruby using type equality prediction. *Proceedings of the ACM on Programming Languages* 5, OOPSLA (2021), 1–27. https://doi.org/10.1145/3485483

[6] Triet Huynh Minh Le, David Hin, Roland Croft, and M Ali Babar. 2021. Deepcva: Automated commit-level vulnerability assessment with deep multi-task learning. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 717–729. https://doi.org/10.1109/ASE51524.2021.9678622

[7] Thanh Le-Cong, Kang Hong Jin, Truong Giang Nguyen, Stefanus Agus Haryono, David Lo, Xuan Bach Le Dinh, and Thang Huynh-Quyet. 2022. AutoPruner: Tranformer-based Call Graph Pruning. In *2022 the 30th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM. https://doi.org/10.1145/3540250.3549175

[8] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *CoRR* abs/1907.11692 (2019).

[9] Ehsan Mashhadi and Hadi Hemmati. 2021. Applying codebert for automated program repair of java simple bugs. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 505–509. https://doi.org/10.1109/MSR52588.2021.00063

[10] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[11] Giang Nguyen-Truong, Thanh Le-Cong, Hong Jin Kang, Xuan-Bach Dinh Le, and David Lo. 2022. VulCurator's Docker Image. https://hub.docker.com/r/nguyentruongggiang/vfdetector.

[12] Giang Nguyen-Truong, Thanh Le-Cong, Hong Jin Kang, Xuan-Bach Dinh Le, and David Lo. 2022. VulCurator's Repository. https://github.com/ntgiang71096/VFDetector.

[13] U.S. National Institute of Standards and Technology. 1999. National Vulnerability Database. https://nvd.nist.gov.

[14] Chanathip Pornprasit, Chakkrit Tantithamthavorn, Jirayus Jiarpakdee, Michael Fu, and Patanamon Thongtanunam. 2021. PyExplainer: Explaining the Predictions of Just-In-Time Defect Models. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 407–418. https://doi.org/abs/10.1109/ASE51524.2021.9678763

[15] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. " Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1135–1144. https://doi.org/10.1145/2939672.2939778

[16] Antonino Sabetta and Michele Bezzi. 2018. A practical approach to the automatic classification of security-relevant commits. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 579–582. https://doi.org/10.1109/ICSME.2018.00058

[17] Arthur D Sawadogo, Tegawendé F Bissyandé, Naouel Moha, Kevin Allix, Jacques Klein, Li Li, and Yves Le Traon. 2020. Learning to catch security patches. *arXiv preprint arXiv:2001.09148* (2020).

[18] Yan Sun, Qing Wang, and Ye Yang. 2017. Frlink: Improving the recovery of missing issue-commit links by revisiting file relevance. *Information and Software Technology* 84 (2017), 33–47. https://doi.org/10.1016/j.infsof.2016.11.010

[19] Yuan Tian, Julia Lawall, and David Lo. 2012. Identifying linux bug fixing patches. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 386–396. https://doi.org/10.1109/ICSE.2012.6227176

[20] Nguyen Truong-Giang, Kang Hong Jin, David Lo, Abhishek Sharma, Andrew Santosa, Asankhaya Sharma, and Ming Yi Ang. 2022. HERMES: Using Commit-Issue Linking to Detect Vulnerability-Fixing Commits. In *The 2022 29th IEEE International Conference on Software Analysis, Evolution and Reengineering*. IEEE. https://doi.org/10.1109/SANER53432.2022.00018

[21] Jiayuan Zhou, Michael Pacheco, Zhiyuan Wan, Xin Xia, David Lo, Yuan Wang, and Ahmed E Hassan. 2021. Finding A Needle in a Haystack: Automated Mining of Silent Vulnerability Fixes. In *2021 36th IEEE/ACM Automated Software Engineering Conference (ASE)*. IEEE. https://doi.org/10.1109/ASE51524.2021.9678720

[22] Xin Zhou, DongGyun Han, and David Lo. 2021. Assessing generalizability of CodeBERT. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 425–436. https://doi.org/10.1109/ICSME52107.2021.00044

[23] Yaqin Zhou and Asankhaya Sharma. 2017. 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)*. 914–919. https://doi.org/10.1145/3106237.3117771

[24] Yaqin Zhou, Jing Kai Siow, Chenyu Wang, ShangQing Liu, and Yang Liu. 2021. SPI: Automated Identification of Security Patches via Commits. *ACM Transactions on Software Engineering and Methodology (TOSEM)* (2021). https://doi.org/10.1145/3468854