Peer Code Review to Prevent Security Vulnerabilities: An Empirical Evaluation

Amiangshu Bosu and Jeffrey C. Carver
University of Alabama, Tuscaloosa, AL, USA
asbosu@ua.edu, carver@cs.ua.edu

***Abstract*—Peer code review, as an effective quality improvement practice, has also been considered important for reducing security vulnerabilities. There is a lack of empirical evidence to quantify and support this claim. Therefore, we propose a research plan to analyze mature open source projects to gather empirical evidence regarding the relationship between peer code review and security vulnerabilities. As a proof-of-concept,we analyzed the Chromium OS project and found that reviewers identified potential vulnerabilities in 32 review requests.**

***Index Terms*—code review; security defects; open source; vulnerability**

I. Introduction

Software inspections, in which developers subject their code to review by peers or other stakeholders to identify defects, are an effective quality improvement practice [1]. Specifically in the security context, McGraw suggests that peer code review is an important practice for detecting and correcting security bugs [2]. For example, expert reviewers can identify code that contains potential security vulnerabilities and help the author eliminate the security flaws or abandon the vulnerable code. Moreover, peer code review can identify attempts to insert malicious code into the codebase. According to McGraw, the longer it takes to detect and fix a security vulnerability the higher the overall cost associated with that vulnerability [3]. Therefore, peer code review can reduce the cost of creating secure software by helping developers eliminate security defects earlier when it is less expensive.

Even with the benefits offered by peer code reviews, their relatively high cost and time requirements have reduced the prevalence with which software teams, in general, adopt them. Conversely, many mature, successful Open Source Software (OSS) communities have recently adopted peer code reviews as an important quality assurance mechanism. Despite widespread adoption in OSS communities, there has been little empirical research to quantify the level usage and the specific benefits provided by peer code review [4], especially as it relates to reducing security vulnerabilities. To advance the field of software security engineering, we must move from anecdotal evidence towards more objective empirical evidence.

This paper describes a research plan to empirically evaluate the impact of peer code review on security vulnerabilities in OSS communities. The paper also provides some initial results as a proof of concept for the proposed approach.

II. Security Vulnerabilities

Security vulnerabilities are specific source code flaws that allow attackers to expose, alter, disrupt, or destroy sensitive information [5]. Even though there are numerous examples of exploited security flaws that resulted in significant financial losses or other serious problems, users still report numerous security vulnerabilities to publicly available vulnerability databases (e.g. Common Vulnerabilities and Exposures - CVE). Most of those reported vulnerabilities belong to one of the following types [6]: i) Buffer Overflow (BOF), ii) Format String (FS), iii) Integer Overflow (IOB), iv) SQL Injection (SQLI), v) Improper Access Control (IAC), vi) Cross Site Scripting (XSS), vii) Command Injection (CMDI), viii) Denial of Service / Race condition (DOS), and ix) Cross Site Request Forgery (XSRF).

Each type of vulnerability has some common characteristics that may be identified via peer code review. For example, BOF bugs are often associated with unsafe string-handling function calls (e.g. *strcpy, strcat, sprintf, and gets*) or memory accesses using array indices. FS bugs are associated with using *printf()* to output user input without proper filtering or formatting. Therefore, when an experienced reviewer examines code that contains risky function calls or memory accesses, s/he may be able to able to identify potential security vulnerabilities and suggest alternate safer functions. Howard also provided some ideas about how to perform an effective code review to detect different types of security flaws [7].

III. Peer Code Review in OSS Communities

Many OSS communities have made peer code review a mandatory practice. For example, according to the Apache server coding policy, at least three core project members should review a code change before including it in the main trunk [8]. Many projects (e.g. Apache HTTP server and FreeBSD) use mailing-lists to facilitate code reviews. In this case, a developer emails a patch to the mailing-list with special designated keywords in subject line. Some projects (e.g. Mozilla foundation projects) have integrated the code review process into the bug tracker. Due to an increased emphasis on peer code review, many OSS projects have started using dedicated code review tools: ReviewBoard [1], Gerrit [2], and RietVeld, [3].

[1]http://www.reviewboard.org/
[2]http://code.google.com/p/gerrit/
[3]http://code.google.com/p/rietveld/

IV. Research Method

During an earlier analysis of the peer code review process in OSS communities [4], we observed that code reviewers often provide detailed comments and suggestions about potential bugs. Therefore, we can find indications of security flaws by text-mining the code review comments. For example, if a reviewer suspects the presence of a potential buffer overflow, his/her review comments will most probably contain either *buffer*, *overflow* or both. For each of the common vulnerabilities types, Table I shows the list of keywords we plan to use when mining the review comments.

To support this mining process, we developed a Java application mines the ReviewBoard or Gerritt code review systems for OSS projects and inserts the extracted information into a MySQL database. We will query the database to create a list of the code review requests that contain the specific keywords of interest. We will manually inspect each review request and its associated patches to determine whether any security vulnerabilities were present. Using this method we expect to answer following research questions:

1) How many vulnerabilities were prevented by the peer code-review process?
2) Which type(s) of vulnerabilities were detected by the peer code-review process?
3) What were the outcomes of the peer code-review suggestions (e.g. abandoned or modified)?
4) Which types of reviewers (in terms of background or expertise) were most likely to detect vulnerabilities?
5) Which type of author (in terms of background or expertise) was most likely to introduce vulnerabilities?

V. Preliminary Motivating Results

As an initial proof-of-concept, we analyzed the code review requests stored in Gerrit for the Chromium OS project [4], a Linux-based operating system written primary in C, C++, and Python. In this project, all code changes must be approved by reviewer(s) before being merged into the trunk. Of the 40,807 review requests posted between February 2011 and January 2013, we identified 258 as potentially relevant to security vulnerabilities. A manual inspection of those requests indicated that reviewers raised concerns about potential vulnerabilities in 62 of those requests. Further analysis showed that 32 of those review requests addressed one or more types of security vulnerabilities. Table II illustrates the different types of vulnerabilities that were addressed in these requests. Due to the nature of the project, written mostly in C or C++, there were a many BOF, FS, and IOF vulnerabilities but no SQLI, XSS, and XSRF. Also, as it is an operating system, there were also a number of IAC and DOS vulnerabilities. In our future work when we analyze a web application project (e.g. MediaWiki), we expect to find evidence of potential SQLI, XSS, and XSRF vulnerabilities identified during code review. These initial results suggest that peer code review is an effective approach for identifying different types of potential vulnerabilities and motivates further study.

TABLE I
Keywords Associated with Security Flaws

Vulnerability Type	Keywords
Buffer overflow	buffer, overflow
Format string	format, string
Integer overflow	Integer, overflow
Cross site scripting	cross-site, cross, CSS, XSS
SQL injection	SQL, injection
Command injection	command, injection
Improper access control	improper, unauthenticated, access
Denial of Service/Race	denial service, DOS, race
Cross site request forgery	cross site, request forgery, CSRF, XSRF
Common	security, vulnerability, vulnerable, hole

[4]http://www.chromium.org/chromium-os

TABLE II
Security Vulnerabilities Eliminated

Vulnerability Type	# of Vulnerable Requests	# of Changes Fixed	# of Changes Abandoned
BOF	6	2	4
FS	5	4	1
IOB	8	8	0
XSS	0	0	0
SQLI	0	0	0
CMDI	0	0	0
IAC	8	2	6
DOS	8	5	3
XSRF	0	0	0

VI. Conclusion

This paper describes the need for an empirical analysis of the relationship between peer code review and security vulnerabilities. We described a methodology for extracting code review data and analyzing it to identify reviews related to security vulnerabilities. Our proof-of-concept analysis indicated that in one particular project, the code review process did identify and remove a number of security vulnerabilities. In the future, we plan to conduct a more thorough analysis of the data from multiple OSS communities to better quantify the effects of code review on security vulnerabilities.

Acknowledgment

This research is supported by the NC State Science of Security lablet.

References

[1] M. E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal*, vol. 15, no. 3, pp. 182 –211, 1976.

[2] G. McGraw, "Software security," *IEEE Security Privacy*, vol. 2, no. 2, pp. 80 – 83, Mar-Apr 2004.

[3] ——, "Automated code review tools for security," *Computer*, vol. 41, no. 12, pp. 108 –111, Dec. 2008.

[4] A. Bosu and J. C. Carver, "Peer code review in open source communitiesusing reviewboard," in *Proc. of the 4th ACM Wksp. on Evaluation and Usability of Prog. Lang. and Tools*, 2012, pp. 17–24.

[5] M. Dowd, J. McDonald, and J. Schuh, *The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities*. Addison-Wesley Professional, 2006.

[6] K. Tsipenyuk, B. Chess, and G. McGraw, "Seven pernicious kingdoms: a taxonomy of software security errors," *IEEE Security Privacy*, vol. 3, no. 6, pp. 81 – 84, Nov.-Dec. 2005.

[7] M. Howard, "A process for performing security code reviews," *IEEE Security Privacy*, vol. 4, no. 4, pp. 74 –79, July-Aug. 2006.

[8] P. C. Rigby, D. M. German, and M.-A. Storey, "Open source software peer review practices: a case study of the apache server," in *Proc. of the 30th Int'l Conf. on Soft. Engg.*, ser. ICSE '08, 2008, pp. 541–550.