Many Eyes Make All Vulnerabilities Shallow? Mining Code Inspection Logs and Vulnerability Records - Final Report

CREU Final Reports 2015

Student Researchers: Kayla Davis Danielle Neuberger Rachel Catipovic Alvaro Pareja-Lecaros

Advisor: Andy Meneely

Institution: Department of Software Engineering Rochester Institute of Technology

Webpage: http://www.se.rit.edu/~archeology/

Goals of the Project

Eric Raymond, in his famous essay The Cathedral and the Bazaar, coined Linus’ Law as: “Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix will be obvious to someone […] given enough eyeballs, all bugs are shallow.” This mantra has become a cornerstone of open source software development and is often the basis for claiming that open source code is less likely to have vulnerabilities. In the context of security, this law becomes critically important because finding vulnerabilities often involves significant effort, an objective viewpoint and expertise in both security and the project itself. The empirical software engineering community has only begun to examine the “many eyes” aspect of Linus’ Law. Empirically, we have found in past years that vulnerabilities exist in places where more people have provided feedback - ostensibly contradicting Linus’ Law. However, the “fix is obvious to someone” phrase indicates that the experience of the developer is also key. In this study, we are examining the following areas related to developer experience. Specifically, are source code files less likely to have a vulnerability if developers have any of the following experiences?

• build sheriffs, i.e. more experienced in maintaining the build system
• code owners, i.e. given explicit authority to approve or reject changes
• worked on compatibility, stability, and regression bugs
• security-adjacent, i.e. worked with people who have worked on fixes to vulnerabilities
• bystanders, i.e. been on reviews with three or more invited reviewers (diffusion of responsibility)

In this project, we have collected and developed scripts to analyze code review data in the Chromium project to answer questions about how developers can find and prevent vulnerabilities in their source code.

Process

The process used for this project was an iterative, milestone-based process. Students collaborated on a Github account with issues being assigned and milestones being defined. Weekly meetings were complemented by many
“work meetings” with a subset of the members to work on technical issues and to brainstorm new ideas. The overall steps of the process were:

1. Project familiarization
2. Develop a data collection strategy
3. Write scripts to collect, parse, and query data
4. Model co-developers and security-experienced developers
5. Analysis wrap-up, generate follow-up research questions

During Step 3, the team built a continuous integration server to aggregate and analyze the data each night. The nightly builds run between 12 and 16 hours each day and provide a way for us to gain regular feedback on our queries and data verification runs on very large data set.

**Conclusions and Results**

We have come to a wide variety of results this year and have published one workshop paper and one conference paper already. We are working on a journal paper that compares results across categories. The results are discussed briefly here.

**Code Ownership.** At a broad level, files with large numbers of owners are more likely to later have vulnerabilities. This result is consistent with the notion that, with many people owning a given file, they are unable to coordinate properly with each other and thus mistakes are missed. These code ownership records, however, don’t line up perfectly with developers making large contributions, either. In the literature, code ownership in open source is often dependent upon making many commits over time; however, Chromium has a large contingent of developers who are owners but not heavy contributors. This dynamic makes the metrics a more novel contribution to the literature.

**Build Sheriffs.** In Chromium, some developers are asked to be “on call” for when the build fails. Since a new patch “lands” on the product every half hour or so, developers must watch for potential integration problems and track down the right people when a problem does arise. These “sheriffs”, as a result, build up a larger social network and become more familiar with parts of the system they would otherwise not normally encounter. When examining the contributions to a source code file, files that have more contributions from build sheriffs (especially when weighted against how many hours those sheriffs have put in) are several times more likely to be vulnerable.

Our other results also show statistically significant correlations in the expected directions regarding developers who work with security-experienced developers. Developers who work on compatibility bugs are also less likely to later be working on vulnerable code. All of these factors are part of what leads to a vulnerable file.

Our overall message is that socio-technical factors play a significant role in vulnerabilities in aggregate but require many different measurements to fully capture the complex development processes that go into producing a product like Chromium.

**Presentations and Publications**
