Characteristics of the Vulnerable Code Changes Identified through Peer Code Review

Amiangshu Bosu
Department of Computer Science
University of Alabama
Tuscaloosa, AL USA
asbosu@ua.edu

ABSTRACT

To effectively utilize the efforts of scarce security experts, this study aims to provide empirical evidence about the characteristics of security vulnerabilities. Using a three-stage, manual analysis of peer code review data from 10 popular Open Source Software (OSS) projects, this study identified 413 potentially vulnerable code changes (VCC). Some key results include: 1) the most experienced contributors authored the majority of the VCCs, 2) while less experienced authors wrote fewer VCCs, their code changes were 1.5 to 8 times more likely to be vulnerable, 3) employees of organizations sponsoring OSS projects are more likely to write VCCs.

Categories and Subject Descriptors

K.6.3 [**Management of Computing and Information Systems**]: Software Management—*Software development, Software process*

General Terms

Measurement, Security, Human Factors

Keywords

code review, security defects, open source, vulnerability, inspection

1. PROBLEM AND MOTIVATION

Security vulnerabilities are critical software flaws that introduce the potential for severe damage. We define ***Vulnerable Code Changes (VCC)*** as “code changes committed to the repository that contain potentially exploitable vulnerabilities.” Developers can reduce the potential for security problems by identifying VCCs and either fixing them or removing them. Identifying VCCs is effort-intensive and often requires specially trained security experts. Unfortunately, these security experts are scarce [10]. Therefore, there is a need to identify which code changes are most in need of examination by these experts. The goal of this research is to develop an empirical method for identifying these VCCs so the valuable time of the security experts can be used most effectively. Using peer code review data from 10 popular Open Source Software projects, we provide empirical evidence about: 1) the characteristics common to VCCs, 2) the locations where VCCs are more likely to occur, 3) the characteristics of authors that are likely to contribute VCCs, and 4) the characteristics of code reviewers that are likely to identify VCCs.

2. RELATED WORK

There have been several attempts to predict the characteristics and locations of vulnerabilities based on code attributes and developer characteristics. Relative to code attributes, researchers have built models using header file inclusions and function call relationships [15], code complexity metrics [18, 20], lines of code, alert density, and code churn information [8], and the presence of non-security defects [7]. Relative to developer characteristics, researchers have found the presence of vulnerabilities correlated with: the number of developers changing a file [12], developer network characteristics [17], and changes from ‘new effective author’ (i.e. first time author for a file) [11]. Most of the prior studies focused on code and file metrics, while there are a small number of studies that consider human aspects (i.e., the type of people are most likely to create or identify vulnerabilities). This study analyzes three aspects (i.e. human aspects, the characteristics of the VCCs, and VCC locations) together using same dataset.

3. APPROACH & UNIQUENESS

Peer code review data is useful for identifying VCCs because the discussion between the developer and the reviewer, documented in the review, can indicate the presence of a VCC. To support this argument, my prior study [3] showed that reviewers often provide detailed comments and suggestions about potential bugs. Therefore, we hypothesized that a targeted keyword search of code review comments can identify VCCs. For example, if a reviewer suspects the presence of a potential buffer overflow, his/her review comments will likely contain either *buffer*, *overflow* or both. The

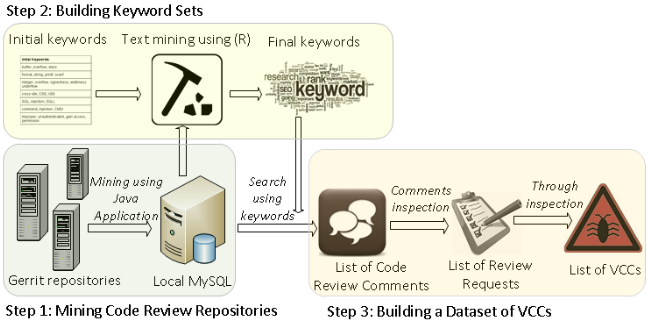

Figure 1: Research Methodology

ICSE ’14 Hyderabad, India
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$15.00.

Table 1: Keywords Associated with Vulnerabilities

Vulnerability Type	Keywords
Buffer Overflow	buffer, overflow, stack
Format String	format, string, printf, scanf
Integer Overflow	integer, overflow, signedness, widthness, underflow
Cross Site Scripting	cross site, CSS, XSS, htmlspecialchar (PHP only)
SQL Injection	SQL, SQLI, injection
Race Condition / Deadlock	race, racy, deadlock
Improper Access Control	improper, unauthenticated, gain access, permission
Denial of Service / Crash	denial service, DOS, race
Cross Site Request Forgery	cross site, request forgery, CSRF, XSRF, forged
Common	security, vulnerability, vulnerable, hole, exploit, attack, bypass, backdoor, crash
Common (added later)	threat, expose, breach, violate, fatal, blacklist, overrun, insecure

results of my preliminary study [4] verified the applicability of this approach. This approach is unique in two key ways: 1) it focuses on the developers' experience writing and identifying vulnerabilities, and 2) it uses data from publicly available open-source peer code review repositories, rather than from code or bug repositories, to build the database of potential VCCs. Figure 1 provides an overview of the 3-step research method. Four researchers, who are listed in the acknowledgement section, assisted me in this work.

Step 1: I developed a Java Application, *Gerrit-Miner*, to mine the publicly accessible data for each review request posted in Gerrit [1] code review repositories and insert that data into a MySQL database using a similar approach as Mukadam et al. [14].

Step 2: To ensure the selection of a proper set of keywords, my colleagues and I used an empirical approach to develop the keyword set (table 1) related to the ten most common vulnerability types according to previous studies [2, 19] and Open Web Application Security Project (OWASP) [16]. We created and refined the initial set of keywords based on our own knowledge. Using text-mining techniques based on prior studies [9] and the R - text mining (tm) package [6], we determined the frequently co-occurring keywords with each of our initial keywords. We manually inspected the list of frequently co-occurring keywords to determine whether we should add them to the final keywords list.

Step 3: We divided this step into three phases based on the study selection procedure in systematic literature reviews [5]. Table 2 provides the number of review requests inspected in those steps for the ten projects. In the first phase, *database search*, we queried the code-review database (created in Step 1) for each projects to identify the code review requests with comments that contained at least one the keywords identified in Step 2.

In the second phase, *comments inspection*, two researchers independently inspected the comments of each code review from the *database search* to identify the reviews that were clearly not related to VCCs. We excluded a review request only if both inspectors independently determined it was not related to a VCC.

In the third phase, *thorough inspection*, we thoroughly inspected the code review discussions and associated patches for each remaining review request. Two researchers independently inspected the review request in detail to determine whether the reviewer identified a VCC and classified the vulnerability based on Common Weakness Enumeration Specification (CWE) [13]. We considered a code change vulnerable only if: a) a reviewer raised concern about potential vulnerabilities, b) our manual inspection of the associated code segment indicated the presence of a vulnerability, and c) the code author either explicitly acknowledged the vulnerability in the review comments or implicitly acknowledged the vulnerability by making the recommended changes in the subsequent patches.

Table 2: Overview of the research steps

Project	Requests mined	Comments inspected	Thoroughly inspected	# VCCs identified
Android	18710	741	205	60
Chromium OS	47392	4148	442	139
Gerrit	4207	190	26	14
ITK/VTK	10549	344	34	14
MediaWiki	59738	930	89	37
OmapZoom	31458	1506	180	45
OpenAFS	9369	267	79	22
oVirt	13647	467	118	35
Qt	53303	1985	131	30
Typo3	18673	465	44	17
Total	**267046**	**11043**	**1348**	**413**

4. RESULTS AND CONTRIBUTIONS

Using the approach in Section 3, we identified 413 VCCs across the 10 projects. Following are the key results of our study.

Characteristics of the Vulnerabilities: Peer code review helps identify most of the common vulnerability types. While most vulnerability types are frequently fixed, some critical vulnerabilities (e.g. improper access, and deadlock) had high abandonment rate.

Characteristics of the Vulnerable Changes: Similar to prior studies, the probability that a patch-set contains a VCC increases with the number of lines changed [11] and modified files are more likely than new files to contain vulnerabilities[12, 17].

Characteristics of the VCC Authors: The majority of the VCCs are introduced by the most experienced authors (i.e. those making the most changes). However, less experienced authors are 1.5 to 8 times more likely to write VCCs. Moreover, employees of organization sponsoring the OSS project are more likely to write VCCs.

Characteristics of the VCC reviewers: The reviewers identifying VCCs are the most experienced reviewers and have more coding experience than the authors writing it.

The contributions of this study include, 1) a method for building an empirically validated set of keywords for mining security defects, 2) a methodology for systematically reducing the number of data items that require manual analysis, thereby allowing similar types of studies to scale to larger problem sizes, and 3) use of peer code review data to examine vulnerabilities at the point when they are be introduced and to examine characteristics of the authors and reviewers of VCCs.

In addition to confirming some previous results, we identified a surprising insight about the characteristics of VCC authors. The projects in this study are popular, successful OSS projects that attract a lot of talented developers. Even so, the most experienced developers still contribute VCCs, which supports McGraw's claim that developers lack knowledge about secure coding practice [10]. We believe the insights from this study can help OSS projects reduce the introduction of and improve the identification of the VCCs.

Acknowledgments

I would like to thank Jeffrey Carver, Munawar Hafiz, Patrick Hilley, and Derek Janni for assistance with this research. This research is partially supported by the NC State Science of Security lablet, and NSF-1156563.

5. REFERENCES

[1] Gerrit code review tool. `https://code.google.com/p/gerrit/`. [Online; accessed 6-Sep-2013].

[2] A. Austin and L. Williams. One technique is not enough: A comparison of vulnerability discovery techniques. In *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on*, pages 97–106. IEEE, 2011.

[3] A. Bosu and J. C. Carver. Peer code review in open source communities using reviewboard. In *Proeedings of the 4th ACM Wksp. on Evaluation and Usability of Programming Language and Tools*, pages 17–24, New York, NY, USA, 2012. ACM.

[4] A. Bosu and J. C. Carver. Peer code review to prevent security vulnerabilities: An empirical evaluation. In *Software Security and Reliability-Companion (SERE-C), 2013 IEEE 7th International Conference on*, pages 229–230, 2013.

[5] T. Dyba, T. Dingsoyr, and G. K. Hanssen. Applying systematic reviews to diverse study types: An experience report. In *First International Symposium on Empirical Software Engineering and Measurement, 2007. ESEM 2007.*, pages 225–234. IEEE, 2007.

[6] I. Feinerer. Introduction to the tm package text mining in r. `http://cran.r-project.org/web/packages/tm/index.html`, 2013.

[7] M. Gegick, P. Rotella, and L. Williams. Toward non-security failures as a predictor of security faults and failures. In *Engineering Secure Software and Systems*, pages 135–149. Springer, 2009.

[8] M. Gegick, L. Williams, J. Osborne, and M. Vouk. Prioritizing software security fortification through code-level metrics. In *Proceedings of the 4th ACM workshop on Quality of protection*, pages 31–38. ACM, 2008.

[9] S. Lukins, N. Kraft, and L. Etzkorn. Source code retrieval for bug localization using latent dirichlet allocation. In *15th Working Conference on Reverse Engineering, 2008. WCRE '08.*, pages 155–164, 2008.

[10] G. McGraw. *Software security: building security in*, volume 1. Addison-Wesley Professional, 2006.

[11] A. Meneely, H. Srinivasan, A. Musa, A. R. Tejeda, M. Mokary, and B. Spates. When a patch goes bad: Exploring the properties of vulnerability-contributing commits. page to appear, 2013.

[12] A. Meneely and L. Williams. Secure open source collaboration: an empirical study of linus' law. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 453–462. ACM, 2009.

[13] Mitre Coroporation. Common weakness enumeration. `http://cwe.mitre.org/`. [Online; accessed 6-Sep-2013].

[14] M. Mukadam, C. Bird, and P. C. Rigby. Gerrit software code review data from android. In *Proceedings of the Tenth International Workshop on Mining Software Repositories*, pages 45–48. IEEE Press, 2013.

[15] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller. Predicting vulnerable software components. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 529–540. ACM, 2007.

[16] OWASP. The open web application security project. `https://www.owasp.org/index.php/Category:Vulnerability`, 2013. [Online; accessed 1-Sep-2013].

[17] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. *Software Engineering, IEEE Transactions on*, 37(6):772–787, 2011.

[18] Y. Shin and L. Williams. An empirical model to predict security vulnerabilities using code complexity metrics. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 315–317. ACM, 2008.

[19] K. Tsipenyuk, B. Chess, and G. McGraw. Seven pernicious kingdoms: a taxonomy of software security errors. *IEEE Security Privacy*, 3(6):81 – 84, Nov.-Dec. 2005.

[20] J. Walden, M. Doyle, G. A. Welch, and M. Whelan. Security of open source web applications. In *Proceedings of the 2009 3rd international Symposium on Empirical Software Engineering and Measurement*, pages 545–553. IEEE Computer Society, 2009.