International Journal of Information Security
https://doi.org/10.1007/s10207-023-00795-8

REGULAR CONTRIBUTION

Vulnerability discovery based on source code patch commit mining: a systematic literature review

Fei Zuo[1] · **Junghwan Rhee**[1]

Abstract
In recent years, there has been a remarkable surge in the adoption of open-source software (OSS). However, with the growing usage of OSS components in both free and proprietary software, vulnerabilities that are present within them can be spread to a vast array of underlying applications. Even worse, a myriad of vulnerabilities are fixed secretly via patch commits, which causes other software re-using the vulnerable code snippets to be left in the dark. Thus, source code patch commit mining toward vulnerability discovery is receiving immense attention, and a variety of approaches are proposed. Despite that, there is no comprehensive survey summarizing and discussing the current progress within this field. To fill this gap, we survey, evaluate, and systematize a list of literature and provide the community with our insights on both successes and remaining issues in this space. Special attention is paid on the work toward vulnerability discovery. In this paper, we also provide an introductory panorama with our replicable hands-on experience, which can help readers quickly understand and step into the pertinent field. Our empirical study reveals noteworthy challenges which need to be highlighted and addressed in this field. We also discuss potential directions for the future work. To the best of knowledge, we provide the first literature review to study source code patch commit mining in the vulnerability discovery context. The systematic framework, hands-on practices, and list of potential challenges provide new knowledge for mining source code patch commit toward a more robust software eco-system. The research gaps found in this literature review show the need for future research, such as the concern on data quality, high false alarms, and the significance of textual information.

Keywords Source code patch commit · Vulnerability discovery · Security patch identification · Semantic learning · Open source software

1 Introduction

Source code commits are the core building block units of a version control system in software development, which can be considered as "*a set of individual snapshots of project content*" [1]. When developers are working on a project, they continuously make changes to the codebase, such as adding new features. Once these changes are ready to be saved, a commit that records the changes made since the last commit will be created.

The patch commit (i.e., code changes + description of changes) [2], or patch for short, is a general concept involving modifications that are specifically focused on code updates such as introducing new features. By contrast, if the patch commits are used to fix known security issues, we call them security patches [3]. Usually, they require more attention because these updates can prevent attackers from exploiting the vulnerabilities to compromise the system.

In the last decade, the number of critical vulnerabilities found in open-source software (OSS) has risen sharply. One public security advisory, the common vulnerabilities and exposures (CVE) system monitors publicly disclosed software vulnerabilities. According to [4], the number of CVE records disclosed in 2022 was 25,059, which is more than 4.7 times greater than the 5288 reported in 2012. When new vulnerabilities are found, it is essential for project maintainers or developers to act timely and correct the problem through submitting a security patch commit to the code repository.

✉ Fei Zuo
fzuo@uco.edu

Junghwan Rhee
jrhee2@uco.edu

[1] Department of Computer Science, University of Central Oklahoma, Edmond, OK 73034, USA

Published online: 06 January 2024

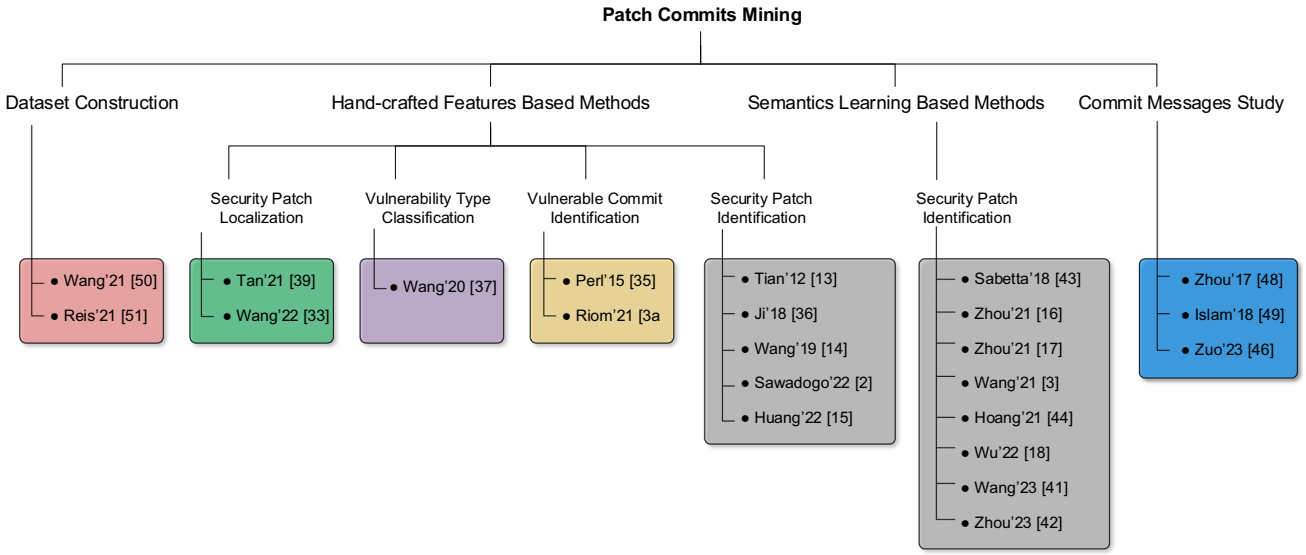

Fig. 1 An overview and taxonomy of the papers studied in this review

A primary motivation of patch commits mining is to analyze vulnerabilities that were fixed without being publicly disclosed.

Previous researches, such as [5, 6], have shown that software providers may opt to silently fix vulnerabilities in new versions of their software without appropriately reporting them to the CVE system. Even worse, they may not explicitly describe the vulnerabilities in their change logs. This kind of decision may be influenced by concerns about reputation and the ease of software development management. Additionally, the widespread practice of sharing code in software development can greatly increase the propagation of vulnerabilities. Although project maintainers promptly fix these bugs, there is still a significant risk because other software using vulnerable code fragments are not aware of the updates. Therefore, identifying these silently fixed vulnerabilities is crucial in modern software ecosystems. Unfortunately, this problem has yet to be fully resolved, despite its importance. While manual verification is a potential solution, it is impractical due to the massive amount of time and labor required. Given the gap between an increasing number of vulnerabilities and limited manpower, automated patch commit mining is a more feasible option.

Conventional software analysis techniques, for example fuzzing [7] and symbolic execution [8, 9], have been widely adopted in the vulnerabilities discovery. Nevertheless, these techniques usually cannot be applied to incomplete code fragment such as the code changes in a commit. Therefore, they are not suitable for patch commits mining. Not only that, these code-oriented methods also cannot support the analysis of natural language such as commit messages.

Inspired by the success of applied machine learning and deep learning in diverse areas [10–12], many data-driven approaches have been proposed for the automated patch commit analysis. For example, previous works [2, 13–15] have used hand-crafted features such as the number of modified lines and characters to describe code changes, but this approach heavily relies on heuristics or domain knowledge. On the other hand, with deep learning's impressive achievements in representation learning, researchers are now turning their focus toward this new arsenal. As a result, many works [3, 16–18] have adopted deep learning techniques to extract semantics from patch commits.

While there exist a surge of works on patch commits mining, many research challenges still remain open. In particular, we currently lack a comprehensive survey summarizing and discussing recent advances as well as unsolved questions within this field. Therefore, this paper aims to fill in the knowledge gap by surveying the literature on the patch commit analysis targeting vulnerability discovery. There are three key contributions presented by this survey. First, we provide an introductory panorama with our replicable hands-on experience, which will help readers step into the field and quickly understand it. Second, we investigate, classify and summarize recent advances in patch commits mining. Our valuable insights can help the community build a deeper understanding about the related topics. Third, we provide a systematic description of the challenges and propose possible directions for future works.

Figure 1 shows an overview and taxonomy of the major papers surveyed in this review. It should be noted that the software patches studied in this paper are in the form of patch commits. The security patches for binaries [19] and the binary analysis-based vulnerability discovery [20] are out of our scope. Furthermore, we especially concentrate on source code patch commit mining targeting vulnerability discover-

ies in this paper. Therefore, the security patch management topic that has been surveyed by the previous work [21] is also beyond the scope of consideration.

The remaining sections of this paper are organized as follows. In Sect. 2, we will introduce necessary information about the background of this field. Next, the main body of the literature review is presented in Sect. 3. Sections 4 and 5, respectively, describe two related research areas in brief, i.e., the empirical analysis on security patches and bug report study. In Sect. 6, we discuss some challenges that need to be addressed and outline the research directions for the future works. Finally, Sect. 7 draws the conclusion.

2 Preliminaries

In this section, we briefly introduce the necessary background of patch commits, bug reports, and representation learning that are involved in this paper.

2.1 Commit

Nowadays, source code patch commits are playing a more and more important role throughout the various phases of the software development life cycle. In general, a commit usually consists of a commit message and a source code difference (also known as "diff"), which records code changes between different software versions. Especially in the software maintenance, an update often includes numerous patch commits. Among them, some patches are applied for non-security purposes, such as improving the performance and introducing new features. In particular, the `git` commit is a representative commit managed by a specific code repository tool, that is `git`. Unless otherwise specified, the commit mentioned throughout this survey is, namely the `git` commit because all the investigated papers in this field only consider this type of commits.

Figure 2 showcases a non-security patch commit in radare2, which is a reverse engineering framework. Specifically, this patch is used to introduce several new macro definitions (Line 18–21). By contrast, other patch commits are particularly used for fixing vulnerabilities, and are thus called security patches. For instance, Fig. 3 is a concrete commit for the security patch to fix vulnerability CVE-2022-0699. This patch is used to get rid of a double free vulnerability by removing Line 19. We present this sample commit to showcase the components of a commit in greater detail. The commit message and the patch diff is separated by a line with a three-dash line. In the diff, we can see the source file "*contrib/shpsort.c*" is modified. Moreover, the line beginning with "`@@`" in a chunk is in red font, which is to display specific information, such as the line number at which the changes begin and the name of the function that

```
1  From 7e56e7165e4e76201678765d228c59efddbca4c6
2  From: pancake <pancake@nowsecure.com>
3  Date: Tue, 21 Feb 2023 17:47:33 +0100
4  Subject: [PATCH] Introduce the new R_CONST macros
   ##api
5
6  ---
7   libr/include/r_types_base.h | 5 +++++
8   1 file changed, 5 insertions(+)
9
10 diff --git a/libr/include/r_types_base.h
   b/libr/include/r_types_base.h
11 index 04aaad0becb0..47519804e36e 100644
12 --- a/libr/include/r_types_base.h
13 +++ b/libr/include/r_types_base.h
14 @@ -262,6 +262,11 @@ typedef struct _utX {
15  #define R_IS_DIRTY(x) (x)->is_dirty
16  #define R_DIRTY_VAR bool is_dirty
17
18 +#define R_CONST_MAYBE
19 +#define R_CONST_TAG(x) ((x)|1)
20 +#define R_CONST_UNTAG(x) (((x)>>1)<<1)
21 +#define R_CONST_FREE(x) do { if (!(x&1)
   { R_FREE(x); }} while(0)
22 +
23  #ifdef __cplusplus
24  }
25  #endif
```

Fig. 2 An example of non-security patch commit

```
1  From c75b9281a5b9452d92e1682bdfe6019a13ed819f
2  From: Albin Eldstal-Ahrens <laeder.keps@gmail.com>
3  Date: Mon, 3 Jan 2022 12:34:41 +0100
4  Subject: [PATCH] Remove double free() in
   contrib/shpsrt, issue #39
5
6  This fixes issue #39
7  ---
8   contrib/shpsort.c | 1 -
9   1 file changed, 1 deletion(-)
10
11 diff --git a/contrib/shpsort.c b/contrib/shpsort.c
12 index e21e9e0..920cd8c 100644
13 --- a/contrib/shpsort.c
14 +++ b/contrib/shpsort.c
15 @@ -113,7 +113,6 @@ static char ** split(const
   char *arg, const char *delim) {
16          free(result[--i]);
17        }
18        free(result);
19 -      free(copy);
20        return NULL;
21      }
22      result = tmp;
```

Fig. 3 A security patch commit for CVE-2022-0699

the changes are made in. The following lines of the diff are the lines of code that have been modified from the original, where the line marked with "−" means such a line is removed from the previous version, while the line marked with "+" means that is a new line added in the current version.

A commit message is comprised of a subject, body, and footer, with both the body and footer being optional. For example, in Figs. 2 and 3, the commit messages are in purple color. In particular, the subject is a single line that best

Table 1 A bug report from the Bugzilla bug tracking system

Bug ID	1567441
Summary	Divide-by-zero in [@mozilla::FramesToTimeUnit] through ADTS
Product	Core
Component	Audio/Video: Playback
Platform	x86_64 Linux
Type	Defect
Priority	P2
Severity	Normal
Description	The attached testcase crashes on mozilla-central revision 29e9dde37bd2 (build with –enable-tests –enable-address-sanitizer –disable-jemalloc –enable-optimize=-O2 –enable-fuzzing –disable-debug) For detailed crash information, see attachment. To reproduce the issue 1. Build or download an ASan –enable-fuzzing build including gtests 2. Run FUZZER=MediaADTS LIBFUZZER=1 MOZ_RUN_GTEST=1 objdir/dist/bin/firefox test.bin

summarizes the changes made in the commit. By default, the subject of a single patch starts with "`Subject: [PATCH]`." But if there exist multiple patches, the subject prefix will instead be "`Subject: [PATCH n/m]`." Sometimes when the change is so simple that no further explanation is necessary, a single subject line is fine. Thus, not every commit includes a body. However, if a commit merits a more thorough description, the body text can provide more details regarding the changes made in the commit. The body is separated by the subject using a blank line. As the last component of a commit message, the footer text is found immediately below the body and is a preferable place to reference issues related to the commit changes. Between the footer and the body, there also exists a blank line as the separator.

2.2 Bug report

A bug report is a document or a report that describes an error or an issue in a software application. It is typically created by a user, a tester, or a quality assurance engineer who has identified a problem with the software. A bug report usually includes information about the symptoms of the bug, the steps taken to reproduce it, and any relevant information about the user's system or environment. The purpose of a bug report is to provide developers with enough information to understand the problem and to be able to fix it. Clear and detailed bug reports can help developers to fix the issue more quickly and efficiently. Therefore, we see many prior works provide insightful recommendations for better bug reports, such as [22, 23].

Furthermore, numerous bug tracking systems are publicly available, which can collect, track and manage software bugs or issues. For example, Bugzilla[1] is a representative bug tracking system that has been widely used by the open-source community. It was originally created and used by the Mozilla Foundation for tracking bugs in the Firefox web browser and related projects, but has since been adopted by many other software development organizations. Table 1 showcases a bug report sample from Bugzilla.

2.3 Deep representation learning

The recent decade has witnessed tremendous advances of neural networks, especially in the area of natural language processing (NLP) and applied graph theory. What's more, an attractive feature of neural networks is they can discover directly from the original representation with minimal feature engineering needed. In other words, deep neural network models can be trained to yield numerical representations of data so that the resulting representations are suitable for the future analysis tasks. The process as such is called deep representation learning.

In particular, graph is a very active research field in deep representation learning. The neural network that is designed to operate on graph-structured data is called a graph neu-

[1] https://www.bugzilla.org/.

ral network (GNN). It can leverage the structure of graphs to learn numeric representations of nodes and edges, which capture both their local and global characteristic in the graph. Some popular architectures of GNNs include graph convolutional networks [24], graph attention networks [25], and GraphSAGE [26].

Meanwhile, we also observe the significant advances in NLP led by deep representation learning. For example, unsupervised learning methods like Word2Vec [27], Sent2Vec [28] and Doc2Vec [29], can, respectively, generate numeric representations for words, sentences, and paragraphs, with preserving the semantic and syntactic information in the resulting embeddings. Furthermore, Recurrent Neural Networks (RNNs), along with its variations Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), are capable of processing sequential data, because they can capture long-term dependencies between words and encode context information. They are also widely used in many supervised NLP tasks. Last but not least, state-of-the-art performance has been achieved by models based on Transformer, such as BERT [30] and GPT [31, 32], in a wide range of applications. They are based on a self-attention mechanism, which enables them to not only capture long-range dependencies between tokens but also be well-suited for parallel acceleration.

3 Comparative evaluation

We split our analysis of the existing works on patch commits mining into four parts. First, we present an extensive survey of the papers involving feature engineering, and systematize them according to their research focus. Second, we review the advances in the field of patch semantics learning, and highlight their characteristics and innovations. Third, we discuss the representative work with a particularly focus on commit messages analysis. Lastly, we turn our attention to the work of building datasets, which lays the foundation for research in the pertinent area.

3.1 Statistic features extraction

Conventional machine learning models often rely on feature engineering, which helps to represent an underlying problem with the most relevant and informative variables. We thus see many practical features targeting different research questions are proposed by prior works, as shown in Table 2. It should be noted that few papers use handcrafted features and learning-based textual features in a mixed manner such as [33]. In spite of this, we list the pertinent papers in Table 2, as long as handcrafted features are adopted. Accordingly, we further categorize them into four groups, and analyze them, respectively.

3.1.1 Security patch identification

Security patch identification is the process of identifying and tracking software patches that are released to address security vulnerabilities in a software system. Inasmuch as this technology plays a key role in the contemporary software eco-system, it has attracted significant interest from the academic community. For example, Tian et al. [13] leverage 55 features to describe the code diff. Also, the bag-of-word is adopted in their scheme to represent the commit message. Later, Wang et al. [14] only consider the code diff as the object of study, and propose using 61 features to form an input vector in their machine learning model, where 22 are borrowed from [13]. To achieve an acceptable precision, they use a voting algorithm that ensembles five classifiers including random forest, Bayes Net, stochastic gradient descent (SGD), sequential minimal optimization (SMO), and Bagging. It is worth noting that, in reality, developers in open-source software projects often do not pay sufficient attention to textual information when submitting patch commits. This results in a significant number of patch commits having low quality or even missing commit messages. Therefore, using only source code for patch analysis becomes more of a requirement than an option when textual information is unavailable. On the other hand, the following works [2, 15] consider both code changes and commit messages as the object of study. Especially, in [2], the 10 most recurrent non-stop words are extracted to represent the commit messages. By contrast, a set of predefined keywords based on authors' experience are used to depict the commit messages in [15]. Therefore we can see that the keywords-based strategies adopted by them are highly empirical and less objective. In addition, we briefly summarize the top 46 commonly used statistic features in related works, as shown in Table 3.

Besides, we consider that [36] is a quite different work comparing with the aforementioned ones. The observation of the authors is one reported issue may correspond to multiple bug-fix commits. In other words, some additional commits may be needed to supplement or correct an initial bug-fix commit. But in practice, the connections between these additional bug-fix commits and the issue reflected originally in a bug report are often missing. Consequently, the motivation of this work is to disclose those hidden supplementary bug-fix commits. To this end, the authors transform the original problem into a question of comparing the correlation between two commits.

3.1.2 Security patch localization

Given a disclosed vulnerability, security patch localization aims at searching a desired security patch commit in a repository. It has been regarded as an important aspect of software security, because it ensures that the affected users in different

Table 2 Overview of existing works on patch commits mining with manually extracted features

References	Year	Research question†	# of features	Learning model	Dataset availability	Object of study
Tian et al. [13]	2012	●	55	SVM+LPU [34]	×	Patch code and commit message
VCCFinder [35]	2015	○	17	SVM	×	Patch code with metadata
Ji et al. [36]	2018	●	8	SVM	×	Patch code and commit message
Wang et al. [14]	2019	●	61	Ensemble	✓	Patch code with metadata
Wang et al. [37]	2020	◑	21	Random forest	×	Patch code with metadata
Riom et al. [38]	2021	○	33	SVM	✓	Patch code with metadata
PatchScout [39]	2021	◐	22	RankNet [40]	×	Patch code and commit message
VCMatch [33]	2022	◐	36	Ensemble	✓	Patch code and commit message
SSPCatcher [2]	2022	●	33	SVM	✓	Patch code and commit message
VpatchFinder[15]	2022	●	40	Random forest	✓	Patch code and commit message

† ● Security patch identification; ◐ Security patch localization; ◑ Vulnerability type classification; ○ Vulnerable commit identification

Table 3 Statistic features that are commonly used by existing works

ID	Feature description	[13]	[14]	[2]	[15]
F1	# of changed files in a commit	✓	✓	✓	✓
F2	# of hunks in a commit	✓	✓		✓
F3–6	# of removed/added/total/net lines	✓	✓	✓	✓
F7–10	# of removed/added/total/net characters	✓	✓		✓
F11–14	# of removed/added/total/net conditional statements	✓	✓	✓	✓
F15–18	# of removed/added/total/net loops	✓	✓	✓	✓
F19–22	# of removed/added/total/net function calls	✓	✓	✓	
F23–26	# of removed/added/total/net logical operators	✓	✓	✓	✓
F27–30	# of removed/added/total/net arithmetic operators		✓		✓
F31–34	# of removed/added/total/net relation operators		✓		✓
F35–38	# of removed/added/total/net expressions	✓		✓	
F39–42	# of removed/added/total/net parenthesized expressions	✓		✓	
F43–46	# of removed/added/total/net assignments	✓		✓	

organizations have access to the latest security updates and can protect their systems from potential attacks or exploits. A representative work in this area is PatchScout [39], which transforms the security patches searching into a ranking problem on code commits. To this end, the authors propose utilizing 22 features between commits and vulnerabilities. Those features fall into four different categories: ① vulnerability identifier, ② vulnerability location, ③ vulnerability type, ④ vulnerability descriptive text. Besides, Wang et al. propose another security patch localization approach, VCMatch [33], involving 36 statistical features between the vulnerability and the patch commit. These handcrafted features can be divided into four groups: ① lines of code (LOC) features, ② identity features, ③ location features, ④ token features. Furthermore, VCMatch uses BERT [30] to generate semantic features of the vulnerability descriptions and commit messages. In the next phase, three classification models (i.e., XGBoost, LightGBM, and CNN) are trained. Finally, the results of these three classifiers are combined to make a prediction with a voting-based ranking fusion method.

3.1.3 Vulnerable commit identification

VCCFinder [35] builds upon the SVM model to automatically detect whether an incoming commit will introduce some vulnerabilities. In detail, the authors propose many statistic differences between vulnerability related and other commits. However, it is worth noting that, according to [38]: "*the original paper does not precisely list all the features extracted.*" Therefore, the number of features used by VCCFinder stated in Table 2 is only based on our understanding. Furthermore, Riom et al. [38] attempt to revisit both the approach and result of VCCFinder, but suffer a failure. Alternatively, they propose to build a variant approach for the detection of vulnerability-contributing commits. Lastly, their experimental results showcase that the performance reported in [35] cannot be achieved by the proposed variant.

3.1.4 Vulnerability type classification

Unlike other existing research that focuses on distinguishing between security and non-security patches, vulnerability type classification such as [37] aims to classify a given security patch according to the type of vulnerability it is fixing. At last, the thus obtained identification result can serve as a basis for the subsequent security patches prioritization.

In general, feature engineering is a crucial step in the machine learning pipeline, as it directly impacts the quality of the model's predictions. However, the feature engineering usually relies on heuristics or domain knowledge. For example, one heuristic adopted by [14] is "*security patches are more likely to modify less code than non-security patches*." Obviously, these heuristics are less objective and hard to be generalized.

3.2 Semantics learning

Representation learning by neural networks can replace manual feature engineering and allow the model to both learn the feature representations and use them to perform a specific task. Inspired by the advances in representation learning, a variety of approaches have been proposed to extract semantics of patch commits from the angle of graphs and natural languages. Table 4 summarizes the existing patch commits mining works that do not involve any handcrafted features. Furthermore, all of these works focus on the issue of security patch identification.

3.2.1 Graph-assisted approaches

To separate security patches from others, E-SPI [18] comprehensively considers both the patch diff and commit messages. In detail, a contextual abstract syntax tree (AST) is first extracted from the code changes. Then, the AST is decomposed into paths, which are later encoded by a BiLSTM (bidirectional Long Short-Term Memory). Furthermore, the tokens in a commit message are regarded as nodes, while the dependency relations between tokens are regarded as edges. Therefore, a commit message is transferred into a graph, which is further handled by a graph neural network (GNN). Moreover, a security patch detection system called GraphSPD is developed in [41], which proposes a novel graph structure, i.e., PatchCPG, to represent patches. Plus, the GNN is used to capture the semantics of code diff. Only patch code is considered by GraphSPD. Besides, in a recent work [42], the control dependence graph (CDG) and data dependence graph (DDG) are extracted to represent semantic and syntactic information of the code diff. Both graphs are decomposed into paths, which are then encoded using a BiLSTM with self-attention. Then, since the commit message is made of natural language text, another BiLSTM with self-attention is used for encoding the commit message.

3.2.2 NLP-assisted approaches

First, conventional NLP methods are used in [43] to build a security patch detector. In this scheme, both the log message and patch code are considered as the *bag of words*. According to their experience, the authors design different preprocessing methods to handle the log message and patch code, respectively. Then, each of the log messages are transformed into a vector of word counts. This is also done to the code for each patch. As a result, a *bag-of-words*-based classifier is contracted. However, though the experiments show a relative high precision of 80%, the recall is only 43%. This indicates many security patches in reality are mistakenly classified as non-security.

Considering the remarkable achievement is made in the NLP domain by using deep neural networks, more and more researchers shift their focus to this popular arsenal. For example, Wang et al. [3] take both commit messages and patches into account. They thus construct a LSTM-based detector. Their experiments demonstrate the total accuracy is 83.57%, but the corresponding false negative rate (FPR) is as high as 26.34%. Similarly, Zhou et al. [17] also consider the commit message as a sequence of tokens. So they choose a

Table 4 Overview of existing works on patch commits mining with semantic learning

References	Year	Semantic extraction	Classifier	Dataset availability	Object of study
Sabetta et al. [43]	2018	Bag of words	SVM	×	Commit message and patch code
VulFixMiner [16]	2021	CodeBERT [45]	MLP	×	Patch code with metadata
SPI [17]	2021	LSTM+CNN	Ensemble	✓	Commit message and patch code
PatchNet [44]	2021	CNN	MLP	✓	Commit message and patch code
PatchRNN [3]	2021	LSTM	MLP	✓ [50]	Commit message and patch code
E-SPI [18]	2022	BiLSTM+GNN	MLP	✓ [17]	Commit message and patch code
GraphSPD [41]	2023	GNN	MLP	✓ [17, 50]	Patch code with metadata
TMVDPatch [42]	2023	BiLSTM+Attention	MLP	✓ [50]	Commit message and patch code
Zuo et al. [46]	2023	Transformer	MLP	×	Commit message and comments

LSTM network to extract semantics from commit messages. For the code diff, they use two LSTM networks to encode the original code and the updated code, respectively. Then, a convolutional neural network (CNN) is applied to learn the difference between the two versions. This hybrid model reports a precision of 86.24% in experiments. Besides, in [44], both commit messages and code changes are handled by CNN-based model.

In another work called VulFixMiner [16], the authors only consider code change information. To extract semantics from the code changes, they particularly adopt CodeBERT [45], which is a pre-trained model targeting programming language. It is noteworthy that VulFixMiner only investigates Python and Java projects. This limitation comes from that the existing CodeBERT are pre-trained merely on six programming languages, i.e., Python, Java, JavaScript, PHP, Ruby, and Go. However, it has been widely acknowledged that C/C++ are the languages containing the highest number of vulnerabilities [3, 14, 35, 37].

3.3 Commit messages study

Though previous works consistently take code changes as the object of study, a very recent work [46] shifts the focus to exploring how much the textual description alone can impact on the final security patch detection. To this end, the researchers propose a Transformer [47]-based security patch detector targeting commit messages and comments. The outcome of this research further confirms the critical importance of well-crafted commit messages for the later software maintenance. On the other hand, there exist a large number of patch commits in reality, where the textual information is often of poor quality or even missing. Therefore, relying solely on the analysis of commit messages is inadequate. In addition, Zhou et al. [48] implement an application to identify vulnerabilities based on commit messages and bug reports. More concretely, they use Word2Vec [27] to generate a 400-dimensional embedding and a 200-dimensional embedding for a commit message and a bug report, respectively. Then they use a K-fold stacking algorithm that ensembles six basic classifiers (i.e., random forest, Gaussian naive Bayes, KNN, SVM, gradient boosting, and AdaBoost) to make the final decision. Another distinct work [49] applies sentiment analysis techniques to explore emotional variations in 24,000 bug-introducing and bug-fixing commit messages over three OSS projects. We can see that NLP techniques play a significant role when the commit messages are solely analyzed.

3.4 Dataset construction

Unlike computer vision and natural language processing that have a large body of well-labeled data, "*vulnerability datasets are a precious (and scarce) resource*" [13]. Many researchers believe that the unavailability of data is a common obstacle for computer security. For example, Riom et al. [38] claim that "*dataset of the original VCCFinder* [35] *article is not directly accessible*." Similarly, Hoang et al. [44] state that the data provided in [13] "*is seven years old and unclean*." Other works such as [17] only make their data partially public. Fortunately, we observe that the issue of dataset availability has raised awareness of the research community. Some recent works targeting the dataset construction have made their resources publicly accessible, for example, [50–52].

Although researchers may customize their dataset construction procedures in terms of different practical requirements, herein we briefly describe a routine method to build up a security patch dataset. Each entry in the CVE database includes a CVE number, a succinct description of the vulnerability, and references (usually in the form of website addresses to external resources). For example, for the project hosted on GitHub,[2] the website address referring to a security related commit has a format as

```
https://github.com/owner/repo/commit/
commit_hash
```

where the `commit_hash` is a SHA1 value that is used as the unique identifier of a commit. Therefore, we can crawl all the reference website addresses in CVE entries and filter out those matching the above rule. After this step, by appending a suffix "`.patch`" at the end of one website address, we can get a patch URL. The corresponding patch file can be retrieved by `wget` according to the given patch URL. For example,

```
https://github.com/jasper-software/
jasper/commit/fd564ee3377d9fc248
4c657e4f464a3fb9764d31.patch
```

is the patch URL for CVE-2021-27845. Furthermore, non-security patches, which are not connected with any CVE records, can be randomly retrieved from GitHub by a web crawler. But manually double-check is often needed to avoid unreported security patches. It is worth noting that though we take GitHub as an example, this method also can be generalized to other platforms that provide Git repositories as a service such as GitLab.[3]

To train a robust machine learning model, a large-scale dataset is usually indispensable. However, the security patches that can be collected by merely using the aforementioned basic solution is limited. To address this challenge, Wang et al. [50] propose *nearest link search* to discover more candidate patches from code repositories. These candidate patches are very likely to be security related because they

[2] https://github.com/.

[3] https://about.gitlab.com/.

have similar features with verified security patches. Following human verification, certain candidates may be labeled as security patches and included in the existing dataset. Finally, they also generate synthetic data based on oversampling to further enrich the dataset.

4 Empirical analysis on security patches

We notice that a few empirical investigations on security patches have been carried out, such as [53, 54]. Unlike the aforementioned research, the core objective of these works is to describe phenomena and provide insights mainly with observations and quantitative methods rather than to discover vulnerabilities. Especially, more than 4000 security patches for over 3000 vulnerabilities from 682 OSS projects are empirically analyzed in [53]. Based on this, the authors concluded that patches were effective in mitigating the security vulnerabilities they addressed, but were not always installed in a timely manner. The study also found that certain types of software vulnerabilities were more likely to receive patches than others. For example, they found "*web-oriented vulnerabilities like SQL injection have significantly shorter life spans compared to errors in memory management*." Similarly, 3663 vulnerabilities pertaining to 1096 OSS projects are empirically studied in [54]. The findings of this study indicate that "*vulnerabilities have high survivability rates in terms of days they stay in the source code*." In their investigation, half of vulnerabilities remain unfixed for at least more than one year.

5 Research on bug report

In modern software maintenance practice, bug reports are closely related to patch commits because multiple commits are often required to fix issues in a bug report. In other words, the bug reports and patch commits have a one-to-many relationship, where many patch commits may be associated with a single bug report. Not only that, they also can complement each other in discovering silent vulnerabilities. Hence, there exist some papers that attempt to link patch commits with bug reports, for example, [55, 56]. To facilitate in-depth explorations for interested readers in this interdisciplinary field, we briefly explain the potential research questions in bug report study, and review the latest progress. But, given the large body of research on the pertinent topic, we only cover the most related works.

5.1 Bug report classification and prioritization

The repositories of OSS projects usually collect bugs reported by users, developers, and other third-parties from all over the world, and maintain the corresponding bug-fixing patches as well. However, due to the gap between the increasing number of bugs and the limited manpower, not all bugs can be fixed in a timely way. In practice, before generating a feasible patch, security bugs and non-security bugs should be first triaged and isolated, especially considering the security bugs often cause more serious impacts. Thus, automatically classifying and prioritizing bug reports is indispensable when developing and maintaining software.

Similar to the patch commits analysis, classical machine learning techniques have been widely applied in bug report classification and priority prediction [57–59]. However, we are more interested in the advances of deep learning-based approaches. For example, Umer et al. [60] propose a method for predicting the priority of bug reports through a model based on CNN. Each word is represented using a Word2Vec embedding [27]. These vectors are fed into a CNN-based classifier along with the sentiment analysis result of the bug reports. Later, Fang et al. [61] first construct a text graph for bug reports and then apply GCNs to extract semantics from bug reports. In addition, Li et al. [62] use the attention mechanism to extract semantics and further make a priority prediction.

5.2 Duplicate bug report identification

Duplicate bug reports are very common in software development because multiple users or testers may encounter the same issue and report it independently. When several bug reports describe the same issue, it can be hard to prioritize and address each report individually. Therefore, automatically identifying and managing duplicate bug reports is crucial for effective bug tracking and resolution. If duplicates were identified and consolidated into a single report, development teams could save time and resources, avoid redundancy, and focus on resolving the underlying issue.

Overall, taking advantage of information retrieval (IR) and NLP techniques to analyze the description, keywords, and other relevant information in each report is the main strategy of duplicate bug report identification. For example, Sun et al. [63] extend the IR model BM25F [64] to improve the detection accuracy. In [65], the authors apply both the Latent Dirichlet Allocation (LDA) topic model and IR techniques to detect duplicate bug reports. Later, a *bag-of-words*-based approach is proposed in [66]. By this means, each bug report is represented by a numeric vector, the element of which is the number of occurrences of a certain word. Furthermore, Deshmukh et al. [67] apply deep neural networks to detect duplicate bug reports. Their model is built using Siamese CNN and LSTM. In addition, Buhiraja et al. [68] create a matrix representation of each bug report using Word2Vec [27]. Pairs of embeddings are then fed to

Table 5 Security patches collected from two public datasets, which are mistakenly labeled as non-security[†]

Source repository (commit hash)	Commit date	Vulnerability
ImageMagick(`e5fd9ab1b7...2f0`)	12/01/2016	④
ImageMagick(`c65a90950a...c93`)	12/02/2016	④
ImageMagick(`b0e61972ff...0a3`)	04/07/2017	②
ImageMagick(`3690a82092...4e4`)	05/02/2017	②
ImageMagick(`1f5e947893...39e`)	07/31/2017	③
ImageMagick(`5304ae1465...4f0`)	08/12/2017	④
ImageMagick(`c72eac2447...481`)	08/21/2019	③
ImageMagick(`6d095e5ac0...761`)	08/21/2019	④
ImageMagick(`c68d49b80d...84d`)	10/22/2019	⑥
ImageMagick(`d9639fd557...bc7`)	05/28/2020	⑥
php-src(`9ee23d7066...747`)	04/23/2004	④
php-src(`35e0565a4b...657`)	06/21/2008	②③
php-src(`fbdd2f3e5e...6b2`)	11/02/2011	①
php-src(`35e0565a4b...8a3`)	11/02/2015	⑤
php-src(`2eaa755660...7ef`)	12/08/2015	②④
php-src(`a63d0f55da...59c`)	01/25/2016	②
php-src(`0f20e113c2...182`)	05/03/2016	③
php-src(`e0f5d62bd6...699`)	03/03/2019	④
php-src(`f79c774274...89f`)	01/20/2020	⑤
php-src(`a681b12820...635`)	03/30/2020	①

[†] ① Integer overflow; ② Memory leak; ③ Double free;
[†] ④ Buffer overflow; ⑤ Use after free; ⑥ Divide by zero

a neural network to predict the similarity of the two bug reports. These methods can greatly help streamline the bug tracking process and improve overall efficiency in software maintenance.

5.3 Empirical study on bug report

Some empirical studies on bug reports have been done in previous works [69–71]. These researches share the similar research question, that is to explore whether security bugs are treated with a higher priority in practice. More concretely, the quantitative analysis in [70, 71] are only carried out in Chromium browser project. In particular, when documenting a bug in the repository, the report follows a keywords-based labeling criteria. Consequently, both [70, 71] adopt a straightforward method to classify the bugs, that is by searching the keywords "security" and "performance" directly in the bug reports. Besides, the case study in [69] merely involves the Firefox browser project. The bug classification is also completed based on a pattern matching strategy, where manual efforts and expertise are required for pre-defining a set of keywords. More importantly, since the researches are limited to specific projects, extending their approaches to other software is impractical.

6 Research gaps and challenges

In this section, we provide our insights on the remaining issues in this field.

6.1 Dataset quality

In data-driven applications, it is critical to maintain data quality and integrity, because the contamination or degradation of data quality can lead to serious consequences in the final decision making. However, unlike NLP or computer vision domain that have a large volume of well-labeled data such as ImageNet,[4] the unavailability of data is a common obstacle for computer security [38, 44, 72]. While some groups have open-sourced their dataset, data pollution is observed under our investigation. More concretely, we carry out a case study on the datasets offered by [15, 51]. As a result, we found a large number of security patches are incorrectly labeled as non-security. For example, Table 5 presents 20 concrete samples among them considering space constraints, where half entries related to `ImageMagick` are from [51], while others pertaining to the `php-src` project come from [15]. The first column of Table 5 shows the hash value used as the unique identifier of each patch commit in source repository. Each

[4] https://www.image-net.org/.

value in the commit date column, that is directly obtained from the patch, represents the vulnerability fix date. Finally, we annotate the vulnerability type for each entry using the third column. It should be noted that what we disclose here is just the tip of the iceberg considering our limited manpower to manually examine the dataset.

6.2 High false alarms

It is worth pointing out that the distribution of security and non-security patches is usually unbalanced. Most patches should be not be related to vulnerabilities fixing in reality. They may be used for various purposes, such as functionality updates or performance enhancements. Therefore, the false-positive rate (FPR) and false-negative rate (FNR) are two important measurements to evaluate the model performance. A higher FPR indicates that more non-security patches are mistakenly predicted as security related, while a higher FNR means the system misses more security patches and predicts them as non-security. However, the existing works usually either underperform in terms of FPR and FNR or neglect reporting these two values. For instance, Table 6 shows the FPR and TPR of three works. Among them, the FPR of [14] can arrive at 41.3%, while the FNR of [3] is also as high as 26.3%. We consider the issue of high false alarms should attract enough attention in future when developing security patch detection systems.

6.3 Importance of textual information

The patch commits contain a significant amount of very useful textual information. However, in the past, researchers often intentionally or unintentionally overlooked this treasure trove. One concern is practitioners often underemphasize the textual information when submitting patches. As a result, we found there exist a large number of patch commits in reality having poor quality or even missing commit messages [73]. It should be noted that the textual information herein should refer to three aspects, i.e., commit messages, code comments, and updated log files.

First, the commit message, as an important component of a patch commit, should raise sufficient awareness in the open-source community. Though a diff may show how the code is changed, only the commit message can properly inform other developers or third-party users of code snippets the reason why code is updated in this way. Unfortunately, our investigation shows massive commits in reality lack well-crafted messages. For example, in the repository of `ImageMagick`, a majority of commit messages are either meaningless characters such as "`...`" or just a URL link. By contrast, the prior successful experience in the bug report management has taught us a lesson. For example, some previous works [22, 23] focus on offering practical recommendations on how to generate a good report. Not only that, existing bug tracking systems also have demonstrated an advisable practice in well guiding users to submit the qualified reports. However, similar work in the domain of patch commits is still a gap waiting to be filled.

Second, some patch code may be associated with comments that are written in natural language. These textual descriptions about the code usually can serve as a form of documentation. Therefore, when analyzing the patches, the code comments should be exploited as much as possible.

Third, we have realized that when submitting a patch commit, many contributors would update not only the source code files but also the log files. What's more, the log files usually contain detailed textual explanations about the updates. We do believe that analyzing the log files would be beneficial for improving the performance of current work. Thus, we propose this as a potential direction for the future.

Table 6 Comparisons of different methods based on FPR and FNR

	E-SPI [18]	PatchRNN [3]	Wang et al. [14]
FPR	12.5%	11.6%	41.3%
FNR	8.8%	26.3%	N/A

7 Conclusion

In the process of software development and maintenance, there are often a large number of patch commits involved. Among them, some security patches that are used to fix known vulnerabilities particularly attract more attention, because unpatched vulnerabilities provide possibilities for compromising the software system. Therefore, patch commits mining toward vulnerability discovery arouses great interest of researchers. However, so far we have not seen a comprehensive survey summarizing and discussing the current advances as well as the challenges need to be addressed within this field. To fill this gap, we survey, evaluate, and systematize a list of literature with providing our insights on both successes and remaining issues. Not only that, the introductory panorama with replicable hands-on experience presented in this paper also can help readers quickly understand and step into this field. We hope the proposed challenges and open opportunities can inspire more and more researchers to participate in this study.

Data Availability Data sharing is not applicable to this article as no datasets were generated during the study. The papers discussed in this review are listed at https://github.com/fzuo/Patch-Commits-Study.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Silverman, R.E.: Git Pocket Guide: A Working Introduction. O'Reilly Media, Inc., Sebastopol (2013)
2. Sawadogo, A.D., Bissyandé, T.F., Moha, N., Allix, K., Klein, J., Li, L., Le Traon, Y.: SSPCatcher: learning to catch security patches. Empir. Softw. Eng. **27**(6), 151 (2022)
3. Wang, X., Wang, S., Feng, P., Sun, K., Jajodia, S., Benchaaboun, S., Geck, F.: PatchRNN: a deep learning-based system for security patch identification. In: Proceedings of the IEEE Military Communications Conference (MILCOM), pp. 595–600 (2021)
4. CVE: Published CVE records (2023). https://www.cve.org/About/Metrics. Accessed 03 2023
5. Snyk: The state of open-source security (2017). https://snyk.io/series/open-source-security/. Accessed 12 2018
6. Snyk: The state of open source security report (2019). https://snyk.io/series/open-source-security/. Accessed 08 2020
7. Liang, H., Pei, X., Jia, X., Shen, W., Zhang, J.: Fuzzing: state of the art. IEEE Trans. Reliab. **67**(3), 1199–1218 (2018)
8. Baldoni, R., Coppa, E., D'celia, D.C., Demetrescu, C., Finocchi, I.: A survey of symbolic execution techniques. ACM Comput. Surv. (CSUR) **51**(3), 1–39 (2018)
9. Luo, L., Zeng, Q., Yang, B., Zuo, F., Wang, J.: Westworld: fuzzing-assisted remote dynamic symbolic execution of smart apps on IoT cloud platforms. In: Proceedings of the Annual Computer Security Applications Conference, pp. 982–995 (2021)
10. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: towards real-time object detection with region proposal networks. In: Advances in Neural Information Processing Systems, vol. 28 (2015)
11. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
12. Zuo, F., Yang, B., Li, X., Zeng, Q.: Exploiting the inherent limitation of L0 adversarial examples. In: Proceedings of the 22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID), pp. 293–307. USENIX Association (2019)
13. Tian, Y., Lawall, J., Lo, D. Identifying Linux bug fixing patches. In: Proceedings of the 34th International Conference on Software Engineering (ICSE), pp. 386–396. IEEE (2012)
14. Wang, X., Sun, K., Batcheller, A., Jajodia, S.: Detecting 0-day vulnerability: an empirical study of secret security patch in OSS. In: Proceedings of the 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 485–492 (2019)
15. Huang, C., Sun, M., Duan, R., Susheng, W., Chen, B.: Vulnerability identification technology research based on project version difference. Chin. J. Netw. Inf. Secur. **8**(1), 52–62 (2022)
16. Zhou, J., Pacheco, M., Wan, Z., Xia, X., Lo, D., Wang, Y., Hassan, A.E.: Finding a needle in a haystack: automated mining of silent vulnerability fixes. In: Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 705–716 (2021)
17. Zhou, Y., Siow, J.K., Wang, C., Liu, S., Liu, Y.: SPI: automated identification of security patches via commits. ACM Trans. Softw. Eng. Methodol. (TOSEM) **31**(1), 1–27 (2021)
18. Wu, B., Liu, S., Feng, R., Xie, X., Siow, J., Lin, S.-W.: Enhancing security patch identification by capturing structures in commits. IEEE Transactions on Dependable and Secure Computing (2022)
19. Xu, Z., Chen, B., Chandramohan, M., Liu, Y., Song, F.: Spain: security patch analysis for binaries towards understanding the pain and pills. In: Proceedings of the IEEE/ACM 39th International Conference on Software Engineering (ICSE), pp. 462–472 (2017)
20. Zuo, F., Li, X., Young, P., Luo, L., Zeng, Q., Zhang, Z.: Neural machine translation inspired binary code similarity comparison beyond function pairs. In: Proceedings of the 26th Network and Distributed Systems Security (NDSS) Symposium (2019)
21. Dissanayake, N., Jayatilaka, A., Zahedi, M., Ali Babar, M.: Software security patch management-a systematic literature review of challenges, approaches, tools and practices. Inf. Softw. Technol. **144**, 106771 (2022)
22. Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., Zimmermann, T.: What makes a good bug report? In: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, pp. 308–318 (2008)
23. Zimmermann, T., Premraj, R., Bettenburg, N., Just, S., Schroter, A., Weiss, C.: What makes a good bug report? IEEE Trans. Softw. Eng. **36**(5), 618–643 (2010)
24. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: Proceedings of the 5th International Conference on Learning Representations (ICLR) (2017)
25. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. In: Proceedings of the 5th International Conference on Learning Representations (ICLR) (2018)
26. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Advances in Neural Information Processing Systems, vol. 30 (2017)
27. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems, vol. 26 (2013)
28. Pagliardini, M., Gupta, P., Jaggi, M.: Unsupervised learning of sentence embeddings using compositional n-gram features. In: Proceedings of Conference of the North American Chapter of the Association for Computational Linguistics (NAACL) (2018)
29. Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: Proceedings of the International conference on machine learning, pp. 1188–1196 (2014)
30. Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding (2018). arXiv:1810.04805
31. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I.: Improving language understanding by generative pre-training. Technical report, OpenAI (2018)
32. Radford, A., Jeffrey, W., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners. Technical report, OpenAI (2019)
33. Wang, S., Zhang, Y., Bao, L., Xia, X., Wu, M.: Vcmatch: a ranking-based approach for automatic security patches localization for OSS vulnerabilities. In: Proceedings of the IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 589–600 (2022)
34. Li, X., Liu, B.: Learning to classify texts using positive and unlabeled data. In: Proceedings of the International Joint Conference on Artificial Intelligence, vol. 3, pp. 587–592 (2003)
35. Perl, H., Dechand, S., Smith, M., Arp, D., Yamaguchi, F., Rieck, K., Fahl, S., Acar, Y.: 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, pp. 426–437 (2015)

36. Ji, T., Pan, J., Chen, L., Mao, X.: Identifying supplementary bug-fix commits. In: Proceedings of the 42nd Annual Computer Software and Applications Conference (COMPSAC), pp. 184–193. IEEE (2018)
37. Wang, X., Wang, S., Sun, K., Batcheller, A., Jajodia, S.: A machine learning approach to classify security patches into vulnerability types. In: Proceedings of the IEEE Conference on Communications and Network Security (CNS), pp. 1–9 (2020)
38. Riom, T., Sawadogo, A., Allix, K., Bissyandé, T.F., Moha, N., Klein, J.: Revisiting the VCCFinder approach for the identification of vulnerability-contributing commits. Empir. Softw. Eng. **26**, 1–30 (2021)
39. Tan, X., Zhang, Y., Mi, C., Cao, J., Sun, K., Lin, Y., Yang, M.: Locating the security patches for disclosed OSS vulnerabilities with vulnerability-commit correlation ranking. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, pp. 3282–3299 (2021)
40. Burges, C.J.C.: From ranknet to lambdarank to lambdamart: an overview. Learning **11**(23–581), 81 (2010)
41. Wang, S., Wang, X., Sun, K., Jajodia, S., Wang, H., Li, Q. Graph-spd: graph-based security patch detection with enriched code semantics. In: Proceedings of the IEEE Symposium on Security and Privacy (SP), pp. 604–621 (2022)
42. Zhou, X., Pang, J., Shan, Z., Yue, F., Liu, F., Jinlong, X., Wang, J., Liu, W., Liu, G.: TMVDPatch: a trusted multi-view decision system for security patch identification. Appl. Sci. **13**(6), 3938 (2023)
43. Sabetta, A., Bezzi, M.: A practical approach to the automatic classification of security-relevant commits. In: Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 579–582 (2018)
44. Hoang, T., Lawall, J., Tian, Y., Oentaryo, R.J., Lo, D.: PatchNet: hierarchical deep learning-based stable patch identification for the Linux kernel. IEEE Trans. Softw. Eng. **47**(11), 2471–2486 (2021)
45. Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, and D., Zhou, M.: CodeBERT: a pre-trained model for programming and natural languages. In: Findings of the Association for Computational Linguistics: EMNLP 2020, pp. 1536–1547. Association for Computational Linguistics (2020)
46. Zuo, F., Zhang, X., Song, Y., Rhee, J., Fu, J.: Commit message can help: security patch detection in open source software via transformer. In: IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA), pp. 345–351 (2023)
47. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems, vol. 30 (2017)
48. Zhou, Y., Sharma, A.: Automated identification of security issues from commit messages and bug reports. In: Proceedings of the 11th Joint Meeting on Foundations of Software Engineering, pp. 914–919 (2017)
49. Islam, M.R., Zibran, M.F.: Sentiment analysis of software bug related commit messages. In: Proceedings of the 27th International Conference on Software Engineering and Data Engineering (SEDE), pp. 3–8 (2018)
50. Wang, X., Wang, S., Feng, P., Sun, K., Jajodia, S.: PatchDB: a large-scale security patch dataset. In: Proceedings of the 51st annual IEEE/IFIP international conference on dependable systems and networks (DSN), pp. 149–160 (2021)
51. Reis, S., Abreu, R.: A ground-truth dataset of real security patches. arXiv:2110.09635 (2021)
52. Zuo, F., Rhee, J., Kim, Y., Oh, J., Qian, G.: A comprehensive dataset towards hands-on experience enhancement in a research-involved cybersecurity program. In: The 24th ACM Annual Conference on Information Technology Education, pp. 118–124 (2023)
53. Li, F., Paxson, V.: A large-scale empirical study of security patches. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, pp. 2201–2215 (2017)
54. Iannone, E., Guadagni, R., Ferrucci, F., De Lucia, A., Palomba, F.: The secret life of software vulnerabilities: a large-scale empirical study. IEEE Trans. Softw. Eng. **49**(1), 44–63 (2022)
55. Zhong, H., Su, Z.: An empirical study on real bug fixes. In: Proceedings of the 37th International Conference on Software Engineering, vol. 1, pp. 913–923. IEEE (2015)
56. Ferenc, R., Gyimesi, P., Gyimesi, G., Tóth, Z., Gyimóthy, T.: An automatically created novel bug dataset and its validation in bug prediction. J. Syst. Softw. **169**, 110691 (2020)
57. Tian, Y., Lo, D., Xia, X., Sun, C.: Automated prediction of bug report priority using multi-factor analysis. Empir. Softw. Eng. **20**, 1354–1383 (2015)
58. Shu, R., Xia, T., Chen, J., Williams, L., Menzies, T.: How to better distinguish security bug reports (using dual hyperparameter optimization). Empir. Softw. Eng. **26**, 1–37 (2021)
59. Ahmed, H.A., Bawany, N.Z., Shamsi, J.A.: CaPBug-a framework for automatic bug categorization and prioritization using NLP and machine learning algorithms. IEEE Access **9**, 50496–50512 (2021)
60. Umer, Q., Liu, H., Illahi, I.: Cnn-based automatic prioritization of bug reports. IEEE Trans. Reliab. **69**(4), 1341–1354 (2019)
61. Fang, S., Tan, Y., Zhang, T., Zhou, X., Liu, H.: Effective prediction of bug-fixing priority via weighted graph convolutional networks. IEEE Trans. Reliab. **70**(2), 563–574 (2021)
62. Li, Y., Che, X., Huang, Y., Wang, J., Wang, S., Wang, Y., Wang, Q.: A tale of two tasks: automated issue priority prediction with deep multi-task learning. In: Proceedings of the 16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 1–11 (2022)
63. Sun, C., Lo, D., Khoo, S.-C., Jiang, J.: Towards more accurate retrieval of duplicate bug reports. In: 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 253–262. IEEE (2011)
64. Robertson, S., Zaragoza, H., Taylor, M.: Simple bm25 extension to multiple weighted fields. In: Proceedings of the 13th International Conference on Information and Knowledge Management (CIKM), pp. 42-49. ACM (2004)
65. Nguyen, A.T., Nguyen, T.T., Nguyen, T.N., Lo, D., Sun, C.: Duplicate bug report detection with a combination of information retrieval and topic modeling. In: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, pp. 70–79 (2012)
66. Gopalan, R.P., Krishna, A.: Duplicate bug report detection using clustering. In: Proceedings of the 23rd Australian Software Engineering Conference, pp. 104–109. IEEE (2014)
67. Deshmukh, J., Annervaz, K.M., Podder, S., Sengupta, S., Dubash, N.: Towards accurate duplicate bug retrieval using deep learning techniques. In: 2017 IEEE International conference on software maintenance and evolution (ICSME), pp. 115–124. IEEE (2017)
68. Budhiraja, A., Dutta, K., Reddy, R., Shrivastava, M.: DWEN: deep word embedding network for duplicate bug report detection in software repositories. In: Proceedings of the 40th International Conference on software engineering: companion proceeedings, pp. 193–194 (2018)
69. Zaman, S., Adams, B., Hassan, A.E.: Security versus performance bugs: a case study on Firefox. In: Proceedings of the 8th Working Conference on Mining Software Repositories, pp. 93–102 (2011)
70. Imseis, J., Nachuma, C., Arifuzzaman, S., Zibran, M., Bhuiyan, Z.A.: On the assessment of security and performance bugs in chromium open-source project. In: Proceedings of the 5th International Conference on Dependability in Sensor, Cloud, and Big Data Systems and Applications, pp. 145–157 (2019)
71. Rajbhandari, A., Zibran, M.F., Eishita, F.Z.: Security versus performance bugs: How bugs are handled in the chromium project.

In: Proceedings of the 20th IEEE/ACIS International Conference on Software Engineering Research, Management and Applications (SERA), pp. 70–76 (2022)
72. Shrestha, M., Kim, Y., Oh, J., Rhee, J., Choe, Y.R., Zuo, F., Park, M., Qian, G.: Provsec: Cybersecurity system provenance analysis benchmark dataset. In: IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA), pp. 352–357 (2023)
73. Tian, Y., Zhang, Y., Stol, K.-J., Jiang, L., Liu, H.: What makes a good commit message? In: Proceedings of the 44th International Conference on Software Engineering, pp. 2389–2401 (2022)