Predicting Vulnerabilities in the Free Open Source Software Ecosystem

by
Elsie Phillips

A THESIS

submitted to

Oregon State University

Honors College

in partial fulfillment of
the requirements for the
the degree of

Honors Baccalaureate of Science in Economics
(Honors Scholar)

Presented May 13, 2016
Commencement June 2016

AN ABSTRACT OF THE THESIS OF

Elsie Phillips for the degree of Honors Baccalaureate of Science in Economics presented on May 13th, 2016. Title: Predicting Vulnerabilities in the Free Open Source Software Ecosystem

Abstract Approved:

Carlos Jensen

Due to the interdependent nature of Free Open Source Software projects, a vulnerability in just one highly used project can have significant and sweeping consequences across many projects, and can inflict hundreds of millions of dollars in damage. This paper proposes a model for predicting software vulnerabilities in highly used FOSS projects using measures of effort and complexity. We used several measures of complexity and effort to look at the top 150 projects listed on the Debian Popularity Contest. We determined that total development effort was the best measure of effort and lines of code was the best measure of complexity for predicting software vulnerabilities in these projects.

Key Words: Open Source, FOSS, Vulnerabilities
Corresponding email: elsiemaephillips@gmail.com

Predicting Vulnerabilities in the Free Open Source Software Ecosystem

by
Elsie Phillips

A THESIS

submitted to

Oregon State University

Honors College

in partial fulfillment of
the requirements for the
the degree of

Honors Baccalaureate of Science in Economics
(Honors Scholar)

Presented May 13, 2016
Commencement June 2016

APPROVED:

Carlos Jensen, Mentor, representing Computer Science

Liz Schroeder, Committee Member, Representing Economics

Lance Albertson, Committee Member, Representing the Open Source Community

---

Toni Doolen, Dean, University Honors College

I understand that my project will become part of the permanent collection of Oregon State University, University Honors College. My signature below authorizes release of my project to any reader upon request.

Elsie Phillips, Author

Introduction

The Free Open Source Software (FOSS) ecosystem is a delicate one. With dependencies spanning the full breadth of the community, vulnerabilities in just a few projects can have serious ramifications for the entire ecosystem. For example, in 2014, a vulnerability in OpenSSL, the most widely used open source cryptographic library[1], caused 17%[2] of the web's secure servers to become susceptible to attack. According to the Heartbleed Bug site, the bug, "...Allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This compromises the secret keys used to identify the service providers and to encrypt the traffic, the names and passwords of the users and the actual content. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users." In practice this means that user names, passwords, instant messages, emails, and other documents and communications became accessible through this vulnerability.[3]

Cleaning up after Heartbleed has been extremely expensive. According to some estimates, fixing this vulnerability has cost at least $500 million.[4] At the time the Heartbleed vulnerability was discovered, OpenSSL was significantly underfunded for the scope of the influence of their project. It provided authentication checking for hundreds of thousands of secure servers and software produced by multi-billion dollar companies, and yet, its average annual operating budget was just $2,000, and only two full-time developers was dedicated to

---

[1] "The Heartbleed Bug," Heartbleed Bug, Codenomicon, Apr, 2014, Web, 04 May 2016, <http://heartbleed.com/>.
[2] Paul Mutton, "Half a Million Widely Trusted Websites Vulnerable to Heartbleed Bug," Netcraft, N.p., 08 Apr, 2014, Web, 04 May 2016.
[3] "The Heartbleed Bug," Heartbleed Bug, Codenomicon, Apr, 2014, Web, 04 May 2016, <http://heartbleed.com/>.
[4] Steve Pate, "Measuring the Aftershocks of Heartbleed," Security Magazine, BNP Media, 13 May 2014, Web, 04 May 2016.

maintaining the code base.[5] Due to this lack of resources, it was unable to stave-off unmanageable technical debt, which led to far reaching consequences. To prevent future high-impact vulnerabilities, endangered critical projects, like OpenSSL, need to be identified and have the appropriate resources channeled to them to maintain the health of the entire FOSS ecosystem.

Related Work

Open source software is a dramatic paradigm shift from traditional proprietary software development. The source code is made available to the user to review and modify. The user may then submit those modifications back to the original developers to be incorporated into the canonical project. FOSS is a subgroup of open source software. In practice there is little difference between open source software and FOSS other than the philosophy of the developers. FOSS developers espouse a user's essential freedoms: to run, study, change and redistribute copies of the software with or without changes. Open source is a development methodology. FOSS is an ethical stance.[6]

In the last two decades open source software has experienced rapid growth, even becoming the defacto software in certain areas. For example, the Apache web server is used by 52.7% of all websites according to W3Techs.[7] Its closest competitor is NGINX is also open source. Despite their ability to submit changes to the codebase, very few users choose to do so.

---

[5]http://arstechnica.com/information-technology/2014/04/tech-giants-chastened-by-heartbleed-finally-agree-to-fund-openssl/

[6] Richard Stallman, "Why Open Source Misses the Point of Free Software - GNU Project - Free Software Foundation," Gnu.org, N.p., n.d, Web, 06 May 2016.

[7] "Usage Statistics and Market Share of Apache for Websites," W3Techs, N.p., 01 May 2016, Web, 04 May 2016, <http://w3techs.com/technologies/details/ws-apache/all/all>.

Of those that do submit code, only a small percentage contribute the majority of code. It was found that across 3149 open source software projects and 25 million lines of open source code, 72% of the code was contributed by the top 10% of contributors (Ghosh and Prakash, 2000). It has also been shown that the ratio of users who contribute code to users who submit bug reports is 1 to 5 (Valloppillil, 1998).

It is difficult to paint a comprehensive picture of the health of an open source project. One measure of health for any software project is its level of technical debt. There is very little consensus on what constitutes technical debt in the literature. The term was first used by Cunningham in 1992 as a metaphor to explain that in the short-run, code quality might be borrowed against to buy something else, but in the long run that quality must be paid back.

> *Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite...The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented or otherwise.* [8]

This metaphor has been extended to describe many states in software development, for example documentation debt, which is the difference between a project's current documentation, and what would be considered "complete" documentation of the project. In many of the varying definitions for technical debt there is the idea of it being a measure

[8] W. Cunningham, 1992, *The WyCash Portfolio Management System*, *OOPSLA' 92 Experience Report*.

between the current state and the ideal state of the software. McConnel and Fowler proposed that technical debt could be categorized as reckless or prudent and deliberate or inadvertent.

A project's ability to pay down its technical debt is dependent on the resources available to it. These resources can be measured in number of contributors or time necessary for the software development. There have been several methods developed to estimate the level of effort required for a software development project. The most widely adopted is the Constructive Cost Model (COCOMO).

Developed in 1981 and updated in 2000 by Boehm, COCOMO is a way to project the necessary cost and developer time necessary for a software project. COCOMO is subdivided into three categories: basic, intermediate, and detailed. Basic COCOMO is good for quick and high level projections as it estimates effort as a function of lines of code but lacks a way to account for differentiating project attributes. The intermediate and detailed levels both take project attributes into account, with the detailed level also factoring in the stage of development of the project.

Methodology

The first step that we took in developing our endangeredness model was to generate a pool of highly used open source projects. To do this, we needed a benchmark for "highly used". For our purposes, "highly used" refers to a project that is included in top 150 projects on the Debian Popularity Contest and has a public git repository. This left us with 85 projects to examine. There were several projects in the top 150 projects that were not included because they

did not have public git repositories, or they were included in other projects, like glibc which shares a git repository with GCC.

Next we cloned the git repositories of each project and ran a data collecting bash script to analyze the source code and the git commits for each project. The script collected information on:

- Lines of code (loc)
- Initial commit date (initialcommit)
- Recent commit date (recentcommit)
- Active contributors (determined by the number of contributors in the last calendar year
    - Jan 1, 2015-Dec 31st 2015) (activedevs)
- The total number of contributors a project has had over its lifetime (totaldevs)

Using this data we performed several other calculations. First, we computed the number of years of active development a project has had. We did this by subtracting the initial commit date from the most recent commit date (yearcount). Second, we calculated the age of the project by subtracting the initial commit date from the most recent date and rounding down (age). We also calculated several measures of development effort. The first was COCOMO, which is divided into a three tier hierarchy of complexity: Basic, Intermediate, and Detailed. We selected the basic categorization for our calculations. Within the basic model projects are categorized as organic, semidetached, or embedded. An organic project has a small experienced contributor base and has flexible project requirements. A semi-detached project has a midsized contributor base with mixed levels of experience and with relatively flexible project requirements. An embedded project has rigid requirements. We decided to uniformly categorize all projects as organic. The formula to calculate effort for an organic project is $a_b(KLOC)^b_b$ where $a_b = 2.4$ and

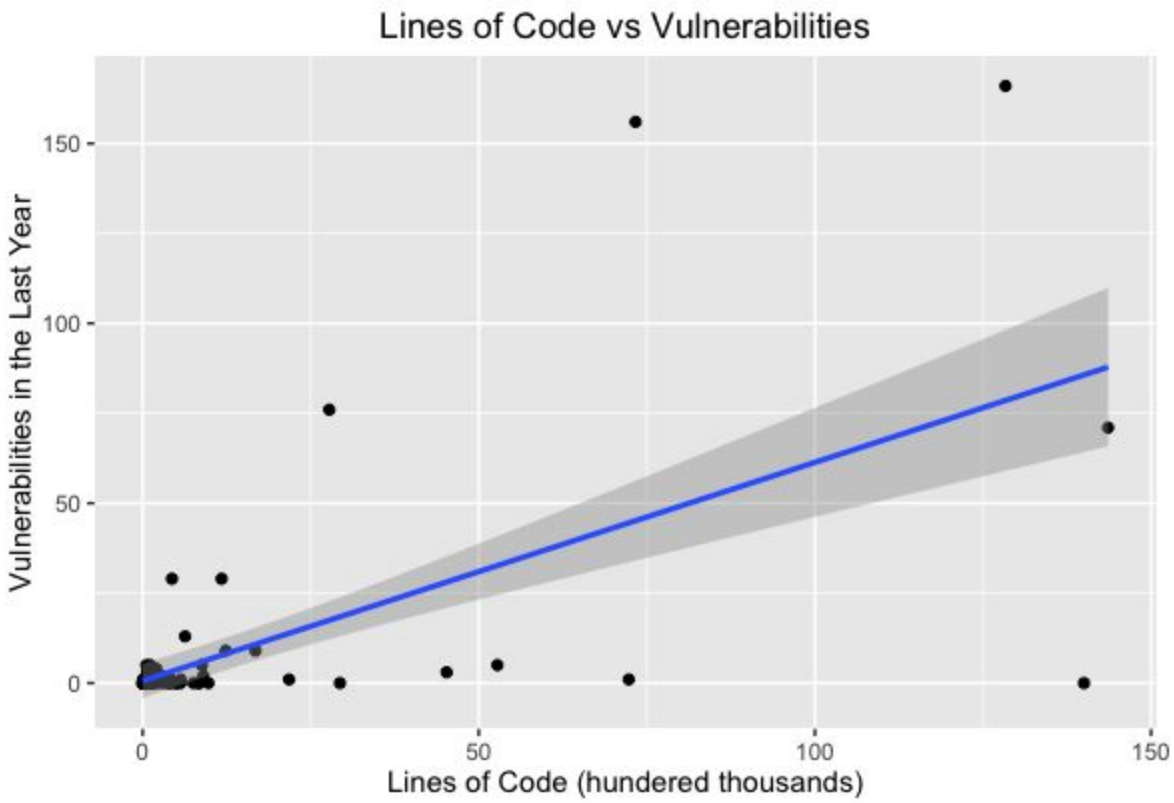
$^{b}_{b} = 1.05$ (COCOMO). We calculated the maximum developer effort that could have been expended on the project by multiplying the total number of developers by the year count (totaleffort). Finally, we calculated the current developer effort by multiplying the current number of developers by the year count (currenteffort).

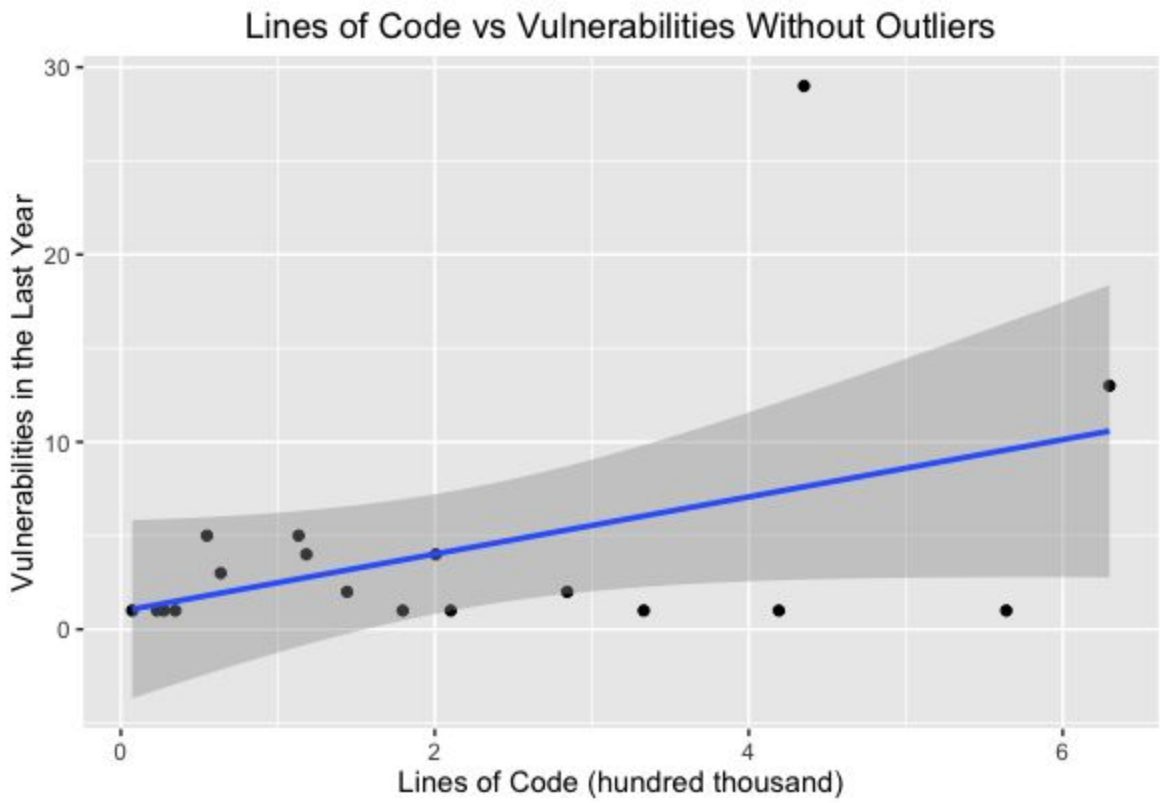
Next we collected information about the number of open tickets in each project. We did this by going to each project's public issue tracker, whether it be bugzilla, github issues, sourceforge, or something else, and without filtering, recorded the number of open tickets. For the projects that used the Debian project bug tracking system we recorded the number of bugs on the unstable release of the project because we had not done any filtering in the other projects' bug tracking systems. However, we did choose not to include the bugs that had been marked as "won't fix" since they were effectively not open. All other categories of bugs were included from the Debian project bug tracker.

Finally we recorded the number of security vulnerabilities filed with the Common Vulnerabilities and Exploits database. This database represents the central repository of information about information security vulnerabilities. The search function for the database is very weak, and operates purely off of keywords. Each vulnerability that had the particular project name in it had to be read and categorized as a vulnerability in the project or in another piece of software manually.

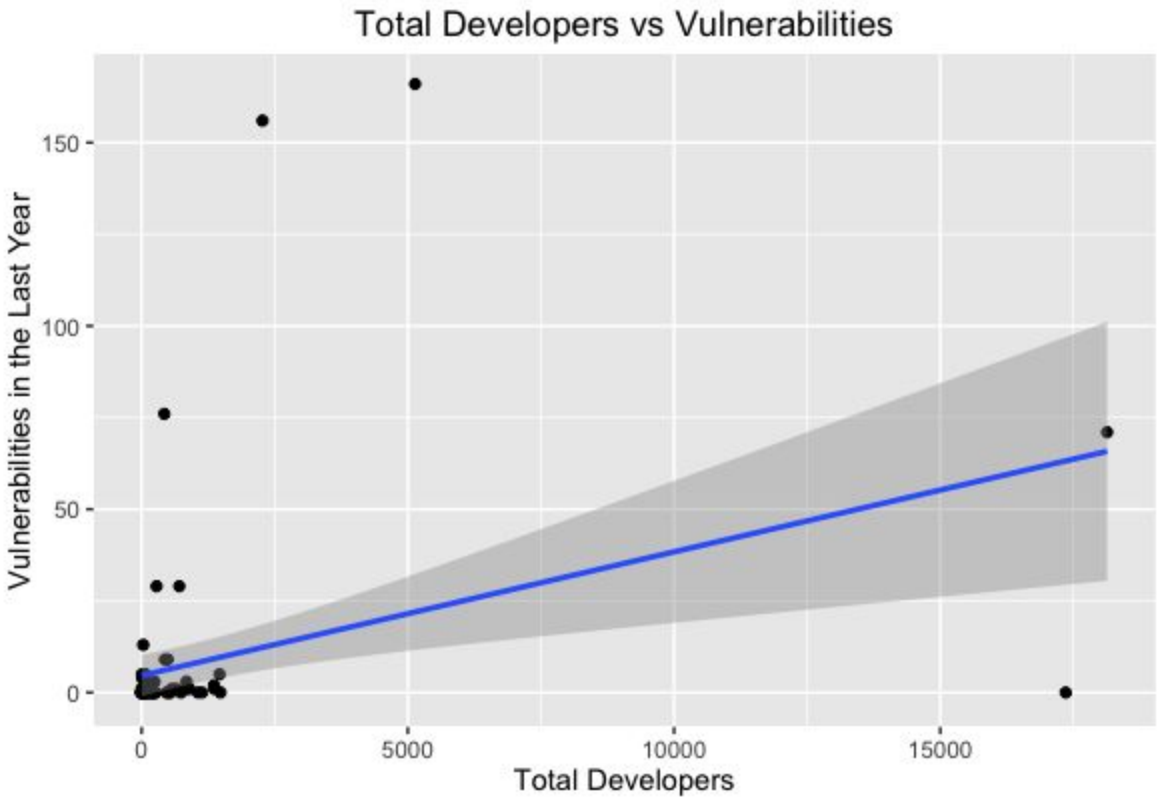
Results

We theorized that vulnerabilities could be predicted using a measure of complexity and effort. After running several correlations we discovered a strong positive relationship between COCOMO and vulnerabilities and loc and vulnerabilities, both having a .632 correlation. This makes sense because COCOMO is a function of loc. As loc increases, there are more opportunities to introduce vulnerable code.





A positive relationship, 0.344309, also arose between total developers and vulnerabilities. This was unexpected, but could be attributed to scale code review as the number of contributors increases. There was not a correlation between bug count and vulnerabilities. This is perhaps due to a project's maintainer or maintainers ability to perform triage and are maybe more likely to fix vulnerable code than buggy code.


Total Developers vs Vulnerabilities
Vulnerabilities in the Last Year
150
100
50
0
0
5000
10000
15000
Total Developers

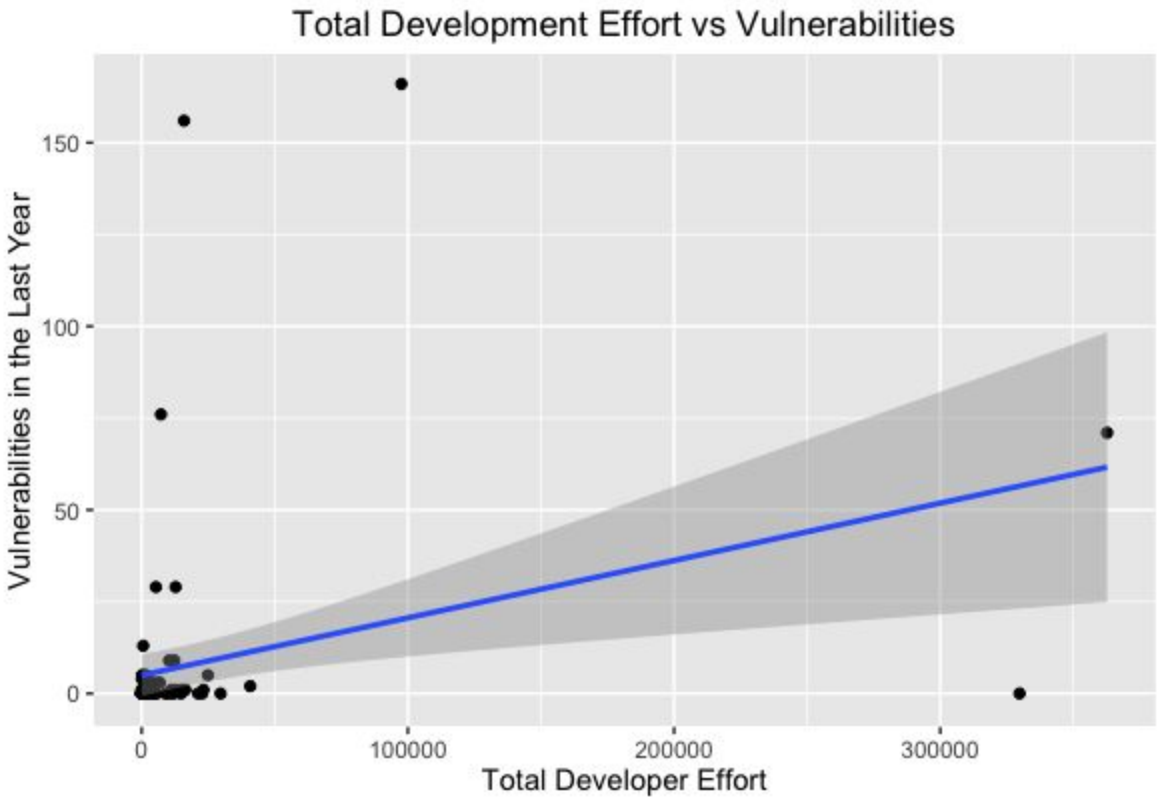

Total Developers vs Vulnerabilities Without Outliers
Vulnerabilities in the Last Year
30
20
10
0
0
500
1000
1500
Total Developers

	Mean	Median	Max	Min	Standard Deviation
age	14.04	15	28	1	279
activedevs	150.1	12	4292	0	639.7855
bugcount	2751.0	108.0	75530	0	NA
COCOMO	4139.00	375.1	57940	0.07	11185.18
loc	1079000	118200	14360000	35	2796273
totaldevs	747.2	84	18130	1	2740.66
vulnerabilities	7.165	0	166	0	26.84428

The following model emerged after looking at the correlations of variables:

$$\text{Vulnerabilities} = \beta 0 + \beta 1 \, \textit{hundred thousand loc} + \beta 2 \, \textit{total effort} + u$$

Loc is the measure of complexity in the equation and is measured in thousands of loc because of the high mean for loc. There needs to be a significant change in loc for there to be an effect on vulnerabilities. Total effort was selected because it combined the effects of both the years of development and total number of developers.


Total Development Effort vs Vulnerabilities
Vulnerabilities in the Last Year
150
100
50
0
0
100000
200000
300000
Total Developer Effort

$\widehat{\beta}$1 hundred thousand lines of code = 1.67

For every hundred kloc added to the codebase while holding the number of developers constant, the number of predicted vulnerabilities increases by 1.67. This result was significant at less than 1%.

$\widehat{\beta}$2 *total effort* = -0.00035

This means that for every one developer year added to a project holding loc constant, the predicted number of vulnerabilities decreases by 0.00035. For an increase of 100 developers to a

project, the predicted vulnerabilities decreases by 0.035. This was also significant at less than 1%. The multiple r-squared for this model is 0.5512.

The robustness of this model was checked in two ways. 65% of the projects we examined did not have vulnerabilities logged with the CVE in the last year. To investigate the effects of adjusted loc and total effort on the extensive vs. intensive margin of vulnerabilities we divided the analysis into two parts. First we estimated a linear probability model, regressing an indicator for having 1 or more vulnerabilities onto loc and total effort. Then we used the Huber-White Sandwich estimators to correct for heteroskedasticity and ran a regression on the group that had at least one vulnerability. Both adjusted loc and total effort were significant predictors of having a vulnerability, as well as increasing the number of vulnerabilities, conditional on having had at least one. The coefficients after the projects with no vulnerabilities were removed and the heteroskedasticity was accounted for with the Sandwich estimators were as follows:

$\widehat{\beta}1$ hundred thousand lines of code = 1.127207

$\widehat{\beta}2$ *total effort* = -0.0002267

We also checked the model's sensitivity to outliers. There were several projects that were much larger than the others and we wanted to check and see if they were driving any of the results or if the group of smaller projects was qualitatively different. We first ran the regression looking only at projects that had a loc of less than 755,555 because there was over 100,000 loc

division between projects. This resulted in the exclusion of 19 projects. The coefficients after excluding those projects were as follows:

$\widehat{\beta}1$ hundred thousand lines of code = 0.8866046

$\widehat{\beta}2$ *total effort* = -0.0000567

Then we ran the regression excluding projects with less than 70 vulnerabilities. We did this because there was also a clear break in the data there. This led to the exclusion of 4 projects. The coefficients were as follows:

$\widehat{\beta}1$ hundred thousand lines of code = 0.061438

$\widehat{\beta}2$ *total effort* = -0.0000276

In all of these robustness checks, the coefficients for the variables decreased, but there were no qualitative change to the coefficients. Due to the high significance levels of the variables, and the lack of qualitative changes to the coefficients after removing the outliers and removing the projects that did not have vulnerabilities, we believe this model to be a robust predictor of vulnerabilities.

With All Projects		Without Projects with > 0 Vulnerabilities		Without Projects with >755,555 LOC	
	Coefficient		Coefficient		Coefficient
LOC	1.67	LOC	1.127207	LOC	0.8866046
Total Effort	-0.00035	Total Effort	-.0002267	Total Effort	-0.0000567
P-value for Model	0	P-value for Model	0	P-value for Model	0.0125

Limitations

Rather than using a pre-existing script for data collection, we chose to write our own. This script could have been flawed in such a way that could have affected the validity of the data. Of the data set collected with the scripts, the age of the project and the initial commit date are both not guaranteed to be accurate. Age is calculated by subtracting the current date from the initial commit date, and then rounding down to the nearest whole year. However, if a project was ported over to git, that initial commit would not reflect the beginning of effort on the project. This would lead to an underestimation of a project's age. However, age did not prominently factor into our model.

Several components of the dataset had to be manually collected. Vulnerabilities had to be individually screened due to the lack of sophisticated filtering for the CVE database. A vulnerability could have been incorrectly categorized as applying or not applying to a project. It

was also not possible to collect the number of bugs per project using a script, so this data also needed to be individually collected. Additionally, several projects lacked publically viewable bug tracking systems. Of these projects, a few had Debian bug trackers. How closely these bug trackers mirror the true state of the project is unknown.

Conclusion

Open source software over the last two decades has seen a rapid proliferation of adoption. These technologies have become the backbones of both small hobby software projects, and multi-million dollar enterprise software. They have developed into a delicate ecosystem of interdependent projects. When the health of one project becomes endangered due to vulnerabilities, it has the potential to have wide sweeping effects across both proprietary and open source software. Several of the projects we examined are supported by large corporations or non-profits. Many however are left at the mercy of the volunteer open source community for their continued maintenance. We have shown that as the number of developers contributing to a project increases, the number of predicted vulnerabilities decreases, but only on a significant scale. It takes 100 additional developer years to reduce vulnerabilities by 3.5%. High impact open source projects need the support of entities that are able to donate developer time or the funds to pay for full-time development. Relying on the community to be able to adequately maintain these critical projects in their spare time and with donations from individuals is a risky gamble that the ecosystem cannot afford.

Citations

Brown, Nanette, Ipek Ozkaya, Raghvinder Sangwan, Carolyn Seaman, Kevin Sullivan, Nico Zazworka, Yuanfang Cai, Yuepu Guo, Rick Kazman, Miryung Kim, Philippe Kruchten, Erin Lim, Alan Maccormack, and Robert Nord. "Managing Technical Debt in Software-reliant Systems." *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research - FoSER '10* (2010): n. pag. Web.

Fowler, M. 1999. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional.

Ghosh, Rishab Aiyer, and Vipul Ved Prakash. "The Orbiten Free Software Survey." *First Monday* 5.7 (2000): n. pag. May 2000. Web. 03 May 2016.

"The Heartbleed Bug." *Heartbleed Bug*. Codenomicon, Apr. 2014. Web. 04 May 2016. <http://heartbleed.com/>.

Mutton, Paul. "Half a Million Widely Trusted Websites Vulnerable to Heartbleed Bug." *Netcraft*. N.p., 08 Apr. 2014. Web. 04 May 2016.

Nugroho, Ariadi, Joost Visser, and Tobias Kuipers. "An Empirical Model of Technical Debt and Interest." *Proceeding of the 2nd Working on Managing Technical Debt - MTD '11* (2011):

n. pag. Web.

Pate, Steve. "Measuring the Aftershocks of Heartbleed." *Security Magazine RSS*. BNP Media, 13 May 2014. Web. 04 May 2016.

Stallman, Richard. "Why Open Source Misses the Point of Free Software - GNU Project - Free Software Foundation." Gnu.org. N.p., n.d. Web. 06 May 2016.

"Usage Statistics and Market Share of Apache for Websites." *W3Techs*. N.p., 01 May 2016. Web. 04 May 2016. <http://w3techs.com/technologies/details/ws-apache/all/all>.

Valloppillil, V., 1998, `Open Source Software: A (New?) Development Methodology', Unpublished working paper, Microsoft Corporation, http://www.gnu.org/software/fsfe/projects/ms-vs-eu/halloween1.html (accessed April 04, 2016).