Characteristics, Root Causes, and Detection of Incomplete Security Bug Fixes in the Linux Kernel

Qiang Liu[1]*, Wenlong Zhang*[1]*, Muhui Jiang*[2,1]*, Lei Wu*[1]*, Yajin Zhou*[1]
[1]*Zhejiang University,* [2]*The Hong Kong Polytechnic University*

Abstract

Security bugs in the Linux kernel emerge endlessly and have attracted much attention. However, fixing security bugs in the Linux kernel could be incomplete due to human mistakes. Specifically, an incomplete fix fails to repair all the original security defects in the software, fails to properly repair the original security defects, or introduces new ones.

In this paper, we study the fixes of incomplete security bugs in the Linux kernel for the first time, and reveal their characteristics, root causes as well as detection. We first construct a dataset of incomplete security bug fixes in the Linux kernel and answer the following three questions. What are the characteristics of incomplete security bug fixes in the Linux kernel? What are the root causes behind them? How should they be detected to reduce security risks? We then have the three main insights in the following.

Insight 1: The fixes of incomplete security bugs in the Linux kernel are gradually decreasing, and happened occasionally recently; their incubation period is 2 to 7 months; they are more likely to appear when denial of service vulnerabilities need to be fixed; the number of files and code changes affected by related patches is generally small.

Insight 2: There are three common root causes behind the incomplete security bug fixes in the Linux kernel. First, a developer was misled by superficial phenomena of the crash before the fix; second, a developer did not pay attention to the similar modules during the fix; third, a developer introduced a semantic error after the fix.

Insight 3: Aiming at the root cause of "not paying attention to similar modules when fixing", we propose a fix-pattern-based detection algorithm. Using this algorithm, we successfully find a new case of and a hidden case of incomplete security bug fixes in the Linux kernel. Compared with most of the previous empirical research that stays in suggestion, this paper has a certain degree of advancement.

*All work was done by Aug., 2022.

1 Introduction

Linux kernel, as a critical component of both the open-source software supply chain and enterprise core software, has long been a prime target for attackers and faces substantial security risks. To date, more than 2,650 security flaws with assigned CVE identifiers have been discovered in the Linux kernel [1]. These vulnerabilities inevitably impose varying degrees of security and operational risks on downstream open-source software, end users relying on the Linux kernel, and even enterprises and government agencies.

Security flaws in open-source software emerge continuously, and stakeholders expect these flaws to be correctly fixed. However, studies have shown that *incomplete security bug fixes* account for 12% of all security fixes, respectively [11], exerting a non-negligible impact on the security of open-source ecosystems. An incomplete security bug fix refers to a situation where not all code affected by a security flaw is fully patched, thereby leaving the original flaw partially or entirely unfixed, or the patch itself contains mistakes, either failing to remediate the original flaw or introducing a new one. Evidently, the existence of such incomplete fixes continues to expose open-source software to potential exploitation. Understanding their characteristics, root causes, and effective prevention measures is therefore of significant value to both researchers and software maintainers.

To reveal the characteristics, underlying causes, and detection strategies of incomplete security bug fixes in the Linux kernel, this paper constructs the first dataset dedicated to this phenomenon. First, using quantitative research methods, we identify and characterize patterns exhibited by incomplete fixes. Although security bug fixing in the Linux kernel follows stricter processes than general bug fixing, our findings demonstrate that developers involved in fixing security flaws still make mistakes. Next, through an empirical study, we analyze for the first time the root causes of 26 incomplete security bug fixes in the Linux kernel from the perspectives of pre-fix, in-fix, and post-fix activities, and provide improvement suggestions for the overall security bug fixing workflow.

Finally, we conduct an in-depth investigation of a prevalent category of incomplete fixes, those arising from overlooking analogous modules during patch development. By leveraging FixMiner [5], a semantic-aware syntax-tree–based similarity analysis tool, we identify two previously unknown incomplete security bug fixes in the Linux kernel. This tool can assist in detecting this specific type of incomplete security fix and serves as a complementary approach to traditional Linux kernel regression testing.

Although prior empirical studies have examined incomplete bug fixes [4, 13, 16, 17, 25], this work differs in both content and practical contributions in two key aspects (see Section 6). First, existing studies primarily focus on non-system software such as Eclipse and Mozilla, whereas our study targets the Linux kernel, a representative and complex system software. Second, previous empirical studies often stop at providing statistical summaries or qualitative recommendations. In contrast, we go beyond characterization and, based on our empirical findings, leverage fix-pattern fingerprints to discover two previously unreported incomplete security bug fixes in the Linux kernel (both have been fixed).

Contributions. This paper has the following contributions:

- First dataset of incomplete security bug fixes in the Linux kernel at https://doi.org/10.5281/zenodo.6423844, enabling systematic study of this long-overlooked phenomenon.
- Empirical characterization of incomplete security bug fixes. Through quantitative and qualitative analysis of 26 cases, we reveal their temporal trends, vulnerability types, patch characteristics, and the recurring root causes across pre-fix, in-fix, and post-fix stages.
- Fix-pattern–based detection method. We design a detection algorithm based on fix-pattern fingerprinting using FixMiner. Applying it to 1,615 Linux kernel CVEs uncovers two previously unknown incomplete security fixes, demonstrating the method's effectiveness.

2 Dataset and Research Questions

This section describes how we construct the dataset of incomplete security bug fixes in the Linux kernel and the three research questions on the characteristics, root causes, and detection of incomplete security bug fixes.

2.1 Dataset

The dataset of incomplete security bug fixes refers to a collection of patch sets in which the initial fix of a security flaw is either incomplete. Each entry in this dataset contains two or more patches. The first patch corresponds to the initial fix of a security vulnerability (denoted as Fix-0), and any subsequent patches (Fix-1, Fix-2, and so on) further correct the incompleteness or mistakes in Fix-0. Each patch consists of two components: a textual description of the vulnerability being addressed and the repair strategy used, and the corresponding patch code. In addition, we enrich each entry with the vulnerability type using the classification scheme published by CVE Details ("The ultimate security vulnerability datasource") [2]. Finally, each dataset entry is indexed by the CVE (Common Vulnerabilities & Exposures) identifier associated with Fix-0.

Constructing a comprehensive dataset of incomplete security bug fixes is highly challenging. For instance, Li et al. [11], in their study of security patches, identified fixes by extracting patch links listed in the reference section of each CVE entry, which is easy to automate. However, it is unsuitable for identifying incomplete security fixes, because real-world CVE reference links only include the patch that claims to fix the vulnerability, and never include information about subsequent corrective patches. Information (such as *incomplete* or *incorrect*) is typically found in the textual descriptions of the vulnerability or the surrounding discussion. Furthermore, not all Linux kernel security fixes are assigned CVE identifiers, and identifying which patches are security, related remains a difficult problem. Although machine learning–based techniques exist [31], they still require manual inspection and labeling by security experts for training and evaluation. To date, no public dataset or automated tool exists for determining whether a Linux kernel patch is security-related.

In this work, we retrieved data from 17 bug-tracking platforms and vulnerability databases and ultimately collected 26 incomplete security bug fix instances from the Linux kernel, covering 52 CVEs. Our approach uses two complementary search strategies. First, we performed a forward search on the CVE Mitre platform using the keywords *incomplete*, *incorrect*, and *Linux kernel*. Second, we conducted a reverse search by querying only *incomplete* and *incorrect* on the CVE Mitre platform, obtaining more than 300 results across various projects and manually filtering those affecting the Linux kernel. We then applied the same strategies across the remaining 16 platforms and databases listed in Table 1. Through manual consolidation and deduplication, we obtained 26 incomplete security bug fix instances and further enriched them with patch descriptions, patch code, and vulnerability type information.[1] The dataset is publicly available at https://doi.org/10.5281/zenodo.6423844 for the research community.

2.2 Research Questions

Based on the constructed dataset of incomplete security bug fixes, we investigate the following three research questions, progressing from phenomenon-level characterization to causal analysis and finally to detection.

- **Research Question 1: What are the characteristics of incomplete security bug fixes?** We first study the temporal distribution of incomplete security bug fix instances in the Linux kernel to understand the phenomenon longitudinally (RQ1.1). We then examine the time lag between Fix-0 and subsequent fixes, which reflects the

[1] CVE-2012-3552 is also incomplete, but its patch is unavailable and therefore excluded.

Table 1: Bug-tracking platforms and vulnerability databases

Platform/Database	Link	Description
Red Hat Bugzilla	https://bugzilla.redhat.com/query.cgi?format=advanced	Red Hat bug tracker
CVE Mitre	https://cve.mitre.org/cve/search_cve_list.html	CVE archival database
NDV	https://nvd.nist.gov/vuln/full-listing	National Vulnerability Database
CERT/CC VND	https://www.kb.cert.org/vuls/search/	CMU vulnerability notes database
LWN	https://lwn.net/Security/Index/	Linux kernel news site
oss-sec	https://seclists.org/oss-sec/	Open-source security mailing list
fulldisc	https://seclists.org/fulldisclosure/	Full-disclosure mailing list
bugtraq	https://seclists.org/bugtraq/	Bugtraq mailing list
Exploit-DB	https://www.exploit-db.com/search	Exploit database
VED	https://www.rapid7.com/db/	Vulnerability/exploit database
Ubuntu	https://ubuntu.com/security/cve	Ubuntu CVE reports
Gentoo	https://security.gentoo.org/glsa	Gentoo security database
SUSE Bugzilla	https://bugzilla.suse.com/query.cgi?format=advanced	SUSE bug tracker
SUSE CVE	https://www.suse.com/security/cve/	SUSE CVE database
Debian	https://security-tracker.debian.org/tracker/data/json	Debian CVE dataset
CVE details	https://www.cvedetails.com/browse-by-date.php	Public vulnerability datasource
Other vendors	https://oss-security.openwall.org/wiki/vendors	Security vendor list

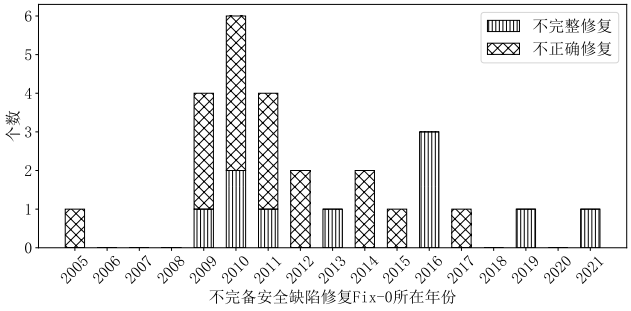

Figure 1: Annual trend of incomplete security bug fixes

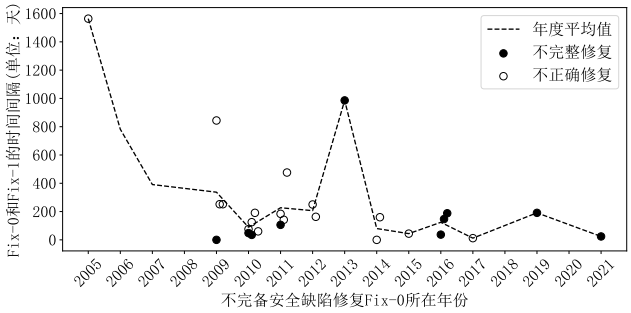

Figure 2: Time interval between discovery and initial fix

degree of stealthiness of incomplete fixes (RQ1.2). Next, we analyze the relationship between incomplete fixes and vulnerability types to inform resource allocation for different categories of security flaws (RQ1.3). Finally, we investigate the characteristics of the patch files themselves to lay the groundwork for identifying the root causes of these incomplete fixes (RQ1.4).

- **Research Question 2: What are the root causes of incomplete security bug fixes?** We analyze and summarize the causes of incomplete fixes in chronological order, before the initial fix, during the fixing process, and after the fix is released, to provide guidance for developing more complete security fixes and to inform the design of detection techniques.
- **Research Question 3: How can incomplete security bug fixes be detected?** Finally, we develop a detection tool inspired by the identified root causes and apply it to the Linux kernel to discover additional incomplete security fixes, thereby reducing the likelihood of such fixes in the future.

3 RQ1: What Are the Characteristics of Incomplete Security Bug Fixes?

In this section, we conduct a quantitative analysis of incomplete security bug fixes in the Linux kernel and answer each subquestion of RQ1.

3.1 RQ1.1: Temporal Distribution of Incomplete Security Bug Fixes

As shown in Figure 1, we categorize incomplete security bug fixes in the Linux kernel by year. Based on our definitions and manual inspection, 11 cases fail to repair all the original security defects in the software, and 15 cases fail to properly repair the original security defects, or introduces new ones.

Finding 1: Descriptions of incomplete security fixes commonly use the keywords *incomplete* and *incorrect*. However, manual inspection reveals misuses of the term *incomplete*. For instance, the fix for CVE-2010-4163 is an incorrect fix, but its public description labels it as incomplete. Such inconsistencies can mislead subsequent analysis. Therefore, we publish our curated dataset with corrected descriptions that strictly follow our definitions.

Finding 2: As shown in Figure 1, incomplete security bug fixes appeared as early as 2005. Over time, their annual counts show a general downward trend starting from 2010, though the phenomenon remains non-negligible. A short-term rebound appears around 2016, likely correlated with the spike of 215 and 449 Linux kernel vulnerabilities reported in 2016 and 2017, respectively. Overall, publicly available data indicate that while occasional increases occur, incomplete fixes

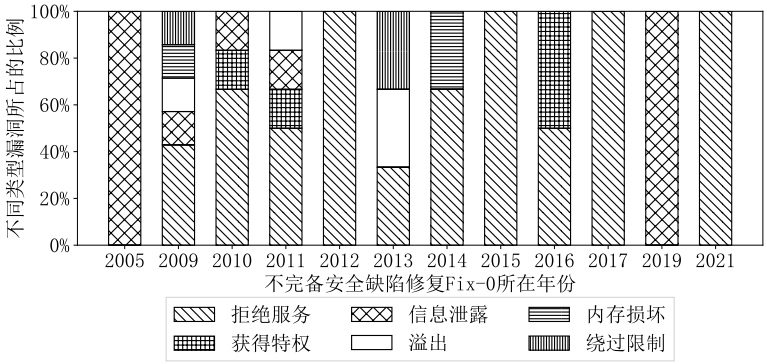

Figure 3: Vulnerability type distribution of Fix-0 in incomplete security bug fixes

Dataset	Modified Files	Added LOC	Deleted LOC	Modified LOC	Patch Size (bytes)
Fix-0	2.32/3.21	32.76/58.23	11.04/24.32	9.50/14.54	3173.68/4452.91
Fix-1	2.08/2.27	26.20/49.52	9.92/21.61	367/3.29	2456.96/3992.47

Table 2: Patch characteristics of incomplete security fixes (mean/std)

show a gradual decline, reflecting more careful security patching practices. Nevertheless, incomplete fixes still appeared in 2021, and this study also discovered new instances (see Section 5). These observations confirm that incomplete security bug fixes remain an important issue.

3.2 RQ1.2: Time Interval Between Initial Fix and Subsequent Correction

Figure 2 shows the time interval (in days) between Fix-0 and Fix-1 for each incomplete security bug fix, showing how long it took for the incomplete fix to be discovered.

Finding 3: Although the first observed case of an incomplete fix (in 2005) took nearly four years (over 1,600 days) to be corrected, Figure 2 shows that the Fix-0 to Fix-1 interval has gradually shortened over time. This suggests incomplete fixes have become easier to uncover and are now detected more promptly, indicating decreasing stealthiness.

Finding 4: Most incomplete security bug fixes are detected within 50–200 days. In some cases, Fix-1 was committed on the same day as Fix-0. However, some fixes remained hidden for years. For example, the spelling error introduced during the fix for CVE-2005-4881 was not discovered until 1,564 days later, leading to an uninitialized data leak (CVE-2009-3612). These cases imply that additional yet-undiscovered incomplete fixes may still exist among historical patches.

3.3 RQ1.3: Relationship Between Incomplete Fixes and Vulnerability Types

As shown in Figure 3, we categorize Fix-0 patches by the vulnerability types defined by CVE Details.

Finding 5: Since 2009, the diversity of vulnerability types associated with incomplete fixes has gradually decreased. According to CVE Details classifications, denial-of-service vulnerabilities dominate, followed by privilege escalation, and information disclosure. Therefore, when fixing these high-frequency categories, additional reviewers, testers, and discussion efforts should be allocated to avoid incomplete fixes.

3.4 RQ1.4: Patch Size and Code Modifications in Incomplete Fixes

Table 2 summarizes code-level characteristics of the collected patches. Commit messages are excluded as they do not reflect actual code modifications. We use `diffstat` to compute added, deleted, and modified lines of code (mean/std).

Finding 6: As shown in Table 2, most patches involve only modest code modifications. Only a few patches modify substantial amounts of code. For example, the fix for CVE-2016-6786 adds 207 lines and deletes 37 lines. Across multiple dimensions, Fix-1 patches tend to be smaller than Fix-0 patches. Overall, incomplete security fixes typically involve patches of limited complexity, facilitating root-cause analysis.

4 RQ2: What Are the Root Causes of Incomplete Security Bug Fixes?

In this section, we analyze the root causes of the 26 incomplete security bug fixes collected from the Linux kernel. We categorize cases into three phases before fixing, during fixing, and after fixing, highlighting representative examples and summarizing common patterns.

4.1 Misled by Symptoms Before Fixing

This category highlights that developers were misled by crash symptoms and failed to identify the underlying root cause.

- **CVE-2015-6937 → CVE-2015-7990:**
  In Figure 4, Fix-0 attempted to resolve a NULL-pointer dereference but did so incorrectly. Fix-1 correctly resolves the issue by introducing appropriate locking.
- **CVE-2017-5986 → CVE-2017-6353:**

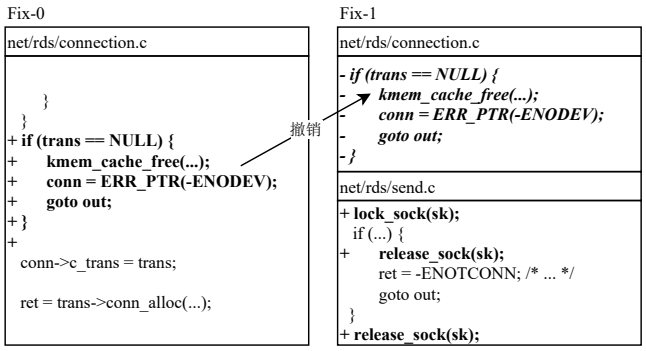

Figure 4: CVE-2015-6927 (left) and CVE-2015-7990 (right)

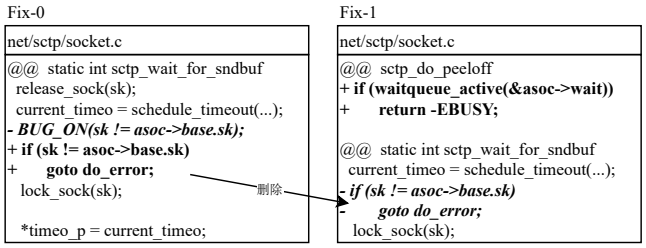

Figure 5: CVE-2017-5986 (left) and CVE-2017-6353 (right)

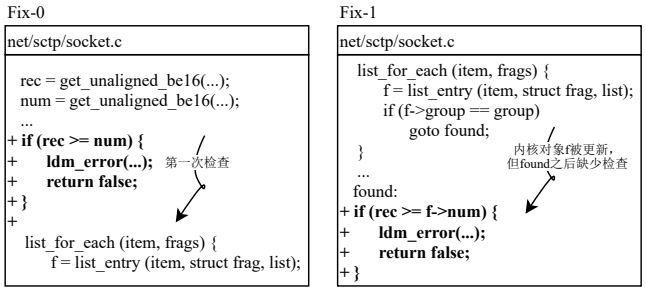

Figure 6: CVE-2011-1017 (left) and CVE-2011-2182 (right)

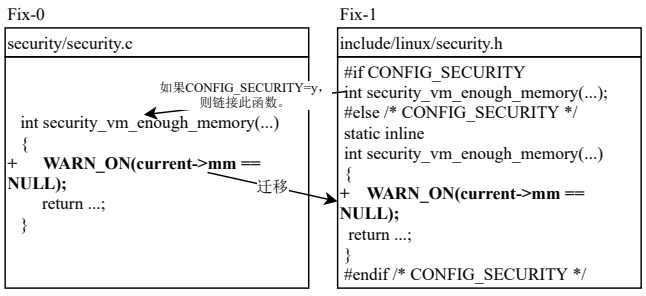

Figure 7: CVE-2010-1643 (left) and CVE-2008-7256 (right)

As shown in Figure 5, Fix-0 removed an assertion improperly, leading to a double-free vulnerability. Fix-1 applies a new fixing strategy that avoids both issues and correctly resolves the flaw.

Takeaway 1: Before applying a fix, developers must thoroughly understand the root cause rather than rely solely on surface-level crash symptoms. Such cases typically require reverting Fix-0 and introducing a new, corrected fix.

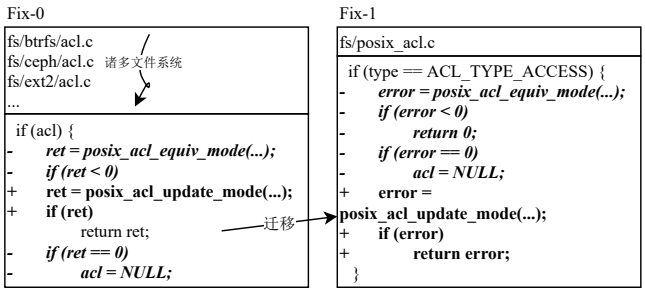

Figure 8: CVE-2016-7097 (left) and CVE-2017-5551 (right)

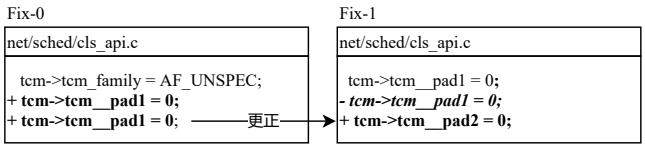

Figure 9: CVE-2005-4881 (left) and CVE-2009-3612 (right)

4.2 Overlooking Similar Issues During Fixing

This category shows that developers correctly identified the root cause but missed analogous issues in similar modules.

- **CVE-2011-1017 → CVE-2011-2182:** As shown in Figure 6, Fix-0 added a check on `rec`, but this single check was insufficient. Fix-1 completes the fix by adding the missing condition.
- **CVE-2010-1643 → CVE-2008-7256:** As shown in Figure 7, Fix-0 patched `security_vm_enough_memory()`, but failed to modify the corresponding function guarded by `CONFIG_SECURITY`. Fix-1 resolves this oversight.
- **CVE-2016-7097 → CVE-2017-5551:** As shown in Figure 8, Fix-0 addressed the issue in multiple filesystems, but omitted the tmpfs variant. Fix-1 completes the fix.
- **CVE-2019-14615 → CVE-2020-8832:** The original vulnerability had been fixed upstream, but the patch was not back-ported to a specific Ubuntu release.

Takeaway 2: When fixing a vulnerability, developers should check whether analogous modules contain similar issues, e.g., different call sites, similar functions, parallel subsystems (filesystems, network protocols, CPU architectures), different branches, or downstream distributions. These cases often require migrating Fix-0 logic to corresponding modules.

4.3 Introducing Semantic Errors After Fixing

This category shows that developers understood the root cause but introduced new semantic errors during the fix.

4.3.1 Variable Level

- **CVE-2005-4881 → CVE-2009-3612:** As shown in Figure 9, Fix-0 mistakenly reinitialized `tem_pad1`, leaving `tem_pad2` uninitialized and causing an information leak. Fix-1 initializes `tem_pad2` properly.

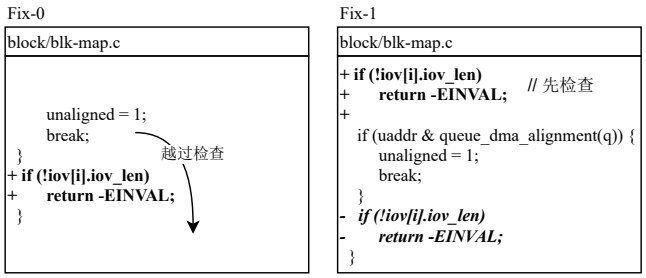

Figure 10: CVE-2010-4163 (left) and CVE-2010-4668 (right)

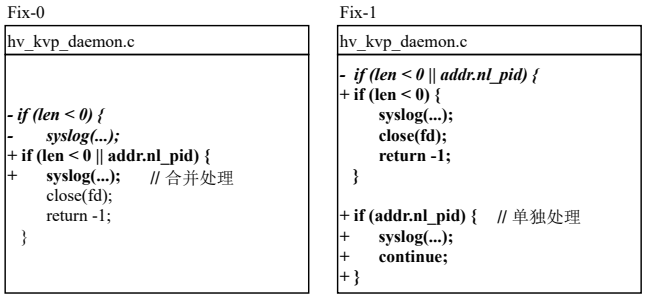

Figure 11: CVE-2012-2669 (left) and CVE-2012-5532 (right)

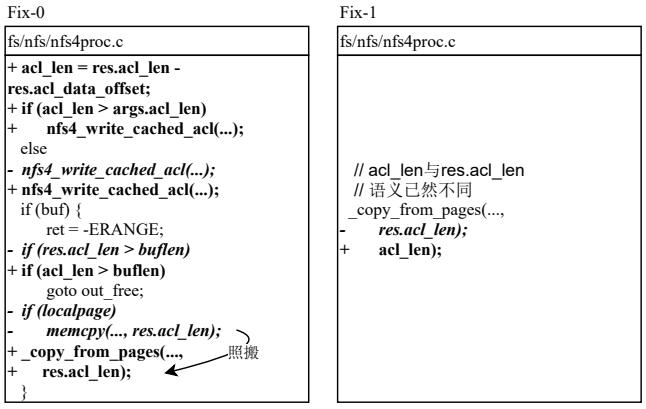

Figure 12: CVE-2011-4131 (left) and CVE-2012-2375 (right)

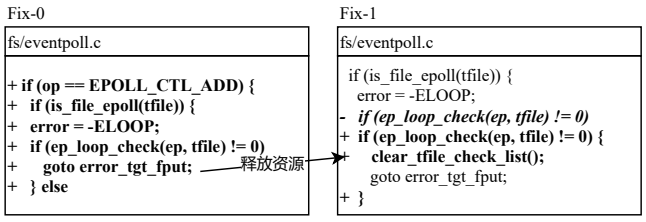

Figure 13: CVE-2011-1083 (left) and CVE-2012-3375 (right)

4.3.2 Function Level

- **CVE-2010-4163 → CVE-2010-4668:** As shown in Figure 10, Fix-0 introduced a check on `iov[i].iov_len`, but placed it incorrectly such that the check was bypassed when the `break` was triggered. Fix-1 moves the check before the break to preserve intended semantics.
- **CVE-2012-2669 → CVE-2012-5532:** In Figure 11, Fix-0 conflated the cases of `addr.nl_pid != 0` and `len < 0`, violating semantic expectations and introducing new issues. Fix-1 handles `addr.nl_pid != 0` separately.
- **CVE-2011-4131 → CVE-2012-2375:** In Figure 12, Fix-0 incorrectly reused the last parameter passed to `memcpy`, despite differing semantics between `res.acl_len` and `acl_len`. Fix-1 corrects this mismatch.
- **CVE-2009-4307 → CVE-2012-2100:** As shown in Figure 14, Fix-0 attempted to validate `groups_per_flex` but left overflow cases unhandled. Fix-1 correctly validates `sbi->s_log_groups_per_flex`.

4.3.3 Resource Level

- **CVE-2011-1083 → CVE-2012-3375:** As shown in Figure 13, Fix-0 introduced new data structures but failed to release all allocated resources, leading to a DoS vulnerability. Fix-1 releases the missing resources.
- **CVE-2013-4312 → CVE-2016-2550:** Fix-0 failed to maintain the reference count for open files, enabling an attacker to exhaust memory by repeatedly opening files.
- **CVE-2010-4250 → CVE-2011-1479:** Fix-0 was too simplistic and introduced a double-free vulnerability.

4.3.4 Access-Control Level

- **CVE-2010-4347 → CVE-2011-1021:** Fix-0 reduced excessive write permissions of `custom_method` in debugfs, but mistakenly preserved write access for the root user. Fix-1 restricts access by enabling the file only in debugging mode. Similarly, when addressing CVE-2016-9576, the fix failed to consider the case where the KERNEL_DS segment selector is active. Under this condition, an attacker could exploit sg_write to gain arbitrary read/write primitives or trigger a denial-of-service attack (CVE-2016-10088).

Takeaway 3: After fixing a vulnerability, developers, testers, and reviewers must verify that the fix preserves intended program semantics. Such cases typically require analyzing at which granularity Fix-0 violated the original semantics and applying a targeted correction.

5 RQ3: Detecting Incomplete Security Bug Fixing due to "Missing Similar Components"

In Section 4, we analyzed the phenomenon of incomplete security bug fixing in the Linux kernel and summarized common mistakes across the three stages of fixing (before, during, and after). Based on the insights drawn from the category of "missing similar components during fixing", this section presents a detection algorithm based on *fix-pattern fingerprints* to automatically identify other potential incomplete security bug fixes in the Linux kernel. This method helps reveal hidden incomplete fixes and can reduce the likelihood of such issues during patch submission.

Fix-0

fs/ext4/super.c

```
- if (!sbi->s_es->s_log_groups_per_flex) {
+ sbi->s_log_groups_per_flex = sbi->s_es->s_log_groups_per_flex;
+ groups_per_flex = 1 << sbi->s_log_groups_per_flex;
+
+ if (groups_per_flex < 2) {
        sbi->s_log_groups_per_flex = 0;
        return 1;
   }
```

Fix-1

fs/ext4/super.c

```
 sbi->s_log_groups_per_flex = sbi->s_es->s_log_groups_per_flex;
- groups_per_flex = 1 << sbi->s_log_groups_per_flex;
-
- if (groups_per_flex < 2) {
+ if (sbi->s_log_groups_per_flex < 1 || sbi->s_log_groups_per_flex > 31) {
        sbi->s_log_groups_per_flex = 0;
        return 1;
  }
+ groups_per_flex = 1 << sbi->s_log_groups_per_flex;
```

Figure 14: CVE-2009-4307 (left) and CVE-2012-2100 (right)

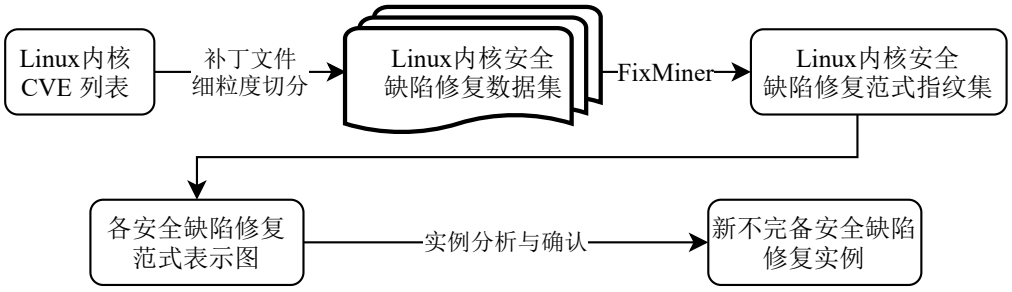

Figure 15: Workflow of the fix-pattern–fingerprinting detection algorithm

```
151f52f
@@ -205,6 +205,8 @@
int blk_rq_map_user_iov(...
                unaligned = 1;
                break;
        }
+       if (!iov[i].iov_len)
+               return -EINVAL;
    }

    if (...)
```

```
UPD for
--- UPD block
------- INS condition TO if              # if -> if (!iov[i].iov_len)
---------- INS expr TO condition         #    {} -> !iov[i].iov_len
--------------- INS operator TO expr      #       {} -> !
--------------- INS name TO expr          #       ! -> !iov[i]
------------------- INS name TO name      #         {} -> iov
------------------- INS index TO name     #         iov -> iov[i]
---------------------- INS expr TO index   #            {} -> []
---------------------- INS name TO expr    #            [i] -> [i]
--------------- INS operator TO expr      #       !iov[i] -> !iov[i].
--------------- INS name TO expr          #       !iov[i]. -> !iov[i].iov_len
------- INS block TO if                   # if () -> if (){ return -EINVAL; }
---------- INS expr TO return            #    return -> return -EINVAL;
--------------- INS operator TO expr      #       {} -> -
--------------- INS name TO expr          #       - -> -EINVAL
```

Figure 16: Example of a fix pattern: the patch (left) and the corresponding FixMiner-generated fix pattern (right)

5.1 Fix-Pattern–Fingerprint Detection

Incomplete security bug fixes of the "missing similar components" type can be characterized by the following rule: Within a set of similar components, the Fix-0 patch updates only a subset of them while omitting others. Here, a component is an abstract concept that may refer to a group of similar functions or a group of modules implementing comparable functionality. In the patch file submitted for a security bug fix, if identical or similar fix patterns are applied to multiple similar components, then the complete set of such components can be further examined to check whether any have been omitted. Following Fig. 15, we applied FixMiner [5] to extract and analyze fix patterns from 1,615 CVE fixes in the Linux kernel and ultimately uncovered two new problematic cases.

We collected 1,615 Linux kernel CVEs using the Linux Kernel CVE database [3], each of which includes commit hashes referencing its fixes. We then constructed a dataset for fix-pattern analysis. For each CVE we retrieved all associated patch files, as well as snapshots of the patched file before and after the patch. If a single CVE patch modified multiple files, we split it into several fine-grained patch files, each corresponding to one modified source file, resulting in 2,753 fine-grained patches, facilitating detailed analysis.

FixMiner was then applied to this dataset to extract fix-pattern fingerprints. FixMiner, built on top of GumTreeDiff [10], automatically extracts abstract syntax tree (AST)–based fix patterns by abstracting the code changes within patches. Figure 16 shows an example patch and its corresponding fix pattern. After configuration and processing, FixMiner generated fix-pattern fingerprints for all 2,753 fine-grained patches. It also computed similarities among fix patterns and linked them accordingly. Ultimately, FixMiner

```
UPD function
--- UPD block
------- DEL expr_stmt
---------- DEL expr
------------ DEL name
------------ DEL operator
------------ DEL name
```

```
// Fix Pattern Infered from (
// linux_2079a5_f57eda_drivers#video#fbdev#intelfb#intelfbhw.c.txt_0,
// linux_1790c22_f009627_drivers#atm#idt77252.c.txt_0,
// linux_f57eda_fbaa19_drivers#video#fbdev#riva#riva_hw.c.txt_0,
// linux_e54560_2788052_drivers#media#usb#cx231xx#cx231xx-video.c.txt_0,
// linux_61241d_22b4dc5_drivers#staging#westbridge#astoria#api#src#cyasusb.c.txt_0
//  )
```

Figure 17: Example fix-pattern fingerprint shared among multiple patches

```
arch/arm/mm/mmap.c
arch/mips/mm/mmap.c
arch/powerpc/mm/hugetlbpage-radix.c
arch/x86/kernel/sys_x86_64.c
arch/x86/mm/hugetlbpage.c
...
```

```
@@ -65,7 +65,7 @@ arch_get_unmapped_area(..., unsigned long addr,
        vma = find_vma(mm, addr);
        if (TASK_SIZE - len >= addr &&
-           (!vma || addr + len <= vma->vm_start))
+           (!vma || addr + len <= vm_start_gap(vma)))
            return addr;
    }
```

```
--- UPD if
------ UPD condition
--------- UPD expr
----------- INS call TO expr
------------ INS name TO call
------------ INS argument_list TO call
------------------ INS argument TO argument_list
---------------------- INS expr TO argument
------------------------ INS name TO expr
----------- DEL name
------------ DEL name
------------ DEL operator
------------ DEL name
```

Figure 18: Fix pattern of CVE-2017-1000365

identified 311 distinct fix-pattern fingerprints related to security bug fixes. Figure 17 shows an example fingerprint shared across multiple patches.

Using these fingerprints, we constructed a fix-pattern graph for each CVE. The root node represents the CVE ID; child nodes correspond to fix-pattern fingerprints that are detected in Fix-0 across its modified files. A large number of repeated nodes within a graph indicates that the Fix-0 patch likely involved modifications across many similar components. We then conducted manual validation for these candidates.

5.2 New Findings

Finding 7: From scanning the patches of 1615 CVEs, we identified two incomplete security bug fixes. One has already been confirmed by the Linux kernel community and is in the process of being merged into the mainline; the other was a hidden incomplete fix that had been silently corrected in a later patch without receiving a CVE ID.

- **CVE-2017-1000364:** CVE-2017-1000364 was submitted in June 2017. Its fix pattern, illustrated in Fig. 18, was applied to multiple architectures—Arc, ARM, FRV, MIPS, Parisc, PowerPC, s390, SH, Sparc, Tile, x86, and Xtensa. During manual analysis, we found that the fix pattern was *not* applied to the NDS32 architecture. Support for NDS32 was added to the Linux kernel in 2018. The newly added module reused existing code to implement standard kernel features such as memory mapping. However, the developers failed to port the CVE-2017-1000364 fix to NDS32 when integrating the new architecture. As a result, the vulnerability described in the CVE persisted on NDS32, even though all other architectures were properly patched.

```
struct dvb_diseqc_master_cmd {
    __u8 msg[6];
    __u8 msg_len;
};
```

Figure 19: Definition of DiSEqC messages

```
 /* Validate length */
-  if (d->msg_len > 15)
+ if (d->msg_len > sizeof(d->msg))
   return -EINVAL;
```

Figure 20: Fix pattern of CVE-2015-9289

- **CVE-2015-9289:** This CVE addressed a buffer overflow in multimedia DVB drivers in the Linux kernel. The DiSEqC message structure is shown in Fig. 19. The field **msg** holds the message data, and **msg_len** indicates its length. A valid DiSEqC message ranges from 3 to 6 bytes (3-byte header plus up to 3 bytes of optional param-

Table 3: Comparison of Studies on Root Cause Analysis of Bug Fixing

Study	Dataset Source (System Software)	Focus	Detection Tool
BuJihun Park et al. [17]	Eclipse and Mozilla (No)	Incomplete General Bug Fixes	✗
Le An et al. [4]	Eclipse and Mozilla, etc. (No)	Incomplete and Re-opened General Bug Fixes	✓
Qing Mi et al. [13]	Eclipse (No)	Re-opened General Bug Fixes	✗
Zuoning Yin et al. [25]	Linux, FreeBSD kernel, etc. (Yes)	Incomplete General Bug Fixes	✗
Zhen Ni et al. [16]	Radare2 and Mozilla (No)	General Bug Fixes	✗
Kai Zhang et al. [28]	Mozilla (No)	Incomplete Security Bug Fixes	✗
This Work	**Linux Kernel (Yes)**	**Incomplete Security Bug Fixes**	✓

drivers/media/usb/dvb-usb/vp702x-fe.c

```
static int vp702x_fe_send_diseqc_msg (..., struct dvb_diseqc_master_cmd *m) {
    // ...
    if (m->msg_len > 4)
        #当meg_len的值为5或6这两个合法值时，将无法通过验证
        return -EINVAL;
    // ...
```

Figure 21: Incorrect checking of DiSEqC message length

eters). The CVE patch corrected improper upper-bound checks on **msg_len** in several DVB drivers (Fig. 20). However, our analysis found that several DVB drivers in the same subsystem still contained incorrect checks (Fig. 21). Some drivers (e.g., `tda8083.c`, `stv0288.c`, `stv0299.c`, `cx24123.c`, `ds3000.c`) lacked *any* length check. Thus, the patch for CVE-2015-9289 was incomplete. Interestingly, maintainers later fixed these missing checks through a normal bug-fix patch without applying for a new CVE ID. This hidden incomplete fix expands our dataset with another overlooked instance.

6 Related Work

6.1 Incomplete Security Bug Fixes

Bug fixing is a common yet essential activity throughout the software development lifecycle. Over the past decade, numerous studies have examined the characteristics of general bug fixes [4,6,17,19,22,24,26,29] and security bug fixes [9,11,14, 15, 18, 20, 23], with the goals of improving bug management, refining fixing practices, and facilitating automated bug-fixing approaches. Most studies [6, 9, 11, 14, 18, 20, 23, 24, 26, 29] focus on answering a set of fundamental questions: the general workflow of bug fixing; whether fixes exhibit spatial locality or repetition at the levels of lines, functions, and files; the extent to which fixes rely on API semantics; required techniques, human effort, and time for locating, diagnosing, and fixing bugs; differences between general and security bug fixes; and the potential for automated bug fixing. Others explore the number of patches involved in bug fixing and its implications.

Bug fixes can be broadly divided into two categories depending on the number of patches involved. The first category involves a single patch; the second involves multiple patches and is referred to as an *incomplete fix*. Generally, incomplete fixes fall into two subtypes: *incomplete fixes* and *incorrect fixes*. An incomplete fix occurs when the initial patch does not repair all code locations affected by the flaw. An incorrect fix refers to a case where the initial patch is flawed—failing to remediate the original issue or introducing new defects. Meanwhile, several studies have highlighted significant differences between the fixing processes of general bugs and security bugs, advocating for treating the two categories separately [7, 9, 27]. Existing studies of incomplete fixes over the past decade are limited in number and mostly focus on non-system software such as Eclipse and Mozilla [4, 13, 17, 28]. Notably, Yin et al. [25] examined incomplete fixes in the Linux kernel in 2011, but their study analyzed both general and security bug fixes jointly, emphasizing the importance of understanding incomplete fixes, summarizing fixing patterns and common mistakes, and providing general recommendations. However, their dataset lacks the past ten years of security bug fixes and is thus outdated.

As shown in Table 3, our work differs by focusing specifically on incomplete *security* bug fixes in the Linux kernel, including cases from the last decade, providing novel insights. Moreover, building upon these findings, we develop a detection tool targeting a prevalent class of incomplete security bug fixes, thereby improving the correctness of security patches and advancing the state of the art.

6.2 Root Cause Analysis of Software Bugs

Several studies over the past decade have investigated the root causes of software bugs [4, 13, 16, 17, 25, 28]. Park et al. [17] analyzed incomplete general bug fixes in Eclipse JDT Core, Eclipse SWT, and Mozilla, identifying diverse causes such as forgotten code porting, incorrect handling of conditional branches, and incomplete refactoring. They emphasized that similarity-based code detection is insufficient to prevent all incomplete fixes. They also found that in 14%–15% of cases, subsequent patches differed substantially from the initial patch both in location and structural dependency, implying that automated patch recommendation systems need redesign. Le An et al. [4] studied the relationship between incomplete bug fixes and re-opened bugs across Mozilla, NetBeans, Eclipse JDT Core, Eclipse Platform SWT, and WebKit. They found partial overlap between the two phenomena and further developed classifiers based on human factors, bug reports, and simple fixing statistics to distinguish them. Zhang et al. [28] examined incomplete security bug fixes in Mozilla and found that such fixes correlate with the complexity of root

causes and complexity of the fix itself; they concluded that more elaborate and effective fixing practices can reduce the likelihood of incomplete security fixes. Our work, in contrast, focuses on incomplete security fixes specifically within the Linux kernel, a large-scale system software project.

6.3 Similarity-Based Bug Detection

Leveraging known fix patterns [5,8,12,21], various similarity-based techniques have been proposed over the past decade to detect software bugs in open-source projects [30]. Zhong et al. [30] used the Grapa tool to analyze code changes before and after bug fixes, extracting bug features from patches and building a classification model. They conducted the first large-scale evaluation on 6,048 bug fixes from four popular Apache projects, confirming three new defects, all of which were subsequently acknowledged and fixed. Our work adopts a similar philosophy but differs in purpose and application: we use fix-pattern fingerprints extracted from incomplete *security* bug fixes to detect additional incomplete security bug fixes in the Linux kernel. By constructing a dataset of 1,615 CVE-related Linux kernel security fixes and extracting fix-pattern fingerprints, we successfully identified one previously unknown and unfixed incomplete security fix, as well as one hidden but already-fixed case.

7 Limitations and Future Directions

This paper presents the characteristics, root causes, and detection techniques of incomplete security bug fixes in the Linux kernel. However, several limitations remain. (1) In analyzing the characteristics and root causes of incomplete security bug fixes in the Linux kernel, we relied primarily on manual analysis due to the relatively small dataset. Manual analysis is inefficient when scaling to all incomplete security bug fixes across large-scale open-source software ecosystems. In the future, improvements can be made from two aspects of static program analysis on patches. First, during our study, we found that patch files and their commit messages often diverge. An automated technique that summarizes patch semantics and contrasts them with the accompanying descriptions would greatly aid in understanding, testing, and auditing security fixes. Second, although Fix-1 patches reflect the presence of incomplete security bug fixes, it is often difficult to directly identify their underlying causes without further analysis. If the manual reasoning process can be formalized and automated, it would significantly accelerate the entire workflow. (2) In detecting incomplete security bug fixes in the Linux kernel, this paper focuses only on the "missing similar components" category. Other categories lack clear patterns or exhibit high complexity, making them more challenging to detect. Future research may proceed in two directions. First, by studying incomplete security bug fixes in a wider range of open-source projects, more root causes may be identified, enabling the design of new detection algorithms. Second, for specific classes of vulnerabilities, e.g., data races, highly specialized detection algorithms may be developed to improve coverage.

8 Conclusion

This paper presents the first dataset of incomplete security bug fixes in the Linux kernel, and provides a comprehensive analysis of their characteristics, root causes, and detection techniques. Finally, by applying our fix-pattern–based fingerprinting algorithm to 1,615 Linux kernel vulnerabilities, we discovered one previously unnoticed incomplete security fix and one newly identified incomplete security fix. These findings demonstrate the potential of automated techniques to aid in the detection of incomplete security bug fixes and thereby contribute to enhancing the security of the Linux kernel.

References

[1] Linux cve details. https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33, 2011.

[2] Vulnerabilities by type. https://www.cvedetails.com/vulnerabilities-by-types.php, 2011.

[3] https://www.linuxkernelcves.com/, 2021.

[4] Le An, Foutse Khomh, and Bram Adams. Supplementary bug fixes vs. re-opened bugs. In *IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 205–214, 2014.

[5] Koyuncu Anil, Kui Liu, Tegawendé F Bissyandé, Dongsun Kim, Jacques Klein, Monperrus Martin, et al. Fixminer: Mining relevant fix patterns for automated program repair. *Empirical Software Engineering*, pages 1980–2024, 2020.

[6] Marcel Böhme, Ezekiel O Soremekun, Sudipta Chattopadhyay, Emamurho Ugherughe, and Andreas Zeller. Where is the bug and how is it fixed? an experiment with practitioners. In *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, pages 117–128, 2017.

[7] Felivel Camilo, Andrew Meneely, and Meiyappan Nagappan. Do bugs foreshadow vulnerabilities? a study of the chromium project. In *International Conference on Mining Software Repositories (MSR)*, pages 269–279, 2015.

[8] Eduardo Cunha Campos and Marcelo de Almeida Maia. Common bug-fix patterns: A large-scale observational study. In *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 404–413, 2017.

[9] Gerardo Canfora, Andrea Di Sorbo, Sara Forootani, Antonio Pirozzi, and Corrado Aaron Visaggio. Investigating the vulnerability fixing process in oss projects:

Peculiarities and challenges. *Computers & Security*, page 102067, 2020.

[10] Jean-Rémy Falleri, Floréal Morandat, Xavier Blanc, Matias Martinez, and Martin Monperrus. Fine-grained and accurate source code differencing. In *ACM/IEEE International Conference on Automated Software Engineering, ASE '14, Vasteras, Sweden - September 15 - 19, 2014*, pages 313–324, 2014.

[11] Frank Li and Vern Paxson. A large-scale empirical study of security patches. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 2201–2215, 2017.

[12] Kui Liu, Dongsun Kim, Tegawendé F Bissyandé, Shin Yoo, and Yves Le Traon. Mining fix patterns for findbugs violations. *IEEE Transactions on Software Engineering (TSE)*, pages 165–188, 2018.

[13] Qing Mi and Jacky Keung. An empirical analysis of reopened bugs based on open source projects. In *International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 1–10, 2016.

[14] Patrick J Morrison, Rahul Pandita, Xusheng Xiao, Ram Chillarege, and Laurie Williams. Are vulnerabilities discovered and resolved like other defects? *Empirical Software Engineering*, pages 1383–1421, 2018.

[15] Dongliang Mu, Alejandro Cuevas, Limin Yang, Hang Hu, Xinyu Xing, Bing Mao, and Gang Wang. Understanding the reproducibility of crowd-reported security vulnerabilities. In *USENIX Security Symposium (Security)*, pages 919–936, 2018.

[16] Zhen Ni, Bin Li, Xiaobing Sun, Tianhao Chen, Ben Tang, and Xinchen Shi. Analyzing bug fix for automatic bug cause classification. *Journal of Systems and Software*, page 110538, 2020.

[17] Jihun Park, Miryung Kim, Baishakhi Ray, and Doo-Hwan Bae. An empirical study of supplementary bug fixes. In *International Conference on Mining Software Repositories (MSR)*, pages 40–49, 2012.

[18] Valentina Piantadosi, Simone Scalabrino, and Rocco Oliveto. Fixing of security vulnerabilities in open source projects: A case study of apache http server and apache tomcat. In *IEEE Conference on Software Testing, Validation and Verification (ICST)*, pages 68–78, 2019.

[19] Renaud Rwemalika, Marinos Kintis, Mike Papadakis, Yves Le Traon, and Pierre Lorrach. An industrial study on the differences between pre-release and post-release bugs. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 92–102, 2019.

[20] Muhammad Shahzad, Muhammad Zubair Shafiq, and Alex X Liu. A large scale exploratory analysis of software vulnerability life cycles. In *International Conference on Software Engineering (ICSE)*, pages 771–781, 2012.

[21] Mauricio Soto, Ferdian Thung, Chu-Pan Wong, Claire Le Goues, and David Lo. A deeper look into bug fixes: patterns, replacements, deletions, and additions. In *International Conference on Mining Software Repositories (MSR)*, pages 512–515, 2016.

[22] Peipei Wang, Chris Brown, Jamie A Jennings, and Kathryn T Stolee. An empirical study on regular expression bugs. In *International Conference on Mining Software Repositories (MSR)*, pages 103–113, 2020.

[23] Song Wang and Nachi Nagappan. Characterizing and understanding software developer networks in security development. *arXiv preprint arXiv:1907.12141*, 2019.

[24] Ye Wang, Na Meng, and Hao Zhong. An empirical study of multi-entity changes in real bug fixes. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 287–298, 2018.

[25] Zuoning Yin, Ding Yuan, Yuanyuan Zhou, Shankar Pasupathy, and Lakshmi Bairavasundaram. How do fixes become bugs? In *ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ECSE/FSE)*, pages 26–36, 2011.

[26] Ruru Yue, Na Meng, and Qianxiang Wang. A characterization study of repeated bug fixes. In *IEEE international conference on software maintenance and evolution (ICSME)*, pages 422–432, 2017.

[27] Shahed Zaman, Bram Adams, and Ahmed E Hassan. Security versus performance bugs: a case study on firefox. In *International Conference on Mining Software Repositories (MSR)*, pages 93–102, 2011.

[28] Kai Zhang, Xiaobing Sun, Xin Peng, and Wenyun Zhao. A study on re-rolling security bug fixes based on mozilla. *Computer Science*, pages 41–49, 2017.

[29] Hao Zhong and Zhendong Su. An empirical study on real bug fixes. In *IEEE International Conference on Software Engineering (ICSE)*, pages 913–923, 2015.

[30] Hao Zhong, Xiaoyin Wang, and Hong Mei. Inferring bug signatures to detect real bugs. *IEEE Transactions on Software Engineering (TSE)*, 2020.

[31] Yaqin Zhou, Jing Kai Siow, Chenyu Wang, Shangqing Liu, and Yang Liu. SPI: automated identification of security patches via commits. *arXiv*, 2021.