Received: 31 December 2020 | Revised: 8 July 2021 | Accepted: 4 August 2021

DOI: 10.1002/smr.2376

RESEARCH ARTICLE - EMPIRICAL

Journal of Software: Evolution and Process WILEY

A comprehensive study on security bug characteristics

Ying Wei[1] | Xiaobing Sun[1,2] | Lili Bo[1,2,3] | Sicong Cao[1] | Xin Xia[4] | Bin Li[1]

[1]School of Information Engineering, Yangzhou University, Yangzhou, China

[2]State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China

[3]Key Laboratory of Safety-Critical Software Ministry of Industry and Information Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China

[4]Faculty of Information Technology, Monash University, Melbourne, Australia

Correspondence
Xiaobing Sun and Bin Li, School of Information Engineering, Yangzhou University, Yangzhou, China.
Email: xbsun@yzu.edu.cn; lb@yzu.edu.cn

Funding information
Jiangsu "333" Project; Key Laboratory of Safety-Critical Software Ministry of Industry and Information Technology, Grant/Award Number: No. NJ2020022; National Natural Science Foundation of China, Grant/Award Numbers: No. 61872312, No. 61972335, No. 62002309; Natural Science Research Project of Universities in Jiangsu Province, Grant/Award Number: No. 20KJB520024; Open Funds of State Key Laboratory for Novel Software Technology of Nanjing University, Grant/Award Numbers: No. KFKT2020B15, No. KFKT2020B16; Six Talent Peaks Project in Jiangsu Province, Grant/Award Number: No. RJFW-053; Yangzhou city-Yangzhou University Science and Technology Cooperation Fund Project, Grant/Award Number: No. YZU201902; Yangzhou University Top-level Talents Support Program, Grant/Award Number: 2019

Abstract

Security bugs can catastrophically impact our increasingly digital lives. Designing effective tools for detecting and fixing software security bugs requires a deep understanding of security bug characteristics. In this paper, we conducted a comprehensive study on security bugs and proposed the classification criteria for security bug category, that is, root cause, consequence, and location. In addition, we selected 1076 bug reports from five projects (i.e., Apache Tomcat, Apache HTTP Server, Mozilla Firefox, Linux Kernel, and Eclipse) in the NVD for investigation. Finally, we investigated the correlation between the classification results and obtained some findings: (1) memory operation is the most common security bug; (2) the primary root causes of security bugs are CON (Configuration Error), INP (Input Validation Error), and MEM (Memory Error); (3) the severity of more than 40% of security bugs is high; (4) security bugs caused by INP mainly occur on web; and (5) security bugs caused by LOG (Logic Resource Error) usually lead to DoS (Denial of Service). We discussed these findings through data analysis, which can also help developers better understand the characteristics of security bugs.

KEYWORDS
bug characteristics, empirical study, security bugs

1 | INTRODUCTION

With the emergence of software applications, personal privacy and information security of companies have attracted more and more public attention. Security bugs are vulnerabilities in software, protocol implementations, or system security policies that allow an attacker to gain unauthorized access or compromise the systems. Despite lots of academic and industrial efforts have been devoted to improving software quality, security bugs are fundamental reason for the prevalence of attacks in software, which contribute to massive disclosure of user data and leakage of private information. For example, in January 2018, there was an incident on the Internet that exploited the remote command execution vulnerability of the WebLogic-WLS component for mining (CVE-2017-10271[1]). This 1-day vulnerability caused a large number of security incidents in a

[1]https://nvd.nist.gov/vuln/detail/CVE-2017-10271

short period of time and affected many fields such as finance, health, and education. It can be seen that preventing security incidents is a process of racing against attackers. Therefore, it is important to fix security bugs as soon as possible. Security bug fixing requires developers to fully understand the characteristics of the security bugs (such as the type of security bugs, the cause, and severity of security bugs). Our study aims to perform a comprehensive empirical study on the security bugs to understand the characteristics of security bugs, which can provide developers with useful insights and guidance to design effective tools to detect and fix security bugs.

A software security bug is code oriented and caused by intentional or unintentional negligence of the operating entity or limitation of the programming language during the software life cycle. They exist in different forms in the software system of all levels. Once it is exploited by the malicious subject, such as gaining higher level rights and disclosing the user privacy data in the software, it will cause security damage to the software system and affect the normal operation of the service built on the software system.

In our study, we analyze 1,076 security bugs between 2015 and 2019 from five projects, that is, Apache Tomcat,[2] Apache HTTP Server,[3] Mozilla Firefox,[4] Linux Kernel,[5] and Eclipse.[6] Based on examining security bugs in these projects, we focus on five research questions to study the characteristics of security bugs and calculated Cohen's Kappa coefficient[1] to verify the consistency of the markings. The findings obtained are as follows:

- **RQ1: (Category) What categories of security bugs occur commonly?**
  We find that the security bugs of resource management, security configuration, memory operation, and authentication restriction are more common than others in the five projects.
- **RQ2: (Root cause) What are the most frequent root causes of security bugs?**
  We find that INP (Input Validation Error), CON(Configuration Error), and MEM (Memory Error) are the primary root causes of security bugs. Moreover, through our study on the relation between categories and root causes of security bugs, we find that MEM often lead to memory operation and resource management security bugs.
- **RQ3: (Severity) How serious are security bugs in general?**
  We find that the severity of over 40% security bugs is high. In addition, memory operation and resource management security bugs account for the most of serious security bugs.
- **RQ4: (Consequence) What are the most frequent consequences of security bugs?**
  We find that the primary consequences caused by security bugs are data leakage and DoS (Denial of Service). Data leakage is the main consequence of authentication restriction and data encryption security bugs. Besides, security bugs caused by LOG (Logic Resource Error) often lead to DoS.
- **RQ5: (Location) What are the locations security bugs usually occur?**
  We find that most security bugs occur on the web and memory. By studying the relation between root causes and locations, we can obtain that lots of security bugs caused by INP occur on web.

In addition to the five questions above, we also examined the relationships between various characteristics of security bugs. Our contributions are as follows:

1. From the perspective of the characteristics of security bugs, we put forward four classification standards: category, root cause, consequence, and location.
2. We conducted a comprehensive empirical study on 1,076 security bugs in five open source projects and classified the characteristics of security vulnerabilities (category, root cause, consequence, and location).
3. We investigated security bug characteristics and the relationships between characteristics from different perspectives and obtained some findings; for example, data leakage is the main consequence of authentication restriction and data encryption security bugs.

The rest of the paper is organized as follows. In Section 2, we describe how we collect and analyze the data for our study, providing classification methodology for the categories, root causes, severity, consequences, and locations of security bugs. From Sections 3 to 7, we answer five research questions. Then, we discuss the practical value of our study in Section 8, followed by the threats to validity in Section 9. Finally, we discuss the related work in Section 10 and conclude this paper in Section 11.

[2]https://tomcat.apache.org/security-9.html#Apache_Tomcat_9.x_vulnerabilities
[3]https://httpd.apache.org/security/vulnerabilities_24.html
[4]https://www.mozilla.org/en-US/security/known-vulnerabilities/firefox/
[5]https://usn.ubuntu.com/
[6]https://www.eclipse.org/

TABLE 1 Empirical data in our study

Project	Apache Tomcat	Apache HTTP Server	Mozilla Firefox	Linux Kernel	Eclipse
Number	36	35	535	409	61
Version	v9.x	v2.4	v43-v67	—	—

2 | METHODOLOGY

2.1 | Data collection

Considering that the diversity of projects can improve the comprehensiveness and accuracy of data analysis, our study is performed based on fixed security bugs from five different projects, that is, Apache Tomcat, Apache HTTP Server, Mozilla Firefox, Linux Kernel, and Eclipse. Apache Tomcat and Apache HTTP Server were chosen because they are the two projects with the highest number of reported software security bugs in the Apache ecosystem.[2] Mozilla Firefox is a free and open source web browser project developed by Mozilla. Linux kernel is the most popular free computer operation system kernel project, and Eclipse is a well-known cross-platform open source integrated development environment. Moreover, they have stable maintenance and complete records on the corresponding official website for viewing, so we chose the security bug reports of these five projects as the research dataset. We also use two sources of data in our study: entries of five projects in NVD[7] and CVE.[8] Though the data in CVE and NVD are similar, they focus on different items. In our study, we combine both databases for exploration. CVE is a list of entries for publicly known security bugs. It is not enough to analyze security bugs only by means of description information in CVE. Hence, we also use NVD which includes more information and detailed analysis for security bug reports. The relation between them is that the CVE list supplies NVD and NVD builds upon the information included in CVE entries to provide enhanced information for each entry such as fixing information, severity scores, and impact ratings. Each security bug has a CVE entry, and each entry has a unique ID, a description, and a severity estimation of the security bug.[2] That is, each security bug report we study corresponds to a CVE ID. To obtain the CVE IDs of the bug reports that have been fixed in all five projects, we referenced three project sites (Apache Tomcat, Apache HTTP Server, and Mozilla Firefox), which summarize the bugs that have been fixed according to the updated version. For the Linux kernel project, we searched for the corresponding CVE ID according to the version update record on the Ubuntu website. For the case that Eclipse contains multiple projects, we search for Eclipse keywords on the CVE website to obtain the corresponding CVE ID. In our study, in order to obtain a more comprehensive and useful security bug report, we adopted the following two steps.

Step 1. We select security bugs between the year of 2015 and 2019 from these five projects. In addition, we remove the security bugs reported after 2019 but have not been fixed or verified. After this step, we identify 1,334 security bugs.

Step 2. As NVD of each CVE entry provides many useful information, we removed the security bug reports without NVD links. In addition, NVD supports Common Vulnerability Scoring System (CVSS) v3.0 standards. CVSS captures the principal characteristics of a security bug and has the numerical score reflecting its severity. The numerical score can be translated into a qualitative representation (such as low, medium, high, and critical) to help organizations accurately assess and prioritize their security bug management process. However, some indicators in CVSS v2.0 are uncertain and complex, which may lead to difficulties in quantitative analysis. At the same time, CVSS v3.0 adds user interaction and scope metrics, which can more comprehensively measure some inherent characteristics of security bugs. Therefore, we also remove the security bug reports without CVSS v3.0 score. Finally, a total of 1,076 security bug reports are obtained.

Table 1 shows the distribution of the data, including the number of security bugs selected from five projects and the corresponding project version. From Table 1, we can see that 36, 35, 535, and 40,961 security bugs from Apache Tomcat, Apache HTTP Server, Mozilla Firefox, Linux Kernel, and Eclipse are used, respectively. The versions of Apache Tomcat, Apache HTTP Server, and Mozilla Firefox are v9.x, v2.4, and v43-v67. Because the version of some security bugs in the Linux kernel is extensive and the specific version cannot be specified, the version number is not provided. For example, in CVE-2018-1120,[9] the security bug was found affecting the Linux kernel before version 4.17. Therefore, its version number is not given. In addition, Eclipse is an open source, Java-based extensible development platform, so there are multiple projects in Eclipse, each with a separate version number. For example, CVE-2019-17091[10] occurred in Eclipse Mojarra, as used in Mojarra for Eclipse EE4J before 2.3.10 and Mojarra JavaServer Faces before 2.2.20. And CVE-2017-7658[11] occurred in Eclipse Jetty Server, involving versions 9.2.x and older, 9.3.x (all non HTTP/1.x configurations), and 9.4.x (all HTTP/1.x configurations). Therefore, we do not give the version number of Eclipse either.

[7]https://nvd.nist.gov/
[8]https://cve.mitre.org/
[9]https://nvd.nist.gov/vuln/detail/CVE-2018-1120
[10]https://nvd.nist.gov/vuln/detail/CVE-2019-17091
[11]https://nvd.nist.gov/vuln/detail/CVE-2017-7658

2.2 | Data analysis

In our study, we manually examine these 1,076 security bug reports to study their characteristics. In order to classify each CVE, we observed a security bug from the following four aspects to determine its category: (1) the description of the security bug on the CVE/NVD website; (2) the CVSS score and CWE category of the security bug on the NVD website; (3) reference links to the security bug in CVE/NVD websites, including Securityfocus, Securitytracker, Bugzilla, and other websites; and (4) if there is a website with the CVE code in the reference link (e.g., GitHub and Bugzilla), we further analyze the code information.

To improve the accuracy of manual marking, we calculate the Cohen's Kappa coefficient[3] to verify the consistency of the mark. Cohen's Kappa coefficient[4] is a measure used to measure the reliability of two raters between classified items. Assume that the element $p_{ij}(i, j \in 1, \ldots, k)$ in the agreement table $A(k \times k)$ represents the proportion of the total number of objects that evaluator E_a is placed in category i and evaluator E_b is placed in category j. The formulas for the totals p_{i+} and p_{+i} of each row and each column are as

$$p_{i+} = \sum_{j=1}^{k} p_{ij} \text{ and } p_{+j} = \sum_{i=1}^{k} p_{ij}.$$

Cohen's Kappa coefficient is defined as

$$\kappa = \frac{p_o - p_e}{1 - p_e},$$

where

$$p_o = \sum_{i=1}^{k} p_{ii} \text{ and } p_e = \sum_{i=1}^{k} p_{i+} p_{+i}.$$

In our research, the third and fourth authors, serving as evaluators, learn the examination scheme independently and manually mark the entries. When 5%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, and 100% of the security bugs are marked, we measure the difference between two evaluators using Cohen's Kappa coefficient rating aggregation. We conducted a total of 11 rounds of evaluation, and each round of evaluation included the previous round of security bug reports. First, the third and fourth authors independently analyze and label approximately 5% of the security bug reports, and Cohen's Kappa coefficient was both only about 25%. Then, we conduct training between two evaluators to determine the specific meaning of each category and modify the label. After the training, we conduct another 10% pilot study, including the top 5% of the security bugs. Then, Cohen's Kappa coefficient reached about 55%. Finally, we further discuss these situations through examples. The Kappa coefficient ranges from 0.0% to 100.0%, and the higher the value, the greater the consistency. In general, the Kappa coefficient of more than 80% is considered almost perfect agreement between the two assessments.[5] Through continuous studies, the coefficient of each category eventually exceeds 90%, so it can be considered that the evaluation in this study is consistent. For example, Table 2 shows the consistency of the two evaluators' classification of the root causes, and the final kappa coefficient is 90.40% (>90%).

TABLE 2 Consistency assessment of root cause classification of security bugs

	Evaluator E_a							
Evaluator E_b	TIM	CON	INP	MEM	LOG	NUM	UNK	Total (row)
TIM	3.8%	0.2%	0.1%	0.3%	0.1%	0.1%	0.1%	4.7%
CON	0.1%	32.7%	0.2%	1.3%	0.0%	0.0%	0.1%	34.4%
INP	0.0%	0.5%	10.4%	0.3%	0.0%	0.1%	0.1%	11.4%
MEM	0.2%	1.1%	0.3%	39.9%	0.1%	0.1%	0.2%	41.9%
LOG	0.2%	0.1%	0.0%	0.0%	0.6%	0.0%	0.0%	0.9%
NUM	0.0%	0.1%	0.1%	0.1%	0.0%	3.0%	0.1%	3.4%
UNK	0.1%	0.1%	0.0%	0.1%	0.0%	0.0%	3.0%	3.3%
Total (column)	4.4%	34.8%	11.1%	42.0%	0.8%	3.3%	3.6%	100.0%

Note: $\kappa = 90.40\%$.

With Cohen's Kappa coefficient, the labeling process is continuously being monitored. After the discussion, a final labeling result is determined. We have published the dataset[12] of this empirical study, which includes 1,076 security bugs for the five projects from 2015 to 2019, along with the corresponding classification information and published date.

2.3 | Categories of security bugs

Developers often encounter different types of security bugs. In our study, we mainly consider code-based security bugs. We have studied existing security vulnerabilities and the latest classification standards (CWE,[13] OWASP,[14] etc.). CWE classification method is an authoritative list of software and hardware weakness types. However, we observed some divergences in the CWE classification of security bugs in NVD. For example, in CVE-2017-7668,[15] there are two CWE classifications. NIST classifies it as CWE-20 (Improper Input Validation), but the Apache Software Foundation classified it as CWE-126 (Buffer Over-read). We decided not to use the CWE classification as security bug category. However, our classification still has a certain mapping relationship with the CWE classification. For example, authentication restriction and authorization management can be mapped to CWE-284 (Improper Access Control), and security configuration can be mapped to CWE-693 (Protection Mechanism Failure) and CWE-703 (Improper Check or Handling of Exceptional Conditions), and so forth. OWASP summarizes the top 10 most likely, common, and dangerous security concerns for Web applications, along with suggestions on how to eliminate them. Such categories are based on the frequency of vulnerability occurrence, while some infrequent categories are not always included, such as OWASP Top 10 (2017), which lacks an important concurrent execution category. Therefore, we observed a large number of security bug reports and came up with a new category that is both easy to use and comprehensive. This category is based on the function attacked by security bugs, that is, the attacker carried out an attack through this function and achieved the expected purpose. The specific categories are as follows:

2.3.1 | Injection

Injection can be used by attackers to import code into a specific computer program to change the purpose of the program or the running process. It includes SQL injection, XML injection, and code injection. For example, in CVE-2018-5158,[16] the PDF viewer does not sufficiently sanitize PostScript calculator functions. This will allow malicious Java-Script to be injected through a crafted PDF file.

2.3.2 | Authentication restriction

If the software cannot correctly verify the user's identity or the access restriction policy of the software is bypassed by a malicious user, the data may be illegally accessed or obtained. For example, in CVE-2018-1312,[17] when generating an HTTP Digest authentication challenge, the nonce sent to prevent reply attacks was not correctly generated. In a cluster of servers using a common digest authentication configuration, HTTP requests could be replayed across servers by an attacker.

2.3.3 | Authorization management

If software or operating systems do not grant privileges to the user properly, the user is not able to perform normal operations or obtains unexpected operational privileges.

2.3.4 | Memory operation

Memory operation refers to a series of problems caused by improper operation of pointers and buffers in the memory space. It can lead to abnormal memory allocation, including buffer overflow, null-pointer exception, and out-of-bounds read and write. For example, in CVE-2019-9805,[18] a

[12]https://github.com/YingWeiYZU/Security-Bug-Data-Analysis-Summary
[13]https://cwe.mitre.org/
[14]https://www.owasp.org
[15]https://nvd.nist.gov/vuln/detail/CVE-2017-7668
[16]https://nvd.nist.gov/vuln/detail/CVE-2018-5158
[17]https://nvd.nist.gov/vuln/detail/CVE-2018-1312
[18]https://nvd.nist.gov/vuln/detail/CVE-2019-9805

```
63  63          prop->name = kstrdup(name, GFP_KERNEL);
    64  +       if (!prop->name) {
    65  +               dlpar_free_cc_property(prop);
    66  +               return NULL;
    67  +       }
```

FIGURE 1 Code Information of CVE-2019-9805

```
2549        -   if (((int)arg >= cdi->capacity))
    2549    +   if (arg >= cdi->capacity)
2550  2550              return -EINVAL;
2551  2551      return cdrom_slot_status(cdi, arg);
```

FIGURE 2 Code Information of CVE-2018-16658

latent security bug exists in the Prio library where data may be read from uninitialized memory, leading to potential memory corruption. In addition, code information can help us determine the category. In the source code of CVE-2019-12614[19] (Figure 1), a security bug occurred when the kstrdup() function applied for memory allocation for prop->name on line 65. Because the return value of this function may be NULL, the program does not check it when performing memory operations, which may cause an attacker to conduct a Dos attack.

2.3.5 | Security configuration

Security configuration refers to improper security configuration of software due to software designers' inadequate understanding and design of internal security constraint policies. For example, line 2549 of CVE-2018-16658[20] (Figure 2) performs a coercive conversion of the data type, converting unsigned long to int. It will cause the boundary check to fail due to specially crafted input, allowing local attackers to read the memory. This is due to the inadequate design of the security policy by the developer, which leads to a security bug in the security configuration.

2.3.6 | Data encryption

Improper encryption of internally sensitive data by software or operating systems before storage or transmission may cause users to obtain data through some malicious operations.

2.3.7 | Resource management

Improper request or release of resources by operating systems will lead to errors in resource allocation management, or lack of certain restrictive measures for control reference of external resources. It includes request or release of invalid pointers or references, unlimited resource allocation, open redirection and a series of specific manifestations.

2.3.8 | Concurrent execution

If the software is unable to handle the concurrent invocation and execution of exceptional threads, it may result in concurrency execution security bug. For example, in CVE-2016-2069,[21] race condition in the Linux kernel before 4.4.1 allows local users to gain privileges by triggering access to a paging structure by a different CPU.

[19]https://github.com/torvalds/linux/commit/efa9ace68e487ddd29c2b4d6dd23242158f1f607
[20]https://github.com/torvalds/linux/commit/8f3fafc9c2f0ece10832c25f7ffcb07c97a32ad4
[21]https://nvd.nist.gov/vuln/detail/CVE-2016-2069

```
745  745        if (!desc->bNrInPins)
746  746                return -EINVAL;
     747  +     if (desc->bLength < sizeof(*desc) + desc->bNrInPins)
     748  +             return -EINVAL;
```

FIGURE 3 Code Information of CVE-2019-15117

2.3.9 | Unknown

Due to insufficient description information of the security bug reports, the type of these security bugs is unknown. In one case, the information provided by CVE was so vague that it was impossible to judge the categories; for example, CVE-2016-2806[22] only mentioned Multiple Unspecified Vulnerabilities. The other situation is that unclear information makes it difficult to classify. For example, in CVE-2016-5243,[23] the Linux kernel's tipc_nl_compat_link_dump function does not copy a specific string correctly, resulting in a bug. It is difficult to distinguish the categories.

2.4 | Classification of root causes

This classification is obtained by referring to the work of Li et al.[6] There are nine categories, including BOV (Bohr-Vulnerability), TIM (Time-related vulnerability), ENV (Software environment related vulnerability), SDD (Security design defect vulnerability), HDF (Hardware design fault vulnerability), MEM (Memory related vulnerability), LOG (Logical resources related vulnerability), NUM (Numerical errors related vulnerability), and UNK (Unknown vulnerability). According to our observations, We remove BOV because this category is not related to root cause. Security bugs in ENV appear in complex interactions between memory environments and software, so we merge ENV into MEM. In our classification of root causes of security bugs, we rename SDD to CON (Configuration Error), which is more suitable to our naming rules. The difference between security bugs and vulnerabilities is that security bugs only exist in software, and vulnerabilities exist not only in software, but also in hardware. Therefore, we delete the category HDF which is caused by hardware design. In addition, there are some security bugs that are difficult to classify manually. For instance, the description in CVE-2018-12392[24] shows that while attackers open a document through a script and manipulate user events in nested loops, it is possible to trigger a potentially exploitable crash due to poor event handling. We do not have adequate information to judge its category since we only know its reason is "poor event handling" and almost all bugs can be concluded as "poor handling." Since we do not know how such a script works or gets into effect, we set a category as UNK (unknown). Based on the above analysis, we classify the root causes of security bugs into the following seven categories.

2.4.1 | TIM (time-related error)

Security bugs with certain race conditions or other strict timing requirements that can be exploited for attacks are categorized as TIM (e.g., thread deadlocks, thread concurrency). For example, in CVE-2018-5814,[25] multiple race conditions occur when handling probe, disconnect, and rebind operations. It can be exploited to trigger a use-after-free condition or a NULL pointer dereference.

2.4.2 | CON (configuration error)

Attackers would exploit security bugs through the careless and thoughtless security configuration (e.g., improper access control, default settings and rights management). For example, in CVE-2017-7674,[26] the CORS Filter in Apache Tomcat did not add an HTTP Vary header. This permitted client and server side cache poisoning in some circumstances. When CVE has related code links, we also used the code information as the basis for judging the category. For example, in CVE-2019-15117[27] (Figure 3), the effective length of the default descriptor is >= 5+"bNrInPins," but in actual use, the length of the descriptor is invalid. This configuration error will cause memory access beyond the range.

[22]https://nvd.nist.gov/vuln/detail/CVE-2016-2806
[23]https://nvd.nist.gov/vuln/detail/CVE-2016-5243
[24]https://nvd.nist.gov/vuln/detail/CVE-2018-12392
[25]https://nvd.nist.gov/vuln/detail/CVE-2018-5814
[26]https://nvd.nist.gov/vuln/detail/CVE-2017-7674
[27]https://github.com/torvalds/linux/commit/daac07156b330b18eb5071aec4b3ddca1c377f2c

```
58        -             if (indev != NULL) {
          58  +         if (indev && indev->ifa_list) {
59        59                    ifa = indev->ifa_list;
60        60                    newdst = ifa->ifa_local;
```

FIGURE 4 Code Information of CVE-2015-8787

```
965   965                 map_bh(bh, inode->i_sb, map.m_pblk);
966   966                 bh->b_state = (bh->b_state & ~F2FS_MAP_FLAGS) | map.m_flags;
967         -             bh->b_size = map.m_len << inode->i_blkbits;
      967   +           bh->b_size = (u64)map.m_len << inode->i_blkbits;
```

FIGURE 5 Code Information of CVE-2017-18257

2.4.3 | INP (input validation error)

Security bugs led by improper input validation are classified as INP (e.g., SQL injection, cross-site scripting[7] [XSS], format string attack, and HTTP response splitting). For example, in CVE-2016-6816,[28] the code in Apache Tomcat parsed the HTTP request line that permitted invalid characters. This could be exploited in conjunction with a proxy that also permitted the invalid characters but with a different interpretation, to inject data into the HTTP response.

2.4.4 | MEM (memory error)

This type of security bugs can be exploited to attack software system by consuming a lot of physical resources (e.g., memory) without freeing. Sometimes this kind of security bugs can also exploit the wrong approach to manipulate such resources (e.g., buffer overflow, memory leak, and stack exhaustion). For example, in CVE-2015-8787[29] (Figure 4), the if judgment on line 58 lacks judgment on whether indev->ifa_list is a null pointer. This memory error may cause null pointer dereference and system crash.

2.4.5 | LOG (logic resource error)

This type of security bugs can be exploited by leaking logical resources repeatedly without freeing (e.g., computing time and infinite loop). For example, in CVE-2018-17189,[30] attackers could send request bodies to plain resources in a slow loris way. The results are that h2 stream for that request would occupy a server thread unnecessarily and that incoming data would be cleaned up.

2.4.6 | NUM (numeric error)

This type of security bugs can be utilized by accumulating numeric errors (e.g., integer overflow). For example, in CVE-2017-8278,[31] integer overflow could occur while reading audio data from an unspecified driver. For example, in code of CVE-2017-18257 [32] (Figure 5), the map.m_len data type is 32-bit unsigned int, Bh-> b_size data type is a size_t which is 64 bits on 64 bits archtecture. Conversion from an unsigned int to size_t will cause an integer overflow error. It may cause a denial of service (integer overflow and loop).

2.4.7 | UNK (unknown error)

Due to insufficient description information of security bug reports, we can not analyze the root causes of these security bugs. For example, in CVE-2018-1000199,[33] we can only obtain information that the Linux kernel contains a dangerous functional vulnerability, so it is difficult to

[28]https://nvd.nist.gov/vuln/detail/CVE-2016-6816
[29]https://github.com/torvalds/linux/commit/94f9cd81436c85d8c3a318ba92e236ede73752fc
[30]https://nvd.nist.gov/vuln/detail/CVE-2018-17189
[31]https://nvd.nist.gov/vuln/detail/CVE-2017-8278
[32]https://github.com/torvalds/linux/commit/b86e33075ed1909d8002745b56ecf73b833db143
[33]https://nvd.nist.gov/vuln/detail/CVE-2018-1000199

TABLE 3 CVSS v3.0 ratings

Severity	Base score range
None	0.0
Low	0.1–3.9
Medium	4.0–6.9
High	7.0–8.9
Critical	9.0–10.0

classify it. In CVE-2016-1966,[34] the submitter only gave the function where the security bug occurred. Therefore, due to the lack of information, it is difficult to analyze the root causes of these CVEs.

2.5 | Classification of severity

CVSS (Common Vulnerability Scoring System) is an industry open standard which is designed to measure the severity of security bugs. It is widely used to evaluate the security of enterprise systems and is supported by eBay, Symantec, Cisco, Oracle, and many other vendors. Generally, CVSS score consists of three metrics: base, temporal, and environmental. Base score represents the intrinsic characteristics of a security bug that is constant over time and across user environments. It is composed of two sets of metrics: the exploitability metrics (attack vector, attack complexity, privileges required, user interaction, and scope) and the impact metrics (confidentiality impact, integrity impact, and availability impact). Our study on the impact of security bugs is based on the base score rating of CVSS V3.0.[35] The severity and the corresponding base score ranges are shown in Table 3. As we can see, the higher score the security bug has, the more harmful the security bug is. For example, a security bug with a CVSS benchmark score of 10.0 is typically a security bug that can be completely breached. It indicates that the attacker can take full control of a system, including the management or "root" rights of the operating system layer. CVSS v3.0 rating includes the following five levels.

2.5.1 | None

Security bugs with *none* severity do not cause any impact or damage.

2.5.2 | Low

Security bugs with *minor* severity can lead to denial of service attacks, data leaks or deceptions.

2.5.3 | Medium

Security bugs with *medium* severity may be difficult to exploit in some cases but may still result in loss of confidentiality, integrity, or availability of resources. These types of security bugs may have had a serious impact, but are less likely to be exploited based on a technical assessment.

2.5.4 | High

Security bugs with *high* severity can easily compromise confidentiality, integrity, or resource availability, which can allow local users to gain privileges, allow unauthenticated remote users to see resources that should be protected by authentication, and so on.

[34]https://nvd.nist.gov/vuln/detail/CVE-2016-1966
[35]https://www.first.org/cvss/specification-document

2.5.5 | Critical

Security bugs with *critical* severity can be used to run attacker code and install software, without requiring user interaction beyond normal browsing.

2.6 | Classification of consequences

Security bugs also have characteristics that cause serious consequences in a short time, such as zero-day attacks (the same malicious programs appear on the same day as the bug exposure).[8] Therefore, after the security bug is exposed and exploited, developers should first analyze the causes of the security bugs according to the consequences of the attack, so that they can provide targeted fixes. We classify the consequences based on investigations on the actual security bug data provided by NVD and other resources[36,37] which also focus on security bugs (e.g., red hat and security tracker). The classification results of consequences are as the follows.

2.6.1 | Data leakage

Data leakage is one common kind of consequences of security bugs in software projects. It is a security incident in which sensitive, protected, or confidential data are copied, transmitted, viewed, stolen, or used by an unauthorized individual. For example, in CVE-2018-12400,[38] favicons are cached in the cache/icons folder as they are in non-private mode. This produces information leakage of sites visited during private browsing sessions.

2.6.2 | DoS (denial of service)

DoS is a cyber-attack in which the perpetrator seeks to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting services of a host connected to the Internet.[9] DoS is typically accomplished by flooding the targeted machine or resource with superfluous requests and prevent some even all legitimate requests from being fulfilled.

2.6.3 | Memory corruption

Memory corruption occurs when CPU uses uninitialized memory or uses memory that is out of allocation (buffer overflow).[10] This consequence is caused by improper operation of pointers to memory. When memory corruption occurs, a user will receive unexpected outputs.

2.6.4 | Deception

Deception is a situation in which a person or a program successfully masquerades as another by falsifying data, to gain an illegitimate advantage. Deception can be a spoofing of a file name, a spoofing of a domain name, or misleading some operations of a user. For example, in CVE-2017-7762,[39] reader mode did not strip the username and password section of URLs displayed in the address bar when entering directly. This can be used for spoofing the domain of the current page.

2.6.5 | Crash

Crash occurs when a client program fails due to an unhandled exception or error at runtime. Most crashes are the results of executing invalid machine instructions. It allows the replication of viruses or the acquisition of inaccessible data. For example, in CVE-2019-9790,[40] a potentially exploitable crash will happen when a raw pointer to a DOM element on a page is obtained and the element is then removed while still in use.

[36]https://securitytracker.com/
[37]https://access.redhat.com/errata/
[38]https://nvd.nist.gov/vuln/detail/CVE-2018-12400
[39]https://nvd.nist.gov/vuln/detail/CVE-2017-7762
[40]https://nvd.nist.gov/vuln/detail/CVE-2019-9790

2.6.6 | Code execution

Code execution is commonly achieved through control over the instruction pointer (such as a jump or a branch) of a running process. In order to execute arbitrary code, attackers inject code into the process and change the instruction pointer to make it point to the injected code.

2.6.7 | Privilege escalation

Privilege escalation is performed by exploiting a bug, design bug, or configuration oversight in an operating system or software application to gain elevated access to resources that are normally protected from an application or user. For example, in CVE-2019-14835,[41] a privileged guest user is able to pass descriptors with invalid length to the host when migration is underway. So they could use this security bug to increase their privileges on the host.

2.6.8 | Security bypass

Security bypass occurs when an attack can bypass a security protection mechanism (e.g., client certificate verification).[11] In some cases, security bypass comes with other consequences (e.g., data leakage). For example, in CVE-2017-7787,[42] same-origin policy protections can be bypassed on pages with embedded iframes during page reloads. It will allow the iframes to access content on the top level page and further lead to information disclosure.

2.6.9 | Others

There are also some outcomes that are not clearly described and therefore do not fit into the above categories, and we classify these outcomes as other outcomes. For example, the consequence of CVE-2016-9077[43] is timing attacks, and the consequence of CVE-2016-2830[44] is tracking users. Their statements about the consequences are vague, making it difficult to classify.

2.7 | Classification of locations

When a security bug occurs, it is critical to find where the bug occurred. Therefore, we also study the location of security bugs. We classify the locations of security bugs into the following types.

2.7.1 | Web

Web here is the general term for front-end and back-end. Among them, front-end refers to user-oriented part, such as the user interface (e.g., address bar, bookmarks, and URLs), rendering engines (e.g., DOM documentation and CSS rules). Back-end is used to implement some code logic in combination with the database, such as data access engine (used to read and modify data from the database), JSP engines (used to explain modules that execute JS scripts). For example, in CVE-2018-5111,[45] when the text of a specially formatted URL is dragged to the address far from page content, the displayed URL can be spoofed to show a different site.

2.7.2 | Service

Service is typically composed of various externally imported packages, libraries, API interfaces, plugins, developer tools, and WebExtenison. For example, in CVE-2017-7821,[46] WebExtensions can download and attempt to open a file of some non-executable file types. This can be triggered

[41]https://nvd.nist.gov/vuln/detail/CVE-2019-14835
[42]https://nvd.nist.gov/vuln/detail/CVE-2017-7787
[43]https://nvd.nist.gov/vuln/detail/CVE-2016-9077
[44]https://nvd.nist.gov/vuln/detail/CVE-2016-2830
[45]https://nvd.nist.gov/vuln/detail/CVE-2018-5111
[46]https://nvd.nist.gov/vuln/detail/CVE-2017-7821

```
1082        -       if (IS_ERR(blkg)) {
1083        -               blkg_free(new_blkg);
        1082 +       if (IS_ERR(blkg))
1084    1083                return PTR_ERR(blkg);
1085        -       }
```

FIGURE 6 Code Information of CVE-2018-7480

without specific user interaction for the file download and open actions. The judgment of the location sometimes also depends on the code information. For example, from the commit information and the code file of CVE-2015-4170,[47] it can be seen that it occurred during the use of the tty reader/writer. Tty is the abbreviation for various types of terminal devices, so it can be considered that the error occurred on the Service.

2.7.3 | Memory

Most security bugs that occur on memory are related to improper memory operations, including pointer errors, buffer overflows, and stack overflows. For example, in CVE-2018-7480[48] (Figure 6), if the creation of blkg_create fails, the new_blkg passed as a parameter will be released by blkg_create, so there is no need to release it again. A double free error has occurred here, and it can be determined that the bug has occurred in memory.

2.7.4 | Database

The database we talked here mainly contains local files, caches, and documents. It also contains various data such as cookies saved by the browser on the hard disk, local storage, and so forth, which can be called by the API provided by the browser engine. For example, in the GitHub link of CVE-2018-13405,[49] we can learn from the code information that the bug occurred in the process of creating a file using the sgid directory, so we believe that the location of the security bug is the database.

3 | SECURITY BUG CATEGORIES

3.1 | Distribution of categories

The normalized distribution of security bug categories is shown in Table 4. The data in Table 4 represents the proportion of different types of security bugs in five projects. To answer RQ1, we can find that the most common security bug categories are memory operation, authentication restriction, and resource management. But we can see that the distribution of security bugs in different projects is different. For example, in Apache Tomcat and Eclipse, the most common type is security configuration. In HTTP server, authentication restriction and resource management security bugs are the most frequent security bugs, accounting for 31.4% and 25.7%, respectively. In addition, memory operation security bugs are more than other types of security bugs in large projects such as Firefox and even reach 46.7% in Linux Kernel.

> **Finding 1.1.** Memory operation is the most common security bug, but its project imbalance means that the proportion of vulnerability types in the project may be closely related to its programming language.

Through observation, the most significant type of bug is memory operation. As shown in Table 4, the number of memory operation bugs reached 381, accounting for more than one-third of the total. Among them, this type occupies a relatively small proportion in Tomcat and Eclipse, because the programming language of these two projects is Java. This is because when the application no longer needs these objects, Java will delete these objects through a mechanism called "garbage collection." However, only when an object is no longer referenced will it be considered useless. Therefore, no matter what programming language is used, developers still need to pay attention to memory management.

In addition, the proportion of authentication restriction security bugs is 8.6% to 31.4%, and the proportion of resource management security bugs is 8.2% to 25.7%. It can be seen that the total number of these two types of vulnerabilities is relatively large and has a proportion in each

[47] https://github.com/torvalds/linux/commit/cf872776fc84128bb779ce2b83a37c884c3203ae
[48] https://github.com/torvalds/linux/commit/9b54d816e00425c3a517514e0d677bb3cec49258
[49] https://github.com/torvalds/linux/commit/0fa3ecd87848c9c93c2c828ef4c3a8ca36ce46c7

TABLE 4 Statistics of security bug categories

Project	Tomcat	HTTP Server	Firefox	Linux Kernel	Eclipse	Total
Injection	1 (2.8%)	1 (2.9%)	20 (3.8%)	10 (2.4%)	5 (8.2%)	37 (3.4%)
Authentication restriction	8 (22.2%)	11 (31.4%)	97 (18.1%)	35 (8.6%)	10 (16.4%)	161 (15.0%)
Authorization management	4 (11.1%)	0 (0.0%)	71 (13.3%)	17 (4.2%)	0 (0.0%)	92 (8.6%)
Memory operation	1 (2.8%)	8 (22.8%)	170 (31.8%)	191 (46.7%)	11 (18.0%)	381 (35.4%)
Security configuration	12 (33.3%)	3 (8.5%)	53 (9.9%)	37 (9.0%)	24 (39.4%)	129 (12.0%)
Data encryption	0 (0.0%)	1 (2.9%)	3 (0.6%)	17 (4.2%)	1 (1.6%)	22 (2.0%)
Resource management	7 (19.4%)	9 (25.7%)	89 (16.6%)	43 (10.5%)	5 (8.2%)	153 (14.2%)
Concurrent execution	2 (5.6%)	1 (2.9%)	5 (0.9%)	38 (9.3%)	1 (1.6%)	47 (4.4%)
Unknown	1 (2.8%)	1 (2.9%)	27 (5.0%)	21 (5.1%)	4 (6.6%)	54 (5.0%)

Note: The percentage in brackets is the percentage of that type in a project.

TABLE 5 Statistics of root causes of security bugs

Project	Tomcat	HTTP Server	Firefox	Linux Kernel	Eclipse	Total
TIM	4 (11.1%)	1 (2.9%)	4 (0.8%)	35 (8.5%)	2 (3.3%)	46 (4.3%)
CON	22 (61.1%)	15 (42.8%)	210 (39.3%)	101 (24.7%)	35 (57.4%)	383 (35.6%)
INP	7 (19.4%)	12 (34.3%)	60 (11.2%)	29 (7.1%)	9 (14.8%)	117 (10.8%)
MEM	2 (5.6%)	4 (11.4%)	226 (42.2%)	213 (52.1%)	8 (13.1%)	453 (42.1%)
LOG	1 (2.8%)	2 (5.7%)	0 (0.0%)	5 (1.2%)	0 (0.0%)	8 (0.7%)
NUM	0 (0.0%)	0 (0.0%)	15 (2.8%)	18 (4.4%)	1 (1.6%)	34 (3.2%)
UNK	0 (0.0%)	1 (2.9%)	20 (3.7%)	8 (2.0%)	6 (9.8%)	35 (3.3%)

Note: The percentage in brackets is the percentage of that type in a project.

project. It is because identity verification and resource management are involved in different types of projects and may be malicious occupation or control of resources due to the lack of consideration of the security of identity verification and resource request methods.

4 | ROOT CAUSE OF SECURITY BUGS

4.1 | Distribution of root cause

The normalized statistics of root causes is shown in Table 5. The data in the Table 5 represents the proportion of different categories of root causes in Apache Tomcat, Apache HTTP Server, Mozilla Firefox, Linux Kernel, and Eclipse. To answer RQ2, we can see that MEM, CON, and INP are the primary root causes of security bugs.

> **Finding 2.1.** MEM is the most common root cause of security vulnerabilities, while CON and INP is a common cause of project universality.

It can be seen from Table 5 that the number of security bugs in Apache Tomcat, Apache HTTP Server, Mozilla Firefox, Linux Kernel, and Eclipse occurring due to MEM accounts for 5.6%, 11.4%, 42.2%, 52.1%, and 13.1% respectively. In Mozilla Firefox and Linux Kernel, MEM is the major cause of security bugs. The reason for the disparity with Apache is that Firefox and the Linux Kernel, as large applications, have more frequent calls to memory based on the user's interaction with the software, leaving the opportunity to attackers.

The proportion of CON in each project is 24.7% to 61.1%. Especially in Apache Tomcat and Eclipse, the security bugs caused by CON accounted for 61.1% and 57.4%. This may result not only from the default configuration of the software itself but also from components such as frameworks, libraries, and other software modules that contain known security bugs. CON occurs through the entire software development cycle. In addition, CON exists throughout the entire software development cycle and is also an important reason for its relatively high proportion.

INP is also the third largest root cause, with 7.1% to 34.3% of security bugs in the five projects caused by INP. For example, in CVE-2018-5172,[50] if a user pastes the script from the clipboard into the PDF viewer while viewing RSS feeds or PDF files, the PDF viewer can execute

[50]https://nvd.nist.gov/vuln/detail/CVE-2018-5172

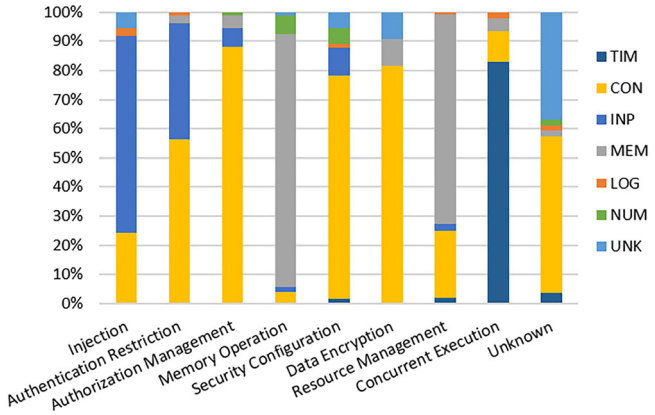

FIGURE 7 Relation between categories and root causes of security bugs

the injected script content. This allows a malicious site to copy and paste malicious script content. Then, the script content could run with the context of either page.

4.2 | Relation between categories and root causes of security bugs

Finding 2.2. CON usually leads to authorization management, security configuration, and data encryption security errors. This shows that security configuration is often related to information security that the public is concerned about.

Figure 7 shows the relation between categories and root causes of security bugs. The *x* axis represents nine categories of security bugs. Different colors on each bar represent the root causes of security bugs. As can be seen from Figure 7, CON accounts for the highest proportion in authorization management, security configuration security bugs as well as data encryption security bugs (over 76%), and even up to 88.0% in authorization management security bugs. This implies that the software security access strategy and algorithm bugs will have a serious impact. In addition, we found that memory operation and resource management security bugs are largely caused by MEM. The reason is that, when the software runs, it frequently request and allocate computing resources, which is largely related to CPU and memory. Importantly, most of the security bugs caused by CON are related to personal privacy, user privileges, and even corporate data. Therefore, developers should take precautions when writing programs, such as providing effective separation between components and users in the construction of the program architecture, confirming user permissions during data invocation to prevent broken access, and so on.

5 | SEVERITY OF SECURITY BUGS

5.1 | Distribution of severity

Finding 3.1. 17.9% of security bugs were of critical severity, with the Firefox project accounting for the largest number of critical vulnerabilities (28.8%). It inspired developers that they should consider the severity of bugs first, and the priority of fixing will increase as the severity of bugs increases.

To answer RQ3, Table 6 presents the normalized distribution of different kinds of severity in the five projects. From Table 6, we can see that, in total, security bugs with medium and high severity account the most, and security bugs with low severity account the least. Specifically, the severity of security bugs in most projects is high, with an average of more than 40%. The severity of 17.9% of the vulnerabilities are critical, with the highest proportion in Firefox (28.8%). Although the number of critical security bugs is not large, we still need to focus on them because security bugs granted at this rating can easily compromise confidentiality, integrity or resource availability. Therefore, when developers encounter multiple security bugs, they should consider their severity firstly, and increase the repair priority of the bugs with higher severity to avoid Irretrievable losses. At the same time, it also inspired researchers to study methods for automatically judging the severity of bugs based on submitted bug information and project information.

TABLE 6 Statistics of severity of security bugs

Project	Tomcat	HTTP Server	Firefox	Linux Kernel	Eclipse	Total
Low	0 (0.0%)	0 (0.0%)	2 (0.4%)	10 (2.4%)	0 (0.0%)	12 (1.1%)
Medium	12 (33.3%)	10 (28.6%)	174 (32.5%)	219 (53.6%)	16 (26.2%)	431 (40.1%)
High	20 (55.6%)	19 (54.3%)	205 (38.3%)	164 (40.1%)	32 (52.5%)	440 (40.9%)
Critical	4 (11.1%)	6 (17.1%)	154 (28.8%)	16 (3.9%)	13 (21.3%)	193 (17.9%)

Note: The percentage in brackets is the percentage of that type in a project.

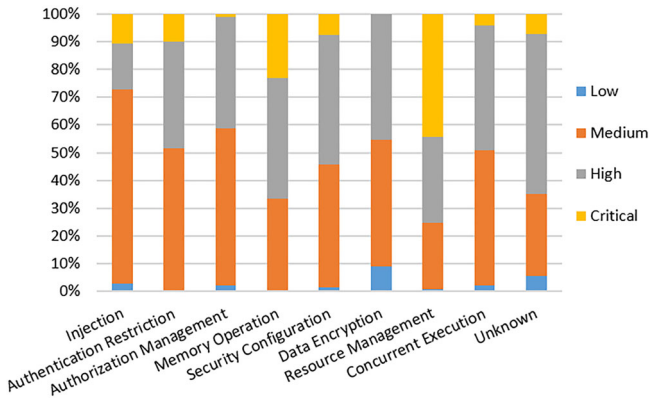

FIGURE 8 Relation between categories and severity of security bugs

5.2 | Relation between categories and severity of security bugs

Finding 3.2. Memory operation and resource management security bugs account for the majority of critical security bugs categories.

Figure 8 shows the relation between categories and severity of security bugs. The *x* axis represents nine categories of security bugs. Different colors on each bar represent the severity of security bugs. As can be seen from Figure 8, the proportion of critical security bugs in memory operation and resource management security bugs is the highest of all categories. This finding is consistent with the pattern we found in RQ2 that developers need to be especially careful with memory operation and resource management security bugs. Their serious impact has caused huge losses to companies and users. Furthermore, the severity of resource management and memory operations security bugs are critical.

In addition, by investigating the severity of the security bugs in Figure 7, we found that although the incidence of some bugs is not high, their severity is not low, which deserves the attention of developers. For example, in most cases, resource management is caused by MEM, but in CVE-2017-5651,[51] if the file processing is completed quickly, the processor may be added to the processor cache twice. This may cause the same processor to be used for multiple requests, which may lead to unexpected errors and/or response confusion. Therefore, the vulnerability is a resource management vulnerability caused by TIM, and its severity is critical.

6 | CONSEQUENCES OF SECURITY BUGS

6.1 | Distribution of consequences

Table 7 shows the statistics of different kinds of consequences in different projects. From Table 7, we can get the answer to RQ4, the primary consequences caused by the security bugs are DoS(26.3%), memory corruption(19.6%), and data leakage(15.0%).

DoS is an important concern for software developers, which accounts for 13.9%, 37.1%, 12.2%, 46.0%, and 15.4% in five projects, respectively. This type of consequence is often caused by a lack of resources. Hackers have many ways to attack software to produce DoS. Through the analysis of security bug reports, we find that the attack principle is mainly to make the victim host or network unable to receive, process or timely respond to external requests.

[51]https://nvd.nist.gov/vuln/detail/CVE-2017-5651

TABLE 7 Statistics of consequences of security bugs

Project	Tomcat	HTTP Server	Firefox	Linux Kernel	Eclipse	Total
Data leakage	19 (52.8%)	6 (17.1%)	87 (14.0%)	62 (12.8%)	13 (20.0%)	187 (15.0%)
Code execution	3 (8.3%)	0 (0.0%)	109 (17.5%)	8 (1.7%)	7 (10.8%)	127 (10.2%)
Memory corruption	0 (0.0%)	1 (2.9%)	98 (15.8%)	137 (28.3%)	7 (10.8%)	243 (19.6%)
DoS	5 (13.9%)	13 (37.1%)	76 (12.2%)	223 (46.0%)	10 (15.4%)	327 (26.3%)
Deception	1 (2.8%)	1 (2.9%)	62 (10.0%)	1 (0.2%)	3 (4.6%)	68 (5.5%)
Crash	0 (0.0%)	5 (14.2%)	101 (16.3%)	0 (0.0%)	2 (3.1%)	108 (8.7%)
Privilege escalation	0 (0.0%)	1 (2.9%)	27 (4.4%)	37 (7.6%)	3 (4.6%)	68 (5.5%)
Security bypass	8 (22.2%)	7 (20.0%)	39 (6.3%)	15 (3.0%)	5 (7.7%)	74 (6.0%)
Others	0 (0.0%)	1 (2.9%)	22 (3.5%)	2 (0.4%)	15 (23.0%)	40 (3.2%) Verified

Note: The percentage in brackets is the percentage of that type in a project.

It can be seen from Figure 7 that memory operation bugs are basically caused by MEM. Therefore, there is no doubt that due to the large number of security bugs of memory operation, the number of memory corruption is also large. In addition, the data in Table 7 show that data leakage accounts for 52.8%, 17.1%, and 20.0% in Tomcat, HTTP Server, and Eclipse, compared with 14.0% and 12.8% in Firefox and Linux Kernel. Data leakage is one of the common consequences of security bugs in Tomcat and HTTP server. Through the analysis of security bug reports, we find that data leakage usually occurs when the program uses the functionality of output or log. For example, during debugging of a web project, developers often need to see the debug information in the output. However, a large number of log files generated by long-running may have a certain impact on the stability of the system after the deployment of Tomcat.

6.2 | Relation between categories and consequences of security bugs

Finding 4.1. Data leakage is the main consequence of authentication restriction and data encryption security bugs. In comparison, the proportion of data leakage caused by memory operations is only 3.2%, but their severity is medium or high, which is still worthy of attention.

Figure 9 shows the relation between categories and consequences of security bugs. The *x* axis represents nine categories of security bugs. Different colors on each bar represent the consequences of security bugs. As we can see from Figure 9, data leakage is associated with authentication restriction and data encryption security bugs, accounting for over 50%. It may be due to the two types of security bugs involving data acquisition permissions and encryption methods, respectively. In contrast, memory operation bugs only account for 3.2% of data leakage, but the severity of these bugs is medium or high. Therefore, when developers pay attention to the frequent bugs, they should not ignore the small number but high-risky bugs. In addition, resource management security bugs often lead to crash. Memory operation and concurrent execution security bugs often lead to DoS. Finally, authorization management usually leads to the consequence of data leakage, but there are also some special cases. For example, in CVE-2018-8781,[52] the function has an integer overflow vulnerability allowing local users with access to the udldrmfb driver to obtain full read and write permissions on kernel physical pages, resulting in a code execution in kernel space.

6.3 | Relation between root causes and consequences of security bugs

Finding 4.2. Security bugs caused by LOG lead to DoS.

Figure 10 shows the relation between root causes and consequences of security bugs. The *x* axis represents seven root causes of security bugs. Different colors on each bar represent the consequences of security bugs. As can be seen from Figure 10, a common consequence of the security bugs caused by LOG is DoS. What's more, TIM, NUM, and MEM also frequently cause DoS, reaching 47.2%, 43.9%, and 30.5%, respectively. The reason for this correlation may be that most of the above root causes are malicious requests and consumption of computer resources. They disable the software to provide normal service or access to resources, making the attacked software stop responding.

[52]https://nvd.nist.gov/vuln/detail/CVE-2018-8781

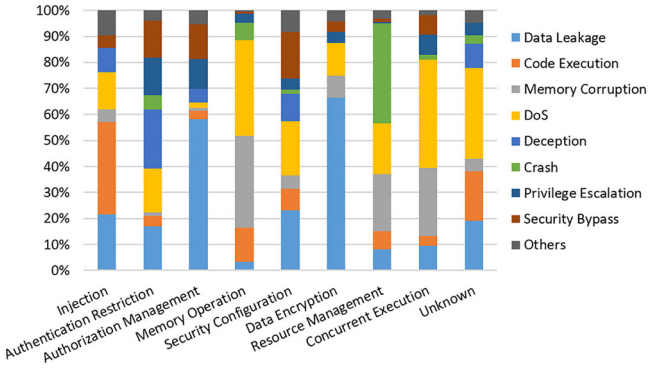

FIGURE 9 Relation between categories and consequences of security bugs

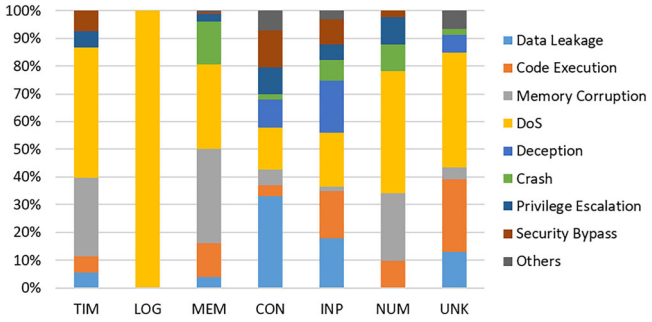

FIGURE 10 Relation between root causes and consequences of security bugs

7 | LOCATIONS OF SECURITY BUGS

7.1 | Distribution of locations

Finding 5.1. In general, services are the most frequent location for security breaches.

The normalized statistics of locations is shown in Table 8. The data in Table 8 represents the proportion of different kinds of locations in five projects. To answer RQ5, we can see that the most frequently attacked location in Apache Tomcat, Apache HTTP Server, and Mozilla Firefox is web, accounting for 63.9%, 82.9%, and 47.3%, respectively. Furthermore, the highest proportion of Linux Kernel is service, because it is an open source system kernel.

The normalized statistics of the location are shown in Table 8. The data in Table 8 represents the proportion of different types of locations in the five projects. Answering RQ5, in terms of overall quantity, service is the location where security bugs occur most frequently. The number of Linux projects occurring on services accounted for the largest proportion (47.2%), nearly half. This is because Linux Kernel is an operating system kernel, unlike other Linux distributions (e.g., Ubuntu), and it does not have many interfaces that interact with other applications. Instead, it has a lot of internal system calls.

However, the web, whose frequency is only lower than that of service, is universal, accounting for 63.9%, 82.9%, 47.3%, 3.9%, and 37.7% of Apache Tomcat, HTTP Server, Mozilla Firefox, Linux, and Eclipse, respectively. At the same time, we can see that the web is the most vulnerable position among the first three open source projects. The reason is that Apache Tomcat and Apache HTTP Server are web application servers and Mozilla Firefox is a web browser. These three projects have many external interfaces to interact with other applications. In summary, we can see that the location of the security bug is closely related to the core function of the project.

7.2 | Relation between categories and locations of security bugs

Finding 5.2. The most common category of security bugs that occur on service is concurrent execution bug.

TABLE 8 Statistics of Locations of Security Bugs

Project	Tomcat	HTTP Server	Firefox	Linux Kernel	Eclipse	Total
Web	23 (63.9%)	29 (82.9%)	253 (47.3%)	16 (3.9%)	23 (37.7%)	344 (32.0%)
Service	12 (33.3%)	0 (0.0%)	193 (36.1%)	193 (47.2%)	22 (36.1%)	420 (39.0%)
Memory	0 (0.0%)	2 (5.7%)	55 (10.3%)	162 (39.6%)	10 (16.4%)	229 (21.3%)
Database	1 (2.8%)	4 (11.4%)	34 (6.3%)	38 (9.3%)	6 (9.8%)	83 (7.7%)

Note: The percentage in brackets is the percentage of that type in a project.

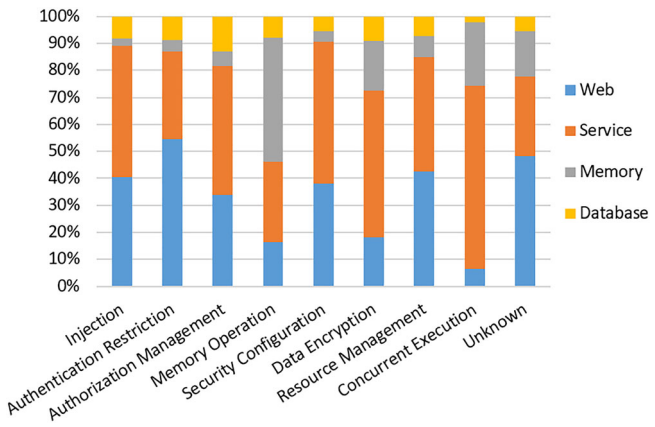

FIGURE 11 Relation between categories and locations of security bugs

Figure 11 shows the relation between categories and locations of security bugs. The *x* axis represents nine categories of security bugs. Different colors on each bar represent the locations of security bugs. From Figure 11, we notice that service is the most vulnerable location for all types of security bugs. Service usually consists of a variety of external imported packages, libraries, API interfaces, plugins, developer tools, and WebExtenison. Therefore, a variety of imported third-party programs are often vulnerable to attack, which may be the reason why the service is the common location for security bugs. As can be seen from Figure 11, the proportion of security bugs occurring on service in concurrent execution security bugs is the highest of all categories, accounting for 68.1%, followed by data encryption(54.5%), security configuration (52.7%), and injection(48.6%). In addition, we find that authentication restriction security bugs most often occur on the web, accounting for 54.7%, because user authentication is usually a necessary module for accessing resources in web.

7.3 | Relation between root causes and locations of security bugs

Finding 5.3. Security bugs caused by INP mainly occur on web.

Figure 12 shows the relation between root causes and locations of security bugs. The *x* axis represents seven root causes of security bugs. Different colors on each bar represent the locations of security bugs. We can see from the Figure 12 that although security bugs are caused by various reasons, they mostly occur on service and web. The proportion of MEM and NUM occurred in memory is the highest among all root causes. In addition, we find that 65.2% of the security bugs caused by TIM occur on service. The possible reason is that, there are a large number of external interfaces available to developers on service, while hackers can input malicious code for frequent read and write operations. Therefore, when multiple processes or threads read and write data frequently, race conditions appear and result in security bugs. Moreover, 53% of security errors caused by INP occur on the Web. Therefore, there may be a large number of malicious attacks on the Web, such as entering carefully prepared sentences in the address bar.

Finding 5.4. Among the security bugs caused by CON, most of the authentication restriction bugs occurred on the web, and more than half of the security configuration bugs occurred on the service.

Combining the association between the category and the root cause, we observed the location of the security bugs corresponding to Figure 7. It is worth noting that, among the security bugs caused by CON, 58.2% of the authentication restriction bugs occurred on the web, and

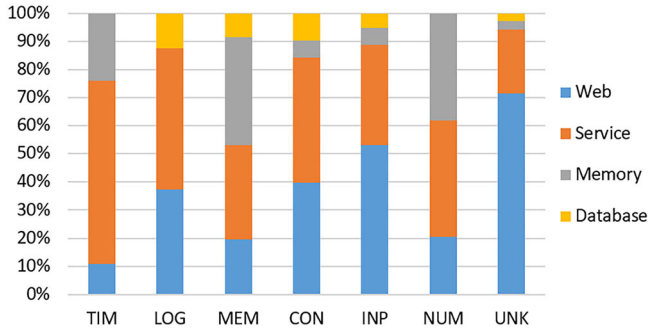

FIGURE 12 Relation between root causes and locations of security bugs

more than half of the security configuration bugs occurred on the service. This is because Web interacts with users most frequently, and service contains a large number of third-party libraries and various interfaces. Therefore, developers should carefully introduce various APIs and pay attention to the bug release of corresponding API versions in time to reduce the occurrence of vulnerabilities.

8 | DISCUSSION

In this section, we discuss the practical value of our study from five aspects.

First, from RQ1, we can see that most of the security bugs in open source projects are memory operation, authentication restriction, resource management and security configuration security bugs. Take memory operation security bugs for example, they occur with high frequency (up to 46.7% in Linux Kernel), so developers need to add more tests for identifying such security bugs in the development of software and think more about determining the rationality and feasibility of resource references when developing programs, thus reducing the occurrence of such security bugs. For example, developers should pay attention to judging the rationality of resource requests, and clean up the memory of used resources in time. Also, developers can learn from Tomcat's measures to add some memory protection mechanism to prevent this kind of security bugs.

Second, we can notice that authorization management, data encryption, authentication restriction and security configuration security bugs are often caused by CON. This requires developers to put more effort on using a secure development methodology. For example, authorization management needs to consider the allocation and control of permissions, which often involves the security configuration of corresponding permissions for different resources. Also, researchers can develop detection tools to detect abnormal data flow related to permission privacy in software. In addition, Of all the categories of security bugs, injection and authentication restriction security bugs account for the highest proportion of security bugs caused by INP. Therefore, developers should fully consider the robustness of the code when programming. Well-designed, comprehensive test cases are helpful to it because test-driven development often works well. Besides, MEM is a major cause of memory operation and resource management security bugs. Developers need to use efficient memory detection tools and have good programming habits (e.g., avoiding memory requests in loops).

Third, we can find from RQ3 that the impact of memory operation and resource management security bugs are critical. Developers should pay more attention to these two types of security bugs. Moreover, after the security bug is exposed, the severity of the bug should be defined first, and the priority of security bug fixing should be determined according to the severity, in case security bugs which have high severity bring serious consequences. Researchers should develop security bug analysis and fixing technology for these types of security bugs to assist developers for quick fixing. In addition, researchers should develop corresponding testing technology to identify this type of security bugs to avoid serious consequences.

Fourth, we can conclude from RQ4 that, although the diversity of the consequences of security bugs makes it difficult for developers to trace the causes of security bugs, there are still some attack patterns. For example, Deception is often associated with authentication restriction and data encryption security bugs. This gives developers an inspiration that when a security bug occurs, there are correlations between its types and consequences, and corresponding tests can be conducted to verify this assumption, thus improving the fixing efficiency.

Finally, we can see from RQ5 that security bugs which are usually from external extensions such as insecure external libraries and interfaces often occur on service. Therefore, developers should use external resources or services carefully and pay attention to the security of these imported third-party programs. At the same time, developers need to pay close attention to updates and error reports of imported programs. Moreover, researchers should study how to recommend trusted services for developers to reduce the occurrence of these security bugs.

9 | THREATS TO VALIDITY

In this section, we discuss threats to our empirical study, including external threats and internal threats.

9.1 | External threats

External threats to our study is the reliability of the data sources. First, the number of projects with security bugs is huge. Our research only selected five well-known projects for analysis, and the projects mainly involve browser, operating system, and integrated development environment (IDE). Therefore, the research scope of this paper still has limitations, and the analysis results may not be applicable to other projects. Second, a potential threat to our study arises from data collection. We work with 1076 publicly reported security bugs of Apache Tomcat, Apache HTTP Server, Mozilla Firefox, Linux Kernel and Eclipse mined from CVE and NVD, and they are selected from 2015 to 2019. We delete some security bugs that are reported before 2015 and after 2019, because the security bugs before 2015 are no longer time-sensitive and those after 2019 have not been fixed. This may lead to bias in the accuracy of the data. Third, due to different reports of security bugs written by different people, the quality of the reports varies widely. For instance, some bug reports may only provide few words about the security bugs, using the word like 'possibly', which causes the ambiguity of the description.

9.2 | Internal threats

There is a major internal threat to our empirical study. In our study, data analysis was all conducted manually. In the initial classification stage, due to the insufficient coverage of classification, the classification strategy needs to be adjusted continuously, resulting in many inconsistencies in the early stage of data analysis. To mitigate this threat, two trained evaluators independently studied the security bugs entries reported by official maintenance documents to label them. The inter-rater agreements are measured using Cohen's Kappa coefficient and the disagreements are reconciled under the monitoring of a supervisor. In addition, when investigating the associations between various categories, we only selected those with meaningful findings instead of studying and writing each association.

10 | RELATED WORK

10.1 | Empirical study of security bugs

Tan et al[12] explored the trend of security bugs in Mozilla, Apache, and Linux. The results showed that security bugs increased significantly over time in terms of number and proportion in these projects. Furthermore, they found that semantic bugs were the dominant root cause of security bugs. Different from their work, our work studies security bug categories, root causes, severity, impacts, and locations. The most critical part in our work is to give another classification criteria for security bugs and also study the correlation between different categories of security bugs. Ayanam et al[13] discussed the basic concept and definition of software security bugs, and put forward two security indicators that can be used to predict security bugs. Software coupling in this study was used as a factor affecting different types of security bugs, such as SQL injection, DoS, and buffer overflow attacks. Castelluccio et al[14] examined patch uplift operations at Mozilla, with the aim to identify the characteristics of the uplifted patches that did not effectively fix the targeted problem that introduced regressions. The results show that uplifted patches that lead to regressions tend to have larger patch size, and most of the faults are due to semantic or memory errors in the patches. Catolino et al[15] analyzed 1,280 bug reports of 119 popular projects belonging to three ecosystems such as Mozilla, Apache, and Eclipse, with the aim of building a taxonomy of the types of reported bugs. Zaman et al[16] compared security bugs and performance bugs through a case study on the Firefox project. They found that security bugs were fixed much faster, but were reopened more than performance bugs. Furthermore, they also found that security bugs involved more developers and impact more files in a project. Sun et al[17] provided an empirical study on security bug fixing on Mozilla. In this work, we classify security bugs in five projects from more perspectives, and analyze the relationship between different categories. In summary, our work focus on the security bug characteristics to help developers better understand security bugs.

10.2 | Classification of security bugs

Classification of security bugs by previous studies[18–20] is mostly based on the root causes and the consequences of the security bugs. Piessens et al[21] classified the software bugs in terms of software development lifecycle phase, including memory safety, race condition, secure input and

output handling, faulty use of an API, improper use case handling, improper exception handling, resource leaks, and pre-processing input strings. Li et al[6] proposed a new approach for software vulnerability classification based on the complexity in vulnerability identification, fixing and exploitation, including accumulation of errors or resource consumption, strict timing requirements, complex interactions between environment and software. They also presented seven attack patterns and explored the mapping between security bug types and attack patterns. Compared with the above work, our study focus on the characteristics of security bugs, and we classify the security bugs based on the designed RQs.

11 | CONCLUSION

In this paper, we presented a comprehensive study to explore the security bug characteristics in open source projects. We investigated the categories, root causes, severity, consequences, location, and fixing duration of security bugs in five projects to study the security bug characteristics. Our study shows that (1) most of the security bugs are memory operation, authentication restriction, resource management, and security configuration security bugs; (2) the primary root causes of the security bugs are CON (Configuration Error), INP (Input Validation Error), and MEM (Memory Error); (3) the severity of most security bugs are medium and high; (4) the primary consequences that caused by the security bugs are DoS, memory corruption, and data leakage; and (5) the location where security bugs often occur on are service and web. Furthermore, we examine the relations between security bug characteristics. We also find that security bugs caused by MEM (Memory Error) often lead to memory operation and resource management security bugs. Besides, security bugs caused by INP (Input Validation Error) mainly occur on web. These results can help guide developers and testers develop the software of high security. For example, when a data leakage occurs in a program, the first thing to consider is to check the relevant code involved in authentication and data encryption.

In the future, we need to further expand the security bug dataset (such as mobile apps) to summarize our findings, making the conclusions more extensive and general. In our study process, analysis of code information is helpful and effective. However, the existing security bug dataset lacks the correlation between CVE and corresponding code information, and we will study it in the future. In addition, we will develop effective security bug detection and fixing tools according to the findings.

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (No. 61872312, No. 61972335, No. 62002309); the Yangzhou city-Yangzhou University Science and Technology Cooperation Fund Project (No. YZU201902); the Six Talent Peaks Project in Jiangsu Province (No. RJFW-053); the Jiangsu "333" Project; the Open Funds of State Key Laboratory for Novel Software Technology of Nanjing University (No. KFKT2020B15, No. KFKT2020B16); the Key Laboratory of Safety-Critical Software Ministry of Industry and Information Technology (No. NJ2020022); the Natural Science Research Project of Universities in Jiangsu Province (No. 20KJB520024); and Yangzhou University Top-level Talents Support Program (2019).

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

REFERENCES

1. Cantor AB. Sample-size calculations for Cohen's kappa. *Psychol Methods*. 1996;1(2):150-153.
2. Piantadosi V, Scalabrino S & Oliveto R Fixing of Security Vulnerabilities in Open Source Projects: A Case Study of Apache HTTP Server and Apache Tomcat. In: 12th IEEE Conference on Software Testing, Validation and Verification, ICST 2019, Xi'an, China, April 22-27, 2019. IEEE; 2019: 68-78
3. Cohen J. A coefficient of agreement for nominal scales. *Educ Psychol Meas*. 1960;20(1):37-46.
4. Warrens MJ. Cohen's kappa can always be increased and decreased by combining categories. *Stat Methodol*. 2010;7(6):673-677.
5. Landis JR, Koch GG. The measurement of observer agreement for categorical data. *Biometrics*. 1977;159-174.
6. Li X, Chang X, Board JA & Trivedi KS A novel approach for software vulnerability classification. In: 2017 Annual Reliability and Maintainability Symposium (RAMS). IEEE. ; 2017: 1-7.
7. Sharma C & Jain SC Analysis and classification of SQL injection vulnerabilities and attacks on web applications. In: International Conference on Advances in Engineering Technology Research (ICAETR - 2014); 2014: 1-6
8. Xu Z. Source Code and Binary Level Vulnerability Detection and Hot Patching. In: IEEE; 2020: 1397-1399.
9. Weiler N Honeypots for Distributed Denial of Service Attacks. In: IEEE Computer Society; 2002: 109-114
10. Xu J, Ning P, Kil C, Zhai Y & Bookholt C Automatic diagnosis and response to memory corruption vulnerabilities. In: Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, Alexandria, VA, USA, November 7-11, 2005. ACM; 2005: 223-234
11. Shinde PS & Ardhapurkar SB Cyber security analysis using vulnerability assessment and penetration testing. In: 2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare (Startup Conclave); 2016: 1-5
12. Tan L, Liu C, Li Z, Wang X, Zhou Y, Zhai C. Bug characteristics in open source software. *Empir Software Eng*. 2014;19(6):1665-1705.
13. Ayanam VS Software Security Vulnerability Vs. Software Coupling: A Study with Empirical Evidence. PhD thesis. Southern Polytechnic State University, 2009.
14. Castelluccio M, An L, Khomh F. An empirical study of patch uplift in rapid release development pipelines. *Empir Software Eng*. 2019;24(5):3008-3044.

15. Catolino G, Palomba F, Zaidman A, Ferrucci F. Not all bugs are the same: Understanding, characterizing, and classifying bug types. *J Syst Software*. 2019;152:165-181.
16. Zaman S, Adams B & Hassan AE Security versus performance bugs: a case study on firefox. In: Proceedings of the 8th working conference on mining software repositories. 2011: 93-102.
17. Sun X, Peng X, Zhang K, Liu Y, Cai Y. How security bugs are fixed and what can be improved: an empirical study with Mozilla. *Sci China Inf Sci*. 2019; 62(1):1-3.
18. Alhazmi OH, Woo S & Malaiya YK Security vulnerability categories in major software systems. In: Proceedings of the Third IASTED International Conference on Communication, Network, and Information Security, October 9-11, 2006, Cambridge, MA, USA. IASTED/ACTA Press; 2006: 138-143.
19. Weber S, Karger PA, Paradkar A. A software flaw taxonomy: aiming tools at security. *ACM SIGSOFT Software Eng Notes*. 2005;30(4):1-7.
20. Lowis L & Accorsi R On a classification approach for SOA vulnerabilities. In:. 2 of 2009 33rd Annual IEEE International Computer Software and Applications Conference. IEEE; 2009: 439-444.
21. Piessens F. A taxonomy of causes of software vulnerabilities in Internet software. In: Supplementary Proceedings of the 13th International Symposium on Software Reliability Engineering. Citeseer; 2002: 47–52.

How to cite this article: Wei Y, Sun X, Bo L, Cao S, Xia X, Li B. A comprehensive study on security bug characteristics. *J Softw Evol Proc*. 2021;e2376. https://doi.org/10.1002/smr.2376