# Do Many Eyes Make All Vulnerabilities Shallow?

## Mining Code Inspection Logs and Vulnerability Records - Final Report

CREU Final Reports 2015


| | |
|---|---|
| **Contributors:** | **Samantha Oxley**<br>**Kayla Nussbaum**<br>**Nathan Evans** |
| **Project Lead:** | **Dr. Andy Meneely** |
| **Institution:** | **Rochester Institute of Technology** |
| **Domain:** | **Chromium Research** |


---

____

### Goals of the Project

There is a commonly repeated rule in the open source development community called Linus' Law, it says "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 idea has impacted the open source community heavily and is frequently the basis for claiming open source software is less likely to have vulnerabilities. When applied to the realm of software security this law becomes critically important because identifying vulnerabilities in a large software project often requires significant effort, objectivity, and domain knowledge in both security practices and the software project. Empirically, the "many eyes" aspect of Linus' Law has been shown to be more nuanced, though an investigation of this principle has only just begun. However, the "fix is obvious to someone" phrase implies that the experience of the developer is also key. In this study we continue last year's CREU work to further examine Chromium's developer network, to determine if aspects of a developer's experience can impact the likelihood of vulnerabilities present in a code review. We will observe the following about a developer's experience in a year of development:

- Degree, i.e. more connections to other developers through participation in code reviews
- Participations, i.e. more code review participations
- Ownership_count, i.e. more code review ownership
- Closeness, i.e. more closely participatory with the larger developer network
- Betweenness, i.e. frequently participating between larger development groups
- Sheriff Hours, i.e more experienced in tracking problems discovered by the build system
- Vulnerability Misses, i.e. more participations on code reviews which have one or more files that are later fixed for a vulnerability
- Vulnerability Fixes Owned, i.e. more ownership of vulnerability fixing code reviews
- Vulnerability Fixes, i.e. more participation in vulnerability fixing code reviews

**Purpose of the Project**

The research, about vulnerabilities in large software projects, conducted during this year is an extension of several years of work performed by Dr.Meneely. The purpose of this research has been to shed some light on the overarching question, “Do many eyes make all vulnerabilities shallow?”, by exploring smaller, related research questions. Specifically, we used this year to examine how developers are connected to each other by conducting social network analysis on a graph of collaboration in code reviews. In doing so, students have found trends and developed metrics which are of value for anyone doing research in this area.

**Process**

The process used for this project was iterative and milestone-based. Students developed research questions which they submitted as issues on GitHub. Students and the Project Lead would attend weekly meetings to check-in and report their progress for their particular issue. These meetings were supplemented by “work meetings”, where the students came together to work on answering their research questions. The group’s progress was recorded in weekly wiki entries, and in the aforementioned GitHub issues. Generally, the steps of the process were:

1. Get familiar with the project and data already derived
2. Develop a research question that the data can provide an answer to
3. Write scripts to collect, parse, query, and extrapolate data
4. Examine trends to see what the data says in response to the posed question
5. Analysis wrap-up, generate follow-up research question

Step 1 is to give students a chance to become familiar with the tools used in the project, as well as the different data that has already been collected. This is both so they are comfortable working with the code base, and to give them a chance to figure out what kinds of research questions they may want to pose to the data that is currently present. Generally, after the first iteration, students would start the next iteration at Step 2. During Step 2, the idea is to develop a research question that would yield information relevant to the overarching, while also giving the student the chance to express and challenge their self in a research capacity. During Step 3, the team added code to a continuous integration server to aggregate and analyze data each night. The nightly build on this server allowed for regular feedback on queries and data verification of our large data set. During Step 4, the team would analyze the data to see what kinds of answers it supports in regards to the questions that have been posed. Sometimes questions are met with obvious conclusions, sometimes questions that we thought were resolved had to be revisited. During Step 5, students would come up with a new research question, typically related to their previous one, that they would carry through a new iteration of the process explained above.

Our research analyzes the correlations between Chromium's developers and how interconnected their vulnerabilities are with the rest of the files in their network. Few of the metrics we study involve how strong a developer's participation is based on their involvement, and how central their contributions are to other developers. We base many of our conclusions on the data we receive from running correlations against vulnerabilities missed and other developer metrics.

**Conclusions and Results**

The research conducted by this group produced results that directly confirmed or denied the hypotheses we made about our metrics and GitHub issues. One of our main accomplishments was to create a table of developers and their data during a one-year period of time. The metrics from this table are important because they provide us a lot of insight into Chromium's commit and review system, not just about developers individually, but about groups of developers. While we were analyzing metrics from a programming perspective, we decided to perform manual investigation to understand exactly how certain developers were recognized in our schema. We ran Spearman Correlations on these developer metrics to discover new things about the developers and how their contributions affect the Chromium community.

Through our investigation and metric correlations, we have made the conclusion that degree of participation in the Chromium community is not associated with the amount of sheriff hours, nor the centrality of a developer. That is, historically, developers do not need to be extremely participatory to be well-connected in the community.

We have also shown via correlations in our data that a developer will be more likely to miss a vulnerability if they had substantial participation on a code review. Through further manual investigation, we have uncovered developer interactions within code reviews, and whether their code review participation could be marked as a controversy. There have been new metrics created that will also neglect any outliers in the data we are manually investigating or running correlations on. This data reveals several results containing facts about developer's experience contributing to how many vulnerabilities are present in a code review.

Between creating new metrics and modifying existing ones in our data, we were able to truly analyze the correlations between developers in the Chromium network. Our apparent collaboration on this project has led us to expedite the outcome for more traceable data to resolve the concerns in our research questions. By having more eyes to seek out vulnerabilities in a large software project, there are more conclusions to infer about developer vulnerability data.

Links:
https://github.com/andymeneely/chromium-history/