SCIENCE CHINA
Information Sciences

• LETTER •

January 2019, Vol. 62 019102:1–019102:3
https://doi.org/10.1007/s11432-017-9459-5

How security bugs are fixed and what can be improved: an empirical study with Mozilla

Xiaobing SUN[1,2,3*], Xin PENG[1,2], Kai ZHANG[1,2], Yang LIU[4] & Yuanfang CAI[5]

[1]*School of Computer Science, Fudan University, Shanghai* 201203*, China;*
[2]*Shanghai Key Laboratory of Data Science, Fudan University, Shanghai* 201203*, China;*
[3]*School of Information Engineering, Yangzhou University, Yangzhou* 225127*, China;*
[4]*School of Computer Science and Engineering, Nanyang Technological University, Singapore* 639798*, Singapore;*
[5]*Department of Computer Science, Drexel University, Philadelphia* 19104*, USA*

Received 20 November 2017/Revised 5 February 2018/Accepted 30 March 2018/Published online 17 December 2018

Citation Sun X B, Peng X, Zhang K, et al. How security bugs are fixed and what can be improved: an empirical study with Mozilla. Sci China Inf Sci, 2019, 62(1): 019102, https://doi.org/10.1007/s11432-017-9459-5

Dear editor,

A bug is regarded as security related when it creates vulnerability in the software, which the malicious attackers could exploit to attack the system [1]. Security bugs widely exist in released software, and are serious threats to organizations, which can cause serious monetary loss and reputation damage. Therefore, addressing security bugs is a top priority in software maintenance [1].

To support developers in addressing security bugs, researchers have developed techniques for security bug identification assessment [2,3], vulnerable component prediction [4], security testing [5,6]. These studies mainly focus on detection of security bugs and how to fix them [7].

Early and quick fix of security bugs is essential for the development team [8]. In practice, however, security bugs are usually fixed in an ad-hoc way and the fixing process often takes a long time due to various reasons. It is common that security bugs are fixed in multiple commits, reopened after the initial fix, or left unresolved due to other blocking bugs. Zaman et al. [9] reported that security bugs are usually triaged and fixed the fastest, but they also found that security bugs are the most reopened and tossed compared with other types of bugs. These findings indicate that there are some factors that hinder the fixing of security bugs.

Study design. To help developers fix security bugs more quickly and easily, there is a need to learn how security bugs are fixed in practice and how it can be improved by answering the following research questions.

• **RQ1**: How is the source code revised to fix different types of security bugs? Are there common patterns for fixing security bugs?

• **RQ2**: Why do some security bugs need to be fixed in multiple commits? Are there common strategies for fixing complex security bugs that require multiple commits?

• **RQ3**: What are the causes for reopening of security bugs?

• **RQ4**: Why is the fixing of security bugs sometimes blocked by other bugs? What is the relationship between security bugs and their blocking bugs?

To answer these questions, we conducted an empirical study on bug fixing practices in Mozilla. We identified 1609 bugs between July 2005 and Aug 2015 that were marked as security bugs. We reviewed the descriptions of each security bug and identified 252 security bugs that were unanimously confirmed to fall within the scope of our study. We analyzed the bug reports, comments, and commits of these bugs to understand the process for fixing them and the corresponding code changes to answer the research questions.

Study results. Security bug fixing patterns. We

* Corresponding author (email: xbsun@yzu.edu.cn)

manually analyzed change history of security bug fixing in Mozilla and identified ten different local fixing patterns. We found that when fixing security bugs, most patches were local and small. In addition, the security bugs could be classified into three groups — data preprocessing, data processing, and data ensuring — according to the data computation process in a code module, as shown in Figure 1. Data preprocessing is usually used to prevent data from external attacks, which is generally set in the entrance of a code module. There are four fixing patterns at the data preprocessing phase: checking data (CKD), filtering illegal data (FID), strengthening data constraint (SDC), and weakening data constraint (WDC). Data processing is used to guarantee that the data is processed in a secure way. There are five fixing patterns at the data processing phase: adding code branch (ACB), changing the type of an object to its parent class (COP), changing the type of an object to its sub class (COS), adding tryCatch module (ATC), and changing the weak reference to strong reference (CWS). Data ensuring is used to guarantee the validity and legality of the code module before transmission to another code module(s). The primary fixing pattern also includes the checking data (CKD-E) pattern similar to that of CKD at the data preprocessing phase.

Process of fixing security bugs. Some security bugs are resolved in multiple commits. Of the 252 security bugs examined in our study, 138 (55%) security bugs required multiple commits.

For security bugs fixed in multiple commits, we summarize their process into five strategies. The first strategy is a widely used fixing process (Strategy i), i.e., developers first fix a security bug and commit it, then they may verify whether their patch is correct. So they test their fixing code and commit again. Since fixing security bugs is usually emergent, developers may employ a temporary patch to fix the security bug. When he/she has enough time, a better patch is used to fix the original security bug (Strategy ii). In addition, there are also some security bugs, which are difficult to fix in a single commit. At this time, developers may intentionally divide the fixing process into multiple commits (Strategy iii). There are also some security bugs that are not correctly fixed in their initial commit. Then developers need to fix the original security bug again (Strategy iv). To improve the quality of fixing security bugs, developers sometimes refactor their previous fixing, which is another typical fixing strategy in multiple commits (Strategy v).

Reopening of security bugs. During the fixing process of security bugs, some may be not fully resolved. Then these security bugs are reopened and fixed again. There are totally 68 reopened security bugs in our study.

First, some security bugs are reopened because they are not completely fixed in their original patch(es). Therefore, this type of bug reopening is used to fix the problem related to their original fixing. The relation between the fixing after reopening and the original fixing includes coordinate relation (Type i) and liner relation (Type ii). Coordinate relation is used to indicate that the fixing patterns, functionalities, and code after a bug reopening are different from their original patch(es), but they focus on the same security bug. For liner relation, it is used to indicate that the fixing patterns and functionalities are similar to their original patch(es), but the position of code patches are different.

There are also some patches for reopened bugs to enhance the fixing of the original security bugs. Two ways are used to enhance fixing of original security bugs, i.e., being wider (Type iii) and being stronger (Type iv). When a security bug is fixed in a wider way after its reopening, it indicates that the protection scope of the original patch is extended. When a security bug is fixed in a stronger way after its reopening, it indicates that the original patch is not enough, in this way, another more secure patch is required. The main difference between being wider and being stronger is that ways of fixing after the reopening and the original fixing are similar for being wider fixing, but different for being stronger fixing.

To fix security bugs, sometimes the patch is too strong, which may affect the performance or functionality of the system. So some security bugs are reopened to relax the patch(es) of the original fixing (Type v).

There are also some security bugs that are not correctly revised, which needs to be reopened for a correct fixing (Type vi). In our study, we also found that there are some patches without any changes on the code after reopening of a security bug. For these security bug reopening, developers just added some tests (Type vii) or checked the original fixing (Type viii).

Blocking for security bugs. One reason that a security bug remains unresolved is that blocking bugs prevent the developers from fixing it. Hence, developers should first fix address blocking bugs. These blocking bugs include both non-security and security related bugs.

Based on the investigation of the 252 security bugs in Mozilla, we found that there were 88 security bugs that have 88 non-security blocking bugs and 15 security-related blocking bugs. For these

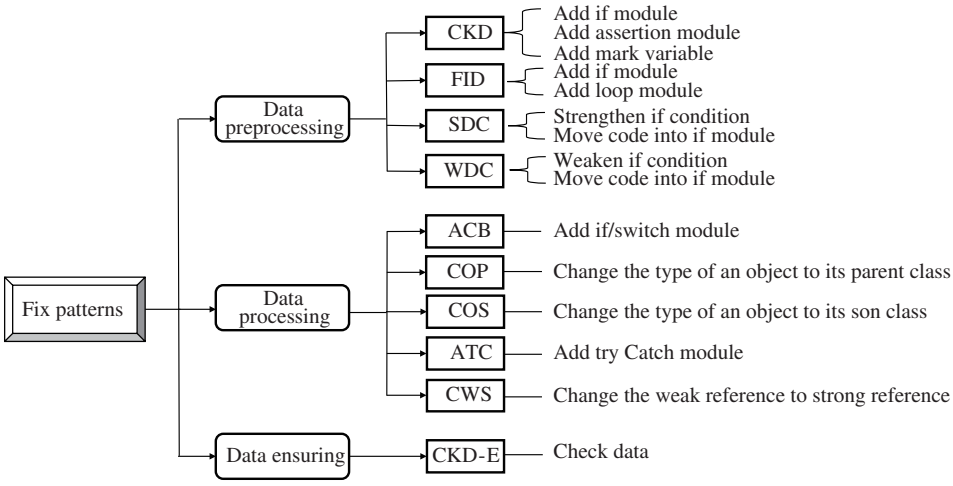

Figure 1 Common fixing patterns of security bugs.

two different blocking bugs, we identify different blocking relations. These blocking bugs are divided into three types, In-functionality blocking (Blocking i), Cross-functionality blocking (Blocking ii), and Platform blocking (Blocking iii). In-functionality blocking represents the blocking bugs within a functionality. When fixing a security bug, if the fixing patches depend on other components within the same functionality, which includes non-security or security bugs, these bugs become the in-functionality blocking bugs. Cross-functionality blocking indicates the blocking between different functionalities, which occurs only for non-security bugs. For platform blocking, it means that there is a bug in the platform that may prevent its associated components (with security bugs) from being fixed. The platform blocking includes both non-security related bugs and security related bugs.

Possible improvements in security bug fixing. Our findings from the study suggest possible improvements for security bug fixing, including:

• developing automatic program repair techniques or tools for security bug fixing considering the specific fixing patterns;

• developing security-related regression testing and fixing solution recommendation techniques or tools to decrease the cost of fixing security bugs;

• using program analysis or clone detection tools to improve the patch quality of security bugs;

• developing techniques or tools for predicting blocking bugs considering different types of blocking relations.

Acknowledgements This work was supported partially by Natural Science Foundation of China (Grant Nos. 61872312, 61402396, 61611540347, 61472344), Jiangsu Qin Lan Project, China Postdoctoral Science Foundation (Grant No. 2015M571489), and Natural Science Foundation of Yangzhou City (Grant No. YZ2017113).

References

1 Viega J, McGraw G. Building Secure Software: How to Avoid Security Problems the Right Way. 1st ed. London: Addison-Wesley, 2011
2 Cai Y, Jia C, Wu S, et al. ASN: a dynamic barrier-based approach to confirmation of deadlocks from warnings for large-scale multithreaded programs. IEEE Trans Parallel Distrib Syst, 2015, 26: 13–23
3 Cai Y, Chan W K. Magiclock: scalable detection of potential deadlocks in large-scale multithreaded programs. IEEE Trans Softw Eng, 2014, 40: 266–281
4 Shar L K, Tan H B K, Briand L C. Mining SQL injection and cross site scripting vulnerabilities using hybrid program analysis. In: Proceedings of the 35th International Conference on Software Engineering, San Francisco, 2013. 642–651
5 Felderer M, Büchler M, Johns M, et al. Chapter one - security testing: a survey. Adv Comput, 2016, 101: 1–51
6 Cai Y, Lu Q. Dynamic testing for deadlocks via constraints. IEEE Trans Softw Eng, 2016, 42: 825–842
7 Cai Y, Cao L. Fixing deadlocks via lock pre-acquisitions. In: Proceedings of the 38th International Conference on Software Engineering, Austin, 2016. 1109–1120
8 Wang L, Sun X, Wang J, et al. Construct bug knowledge graph for bug resolution: poster. In: Proceedings of IEEE/ACM International Conference on Software Engineering, 2017. 189–191
9 Zaman S, Adams B, Hassan A E. Security versus performance bugs: a case study on firefox. In: Proceedings of the 8th Working Conference on Mining Software Repositories, New York, 2011. 93–102