Empir Software Eng (2017) 22:1305–1347
DOI 10.1007/s10664-016-9447-3

Do bugs foreshadow vulnerabilities? An in-depth study of the chromium project

Nuthan Munaiah[1] · Felivel Camilo[1] · Wesley Wigham[1] · Andrew Meneely[1] · Meiyappan Nagappan[1]

Published online: 18 August 2016

Abstract As developers face an ever-increasing pressure to engineer secure software, researchers are building an understanding of security-sensitive bugs (i.e. vulnerabilities). Research into mining software repositories has greatly increased our understanding of software quality via empirical study of bugs. Conceptually, however, vulnerabilities differ from bugs: they represent an abuse of functionality as opposed to insufficient functionality commonly associated with traditional, non-security bugs. We performed an in-depth analysis of the Chromium project to empirically examine the relationship between bugs and vulnerabilities. We mined 374,686 bugs and 703 post-release vulnerabilities over five Chromium releases that span six years of development. We used logistic regression analysis, ranking analysis, bug type classifications, developer experience, and vulnerability severity metrics to examine the overarching question: are bugs and vulnerabilities in the same files? While we found statistically significant correlations between pre-release bugs and post-release vulnerabilities, we found the association to be weak. Number of features, source lines of code, and pre-release security bugs are, in general, more closely associated with post-release

Communicated by: Romain Robbes, Martin Pinzger and Yasutaka Kamei.

✉ Nuthan Munaiah
nm6061@rit.edu

Felivel Camilo
fddc7162@rit.edu

Wesley Wigham
waw5336@rit.edu

Andrew Meneely
axmvse@rit.edu

Meiyappan Nagappan
mxnvse@rit.edu

1 Department of Software Engineering, Rochester Institute of Technology, 134 Lomb Memorial Drive, Rochester, NY, 14623, USA

vulnerabilities than any of our non-security bug categories. In further analysis, we examined sub-types of bugs, such as stability-related bugs, and the associations did not improve. Even the files with the most severe vulnerabilities (by measure of CVSS or bounty payouts) did not show strong correlations with number of bugs. These results indicate that bugs and vulnerabilities are empirically dissimilar groups, motivating the need for security engineering research to target vulnerabilities specifically.

Keywords Vulnerability · Code review · Bugs · Security

1 Introduction

Developers face a difficult challenge today: deliver functional software that is secure. A simple coding mistake or design flaw can lead to an exploitable vulnerability if discovered by the wrong people. These vulnerabilities, while rare, can have catastrophic and irreversible impact on our increasingly digital lives. Vulnerabilities as recent as Shellshock (CVE-2014-6271) and Heartbleed (CVE-2014-0160) are reminders that small mistakes can lead to widespread problems. To engineer secure software, developers need a scientifically rigorous understanding of how to detect and prevent vulnerabilities.

A vulnerability may be viewed as a special kind of software bug, one that has security consequences. Formally, a software vulnerability may be defined as a software defect that violates an implicit or explicit security policy (definition adapted from Krsul (1998); Bird et al. (2015)). Prior empirical analysis of software bugs has greatly aided our understanding of software quality. These studies have resulted in a myriad of metrics, prediction models, hypothesis tests, and other actionable empirical insight that speaks to the nature of software bugs (Chen et al. 2012; Gegick et al. 2009; Meneely et al. 2014).

Since vulnerabilities are a special kind of software bugs, one may assume that all software quality research translates into software security. However, vulnerabilities are conceptually different from traditional bugs. While traditional bugs indicate missing, insufficient, or incorrect functionality, vulnerabilities indicate misuse or abuse of functionality. In other words, vulnerabilities allow the use of functionality beyond what the system was intended for, resulting in an increased exposure to malicious misuse. Vulnerabilities may also be viewed as hidden features for attackers to discover. For example, an open permissions policy may work perfectly well for most users but it can be exploited by attackers, a simple memory leak that can be coerced into a denial-of-service attack, etc.

The empirical delineation of the nature of relationship between bugs and vulnerabilities may lead to a better understanding of means to improve software security. Empirical evidence of a strong relationship between bugs and vulnerabilities could imply that future empirical software security research should primarily focus on the super group of bugs.

The objective of this research is *to improve our fundamental understanding of vulnerabilities by empirically evaluating the relationship between bugs and vulnerabilities*. Specifically, we wish to understand if bugs and vulnerabilities are in the same place. The subject of study used in the empirical evaluation was the Chromium open-source web browser project (a.k.a. Google Chrome). We conducted an in-depth analysis of Chromium by collecting version control data, bug reports, code review reports, and data from post-release vulnerabilities for over six years of the project. We evaluated the association between bugs and vulnerabilities using a non-parametric null hypothesis test. We used effect size measure to evaluate the strength of the association (if any). We segregated bugs into categories and used regression analysis to evaluate if certain categories of bugs were more closely associated with vulnerabilities. We used rank analysis to investigate if the most buggy files were also most vulnerable. We also investigated the relationship between bugs and vulnerability severity. The severity of a vulnerability was determined by the Common Vulnerability Scoring System (CVSS) base score assigned to it. We also approximated vulnerability severity using the bounty paid to developer(s) who reported and/or fixed the bug that resolved the vulnerability. As an additional dimension, we evaluated if developers' experience reviewing certain categories of bugs had any effect on vulnerability likelihood. All of our analyses were repeated on five releases of the Chromium project with each release separated from its predecessor by approximately one year of development.

We addressed the following research questions (and had the following results):

RQ1: *Are source code files fixed for bugs likely to be fixed for future vulnerabilities?*
(We found that files fixed for more pre-release bugs were *slightly* more likely to also be fixed for post-release vulnerabilities).

RQ2: *Are some types of bugs (including features) more closely related to vulnerabilities than others?*
(We found that some types of pre-release bugs were more strongly associated with post-release vulnerabilities than others, however, the association was overall weak.)

RQ3: *Do source code files with most bugs also have the most vulnerabilities?*
(We found that files with a higher number of pre-release bugs had only a few post-release vulnerabilities).

RQ4: *Are bugs more likely to be in files with high vulnerability severity?*
(We found that files with severe vulnerabilities did not have a correspondingly high number of pre-release bugs).

RQ5: *Do source code files reviewed by more bug review experienced developers have fewer vulnerabilities?*
(We found that vulnerable files had fewer bug review experienced developers than neutral files. However, review experience metrics did not out-perform corresponding pre-release bug metrics in indicating vulnerability likelihood).

RQ6: *Does the relationship between bugs and vulnerabilities generalize beyond Chromium?*
(Where direct comparisons could be made, we found the relationship between bugs and vulnerabilities to generalize to Apache httpd).

The remainder of this article is organized as follows: we begin by describing the various terms used in the article in Section 2. In Section 3, we provide an overview of the state of the art in bugs and vulnerabilities research. In Section 4, we briefly introduce the Chromium project and describe the various sources of data used in the empirical analysis. The various file level metrics collected from source code files are described in Section 5. In Section 6, we describe the analysis methodology, answer the research questions, and discuss the results. In Section 7, we present the implications of the our results on software security research. We discuss the threats to validity in Section 8 and conclude the article with a brief summary in Section 9.

2 Terminology

We use several technical and statistical terms when describing the processing and analyses of the data. In the subsections that follow, we define these terms to ensure that the essence of the paper is easily comprehensible.

2.1 Data Terms

The term **release** refers to a milestone in the development life cycle of a software project. A release represents a snapshot of all source code files at a specific point in time. The project evolves from one release to the next through small changes called **commits**. These commits represent unique changes to the source code of the project, and are recorded in a version control system (e.g. Git). We make a special distinction between non-security related software flaws (i.e. **bugs**), that manifest in the lack of an expected functionality, and security related software flaws (i.e. **vulnerabilities**), that manifest in an exploit resulting in the violation of the system's security policies (Krsul 1998). We categorize files of a release as **vulnerable** if they were fixed for a post-release vulnerability, and the remaining as **neutral** (i.e. no vulnerabilities were known to be associated with them).

We also use the Source Lines of Code (**SLOC**) metric as a measure of file size. We note that blank lines and lines containing only source code comments are not included in SLOC value computed. SLOC is a member of a group called traditional software metrics (Tegarden et al. 1992). Code churn and cyclomatic complexity are some of the other metrics in the group.

2.2 Statistical Terms

2.2.1 Spearman's Rank Correlation Coefficient (ρ)

The Spearman's rank correlation coefficient, ρ, is a measure of association between two or more variables. The magnitude of ρ may be used to assess the strength of the association, whereas, the sign of ρ may be used to assess the direction of the association (i.e. positive or negative). We use the following scale (from Mukaka (2012)) in interpreting the strength of correlation: (a) "negligible correlation" when magnitude of ρ is between 0.00 and 0.30, (b) "low correlation" when magnitude of ρ is between 0.30 and 0.50, (c) "moderate correlation" when magnitude of ρ is between 0.50 and 0.70, (d) "high correlation" when magnitude of ρ is between 0.70 and 0.90, and (e) "very high correlation" when magnitude of ρ is between 0.90 and 1.00.

2.2.2 Mann-Whitney-Wilcoxon (MWW)

The MWW test is a non-parametric statistical test to evaluate the null hypothesis that two populations have an identical distribution. The rejection of the null hypothesis would indicate that the two populations (in this study, vulnerable and neutral) are distributed differently and hence have different statistical parameters such as mean and median. Prior literature on software metrics (Meneely et al. 2014; Schneidewind 1992) have suggested MWW as a suitable test for validating software metrics.

2.2.3 Cohen's d

The MWW test reveals if there exists a statistically significant association between two populations (in this study, vulnerable and neutral), however, the strength of the association cannot be inferred from the test outcome. The strength of an association may be evaluated using an effect size measure statistic. In this study, we use the Cohen's d statistic (Cohen 2013; Ruscio 2008) as the effect size measure to complement the outcome from MWW test, and to measure the degree to which the population of vulnerable and neutral files differ.

The Cohen's d measure of effect size between two populations, with different number of observations and heterogeneous variance, may be estimated using the following equation:

$$d = \frac{\mu_1 - \mu_2}{\sigma_{pooled}}$$

Where, μ_1 and μ_2 are the means of the two populations and σ_{pooled} is the pooled standard deviation computed using the following formula:

$$\sigma_{pooled} = \sqrt{\frac{(n_1 - 1) \cdot \sigma_1^2 + (n_2 - 1) \cdot \sigma_2^2}{n_1 + n_2 - 2}}$$

Where, n_1 and n_2 are the number of observations and σ_1^2 and σ_2^2 are the variances in the two populations.

We use the heuristic proposed by Cohen (1992) in interpreting the strength of effect of a phenomenon from the value of d. The heuristic is as follows: (a) "negligible effect" when $d < 0.20$, (b) "small effect" when $0.20 \leq d < 0.50$, (c) "medium effect" when $0.50 \leq d < 0.80$, and (d) "large effect" when $d \geq 0.80$.

2.2.4 *Logistic Regression*

Logistic regression analysis may be used to determine if there exits a model that can depict the relationship between a set of explanatory variables and a binary outcome (Chen et al. 2012; Cruz and Ochimizu 2009). In our study, the binary outcome to be explained was the vulnerability likelihood of a file as determined by a specific set of metrics collected from a software system.

2.2.5 *Model Goodness of Fit*

In practice, the regression model, that depicts the relationship between a response variable and a set of explanatory variables, is only an estimation of the true regression model generated using the available data set. The goodness of fit of a model is a measure of how well the purported model fits the data set from which the model itself was estimated. There are several metrics that quantify the goodness of fit of a model. In our study, we have used the following metrics:

Akaike Information Criterion (AIC) AIC is a measure of relative model goodness of fit calculated on generalized models. The AIC metric may be used to rate a group of models and find the best one among them. The absolute value of AIC is not a measure of the goodness of fit of a model, however, the change in the AIC value between two models trained with the same data set may be used to identify the model that better fits the data. A lower AIC value means a better fit model (Chen et al. 2012; Raftery 1995; Burnham and Anderson 2004).

Percent of Deviance Explained (D^2) The percent of deviance explained, D^2, indicates how well the data fits a statistical model. D^2 is the equivalent of R^2 for linear models (Chen et al. 2012; Guisan and Zimmermann 2000), except that this metric is used to evaluate the amount of deviance that the model accounts for. The value of D^2 may also be used to evaluate the goodness of fit of a group of generalized linear models, where a higher D^2 indicates a better fit.

2.2.6 Model Performance

Typically, regression models are built with the intention of predicting the response variable for some yet unknown values of the explanatory variables. The data set used to build the regression model is often called the *training set* since the model is being "trained" to learn patterns in the training set data. The performance of a model is a measure of how well the purported model performs when applied to predicting response from unknown data (i.e. data not in the training set). Depending the type of the response variable (binary, continuous, etc.), there are several metrics that quantify the performance of a model. In our study, since the response variable is binary, we have used the following metrics:

Precision and Recall When a logistic regression model is used to predict a binary outcome, say vulnerability likelihood, the model can make two types of mistake: False Vulnerable (FV) and False Neutral (FN). As an example, consider a logistic regression model that predicts whether a given file is vulnerable or neutral. In the context of this model, false vulnerable is a scenario where a neutral file is purported to be vulnerable and false neutral is a scenario where a vulnerable file is purported to be neutral. The relevant values, i.e. when the model accurately identifies a vulnerable and neutral file, are referred to as True Vulnerable (TV) and True Neutral (TN), respectively.

Precision is defined as the fraction of all the predictions that are relevant, and recall is defined as the fraction of all relevant instances that are retrieved. Precision and recall of a regression model may be estimated using the following equations:

$$\text{Precision} = \frac{\text{TV}}{\text{TV} + \text{FV}} \qquad \text{Recall} = \frac{\text{TV}}{\text{TV} + \text{FN}}$$

As an example, consider a data set of 50 files of which 30 files are known to be "vulnerable" and the remaining 20 are known to be "neutral". If a logistic regression model was used to retrieve the 30 vulnerable files and if it identified 10 vulnerable files correctly and misidentified 20 neutral files as vulnerable, then Precision = 10/20 and Recall = 10/30.

F-measure The F-measure, F_1, may be interpreted as a weighted average of precision and recall. The F-measure is considered a more complete metric than using only precision or recall (Chen et al. 2004) because it takes both values into account. The F-measure may be computed using the following equation:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Area Under the ROC Curve (AUC) Receiver Operating Characteristic (ROC) curves may be used to graphically represent the performance of a logistic regression model by plotting the false vulnerable rate versus the true vulnerable rate. The area under the ROC curve measures the ability of the model to correctly classify the binary outcome. As an example, assume that we have separated the vulnerable and neutral files. If we were to randomly select two files, one from the vulnerable group and the other from the neutral group, and used a model to predict the outcome (vulnerable or neutral), AUC represents the percentage of observations in which the model will be able to tell the two files apart.

As mentioned earlier, regression models are typically built with the intention of predicting the future value of the response variable using the values of the explanatory variables at some future state. However, during regression analysis, the future state of the explanatory variables cannot be determined and hence must be simulated to test the performance of the model. The most common approach is to split the available data set into training set and testing set. The model is built using the training set and its performance measured using the testing set. There are multiple algorithms that leverage a variation of this underlying principle. In our study, we have used two such algorithms, they are:

k-fold Cross-validation In k-fold cross-validation, the data set is randomly split into k equal segments. k models are then built leaving one of the segments as the testing set. The model performance metrics collected from each of the k models are aggregated to assess the overall model performance. Since the splitting of the data set into k equal segments is randomly achieved, each k-fold run may be repeated multiple times. In our study, we have used 10-fold cross-validation with each 10-fold repeated 10 times.

Next Release Validation An approach to model performance assessment that is particularly useful in the context of Software Engineering research is the next release validation. In next release validation, a model is trained with data from one release and its performance is assessed by testing it with data from a chronologically adjacent release. For example, if we were to build a bug prediction model for a piece of software that has two releases, 1.0 and 1.1, the model could be trained with data from release 1.0 and tested with data from release 1.1. Next release validation lends itself nicely to the simulation of a typical software development workflow, in which data from past releases is available and may be used in predicting the likelihood of future occurrence of a phenomenon.

In our study, we have used k-fold cross-validation when building logistic regression models using the entire data set and next release validation when building logistic regression models using data set obtained from specific releases.

3 Related Work

Researchers have studied the relationship between bugs and vulnerabilities in the past. In our study, we extend some of the research questions from prior literature by performing an in-depth analysis on a larger data set.

Gegick et al. (2009, 2008) studied the possibility of predicting the attack proneness of a component by identifying and ranking components that were more likely to present vulnerabilities. These rankings were to aid in prioritizing the security risk management efforts. In these studies, the authors used automated static analysis tools to collect different metrics and analyze how these metrics related to higher vulnerability risk. The objective in these studies was to evaluate the vulnerability-proneness of files based on non-security factors (i.e. code churn, SLOC, and previous manual inspections). In our study, we used a more elaborate data set that contained data aggregated from multiple sources such as the version control system, bug tracking system, code review system, and vulnerability database. We chronologically segregated the data into subsets corresponding to releases of the software system, enabling the evaluation of our hypotheses as the project evolved.

Meneely et al. (2014) evaluated the effectiveness of code review and the validity of Linus' laws when applied to vulnerabilities. Specifically, the authors evaluated the validity of the statement "many eyes make all the bugs shallow" as applied to vulnerabilities. The results from this study showed that, in contrast with traditional bugs, vulnerabilities were still missed by many reviewers, indicating an intrinsic difference in the two groups that we aim to clarify. We have refreshed the data set from this previous study by including four additional releases, recollecting commit logs, code reviews, and vulnerability entries, and introducing bug report data.

Prior studies have shown the potential use of logistic regression analysis of traditional software metrics (Tegarden et al. 1992), and defects, to predict future software flaws. A study by Chen et al. (2012) showed the application of logistic regression analysis to various static and historical defect metrics to explain future software defects. This study considered the defect history of each topic and evaluated the probability of future defects. We borrowed the historical analysis aspect from their approach and applied it to explain future vulnerabilities. Shihab et al. (2011) evaluated different software metrics looking for patterns that led to high-impact defects. These high-impact defects break the functionality of production software systems and cause a negative impact on customer satisfaction. The authors used logistic regression to predict the probability that a file contained a high-impact defect. While these studies used logistic regression analysis to build a model capable of predicting future defects, we used logistic regression analysis to evaluate the strength of variables in predicting vulnerability likelihood.

Other Studies have evaluated the occurrence of security-related bugs through the analysis of source control repositories using static analysis tools. Mitropoulos et al. (2013) studied the characteristics of security bugs in relation to other software bugs in Maven repositories. The results from this study indicated that security bugs did not have a recognizable pattern and that further investigation was needed. Mitropoulos et al. (2012) also studied the evolution of security-related bugs by tracking their introduction. The results from this study showed that the number of security-related software bugs increased as the system evolved and that the number was influenced by external software dependencies. These two studies used static analysis tools to identify potential bugs. In contrast, we used actual bug data from a bug tracking system maintained by the project team.

A recent study by Tantithamthavorn et al. (2015) evaluated the impact of mislabeling in the interpretation, and performance of defect prediction models. The authors showed that bug labeling mechanisms are reliable sources of information for the accurate classification of bugs. Their findings increase the relevance of our research as we made use of labels from the bug tracking system in classifying pre-release bugs in the Chromium project.

The de facto standard for quantifying vulnerability severity is using its CVSS base score. Researchers (Bozorgi et al. 2010; Allodi and Massacci 2012b; Allodi et al. 2013) have criticized CVSS for its limitations in reflecting the *actual severity* of an exploited vulnerability. We hypothesize that vulnerability severity may be approximated by the “price” of the vulnerability on the vulnerability market, specifically, the legitimate, bounty-based market. Researchers have studied bounty programs and their effects on software development, security, and vulnerability disclosure (Krishnamurthy and Tripathi 2006; Radianti and Gonzalez 2007; Finifter et al. 2013; Algarni and Malaiya 2014). To our knowledge, there are no studies using vulnerability bounty as an indicator of vulnerability severity.

4 The Chromium Project

Chromium is the open source project on which the Google Chrome web browser is based. The Chromium project consists of over four million SLOC. We chose to study Chromium because of its popularity among the open source developer community and consumers. Furthermore, the record-keeping practices employed by the Chromium project team provides different analysis opportunities that can be explored using software repository mining techniques (Poncin et al. 2011). The data used in the empirical analysis of our research questions were produced by curating data obtained from analyzing code reviews, investigating vulnerability claims, and establishing traceability between the various artifacts.

4.1 Chromium Vulnerabilities (Source: National Vulnerability Database)

The National Vulnerability Database (NVD[1]) is a publicly accessible repository of curated vulnerability data managed by the U.S. government. Common Vulnerabilities and Exposures (CVE[2]) is a similar repository that enumerates the vulnerabilities and exposures. Each entry in the CVE database is assigned a unique identifier called a CVE-ID (or simply CVE). Software vendors request for a CVE when a vulnerability is discovered or exploited. CVEs enable traceability between the CVE, NVD, and any other software repositories that may be maintained by the vendor. Although CVE is the de facto repository for vulnerabilities and exposures, it may not be most accurate when enumerating the vulnerabilities that are *actually* in a product. Fortunately, with each new release of the Google Chrome browser, the Chromium project team reports the CVE of vulnerabilities that have been fixed since the last release. In our study, we manually verified every CVE reported against the Chromium project and filtered out those that were not acknowledged by the Chromium project team. For each vulnerability that was acknowledged by the Chromium project team, we obtained the CVSS base score (a number between 0.0 and 10.0 that quantifies the severity of a vulnerability) from the respective NVD page. The manual verification also enabled us to establish traceability between the fix for a vulnerability and any associated code reviews and commit(s) to the version control system that contained the fix for the vulnerability. We were able to establish traceability for 703 CVEs to their respective code reviews, and the commit(s) that fixed the associated vulnerability.

We mined 242,635 Git commits, spanning six years of development of the Chromium project. These commits contain vital information such as the files that were added/removed/modified, the lines that were added/removed/modified, the user responsible for authoring the change, and the user responsible for committing the change. In case of the Chromium project, the commit messages contained additional metadata such as an identifier to a code review report that recorded the details of the inspection that the commit was subject to before being accepted into the version control system, and an identifier to a bug report if the commit fixed a bug. The data set used in our study is centered around the commit with code reviews used to tie bugs and vulnerabilities together.

4.2 Chromium Bugs (Source: Google Code)

At the time of this writing, the Chromium project team used Google Code as the bug tracking system. We collected 3,801,444 comments from 374,686 bug reports with 5,825 different labels. An entry in the bug tracking system does not necessarily indicate an actual flaw in the system; system features, and other tasks are in this database as well. The taxonomy of bugs into various categories is facilitated by labels in the Google Code bug tracking system. These labels allow developers to tag bug reports with specific keywords (including the label "bug"). The system uses these labels to categorize the bugs enabling easy filtering of and searching for bugs in the bug repository. Special labels are used to associate critical information with bug reports like priority (Pri), product category (Cr), operating system

[1]https://nvd.nist.gov/.

[2]https://cve.mitre.org/.

Table 1 Version number and release date of the five Chromium releases considered in our study

Version Number	5.0	11.0	19.0	27.0	35.0
Release Date	Jan 2010	Jan 2011	Feb 2012	Feb 2013	Feb 2014

where the bug occurs (OS), possible milestone to release a fix (M-value), and type of bug. Google Code also allows users to tag bugs with arbitrary labels. We manually inspected each of the 5,825 distinct labels used and found no two labels, referring to the same attribute, to have different spellings.

We aggregated the labels and examined the frequency of their occurrence in the bug tracking system. In answering RQ2 (bug categories), we only examined labels that were used over 1,000 times and identified categories that were not specific to Chromium (e.g. the label "stability" has a generic meaning, whereas, the label "Milestone18 Migration" has a specific meaning only in the context of the Chromium project).

4.3 Chromium Bug Bounty (Source: Chrome Releases Blog)

With every release of the Google Chrome browser (and the Google Chrome Operating System), the Chromium project team posted an article to the Chrome Releases Blog[3] detailing the release. Articles tagged with the "Stable updates" label describes releases to the end users, in contrast to other types of releases such as developer release (tagged "Dev updates") and beta release (tagged "Beta updates"). Every stable release article lists the CVE of vulnerabilities the release resolves (if any). If a vulnerability was reported by the developer community or an external researcher, the cash reward (a.k.a. bounty) paid to the contributor through the Chrome Rewards Program is also mentioned.

We manually analyzed every stable release article identifying the bounty amount and the associated CVE. We were able to obtain the bounty amount for 204 of the 703 CVEs considered in this study. A large portion (70 %) of the vulnerabilities were discovered and fixed by the internal Chromium project team, and hence, were not awarded a bounty.

4.4 Chromium Releases (Source: Git)

The data set used in our study contains data segregated into releases. The Chromium project utilizes a rapid release cycle with a major release occurring, approximately, every two months. In a prior work (Meneely et al. 2013), we found that vulnerabilities, on average, remained in a system for approximately two years before being fixed. On similar lines, it is reasonable to consider a vulnerability fixed within a year of a release to have existed in the system for up to one year prior to the release. We conducted our analysis in time segments that are non-overlapping to avoid double counting. Thus, we conducted our study in the context of five major releases that were approximately one year apart. The releases and their respective release dates are shown in Table 1.

Chromium releases are tagged in the project's source code repository. We performed a manual investigation of each specific release to corroborate the release date (shown in Table 1), comparing the actual public release date of the product and the date of the final version commit.

[3]http://googlechromereleases.blogspot.com.

We use the phrase “pre-release bugs” to refer to bugs that were fixed prior to a particular release. The “pre-release” qualifier that we have chosen to use may not be entirely accurate in the context of Chromium. Users may have been exposed to some of the bugs in some major release owing to the rapid release cycle that the Chromium project employs. We did not find any consistent labels assigned to bugs that may have helped differentiate pre-release bugs from post-release bugs. Thus, the qualification of bugs as pre-release is relative to our selection of five major releases and may not be used to indicate that the bugs were never part of a production release. Similarly, we use the phrase “post-release vulnerabilities” to refer to vulnerabilities that were fixed after a particular release.

We ensured that no overlap existed between pre-release bugs and post-release vulnerabilities when enumerating the two populations for a particular release. As shown in Fig. 1, bugs between releases 5.0 and 11.0 are used exactly once: in the analysis of release 11.0. The only overlap is between pre-release security bugs and post-release vulnerabilities: these groups may not entirely be the same as some vulnerabilities were not recorded in the bug tracking system and some security bugs were never released to production.

All of the statistical analyses in our study were performed two way: using data collected from each of the five Chromium releases we examined and using the entire data set (i.e. data from all five releases taken together). We present the overall analysis to provide an overview of the results, and the individual release analysis to examine how Chromium evolved over time.

5 Metrics

We collected two groups of file level metrics in our study, they are: (a) bug metrics and (b) review experience metrics. The Chromium project is composed of files written in over 30

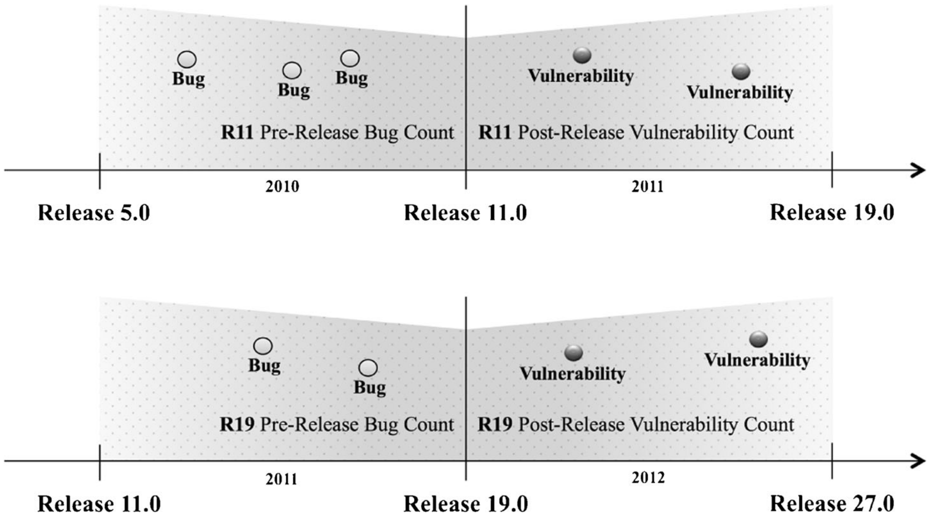


Fig. 1 Release timeline explaining bug and vulnerability selection criteria (For example, in Release 11.0, the number of pre-release bugs is equal to the number of bug reports opened in (roughly) 2010. Similarly, the number of post-release vulnerabilities is equal to the vulnerabilities reported in the year 2011)

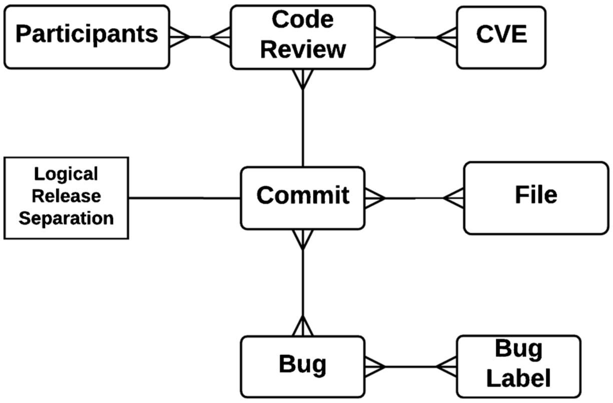


Fig. 2 Conceptual diagram of the Chromium data set

different programming languages. We restricted our study to files containing C/C++ source code only (identified by file extension).

The metrics were collected from each of the five Chromium releases and persisted to a database using an application, called `chromium-history`, implemented in Ruby and Ruby on Rails ActiveRecord library. `chromium-history` was configured to build nightly, integrating changes and executing long running analysis scripts. The metric values contained in the database were analyzed using R (R Core Team 2015). The source code of `chromium-history` is open-source and is available at `https://github.com/andymeneely/chromium-history`. Additionally, the entire Chromium data set used in our study has been made public and may be downloaded as a CSV from `https://gist.github.com/nuthanmunaiah/bc8f49e8fb77210631e4ffa8b34c263b`. The conceptual data diagram depicting the relationship between various entities used in composing the data set is shown in Fig. 2. In the subsections that follow, we describe the metrics collected in our study in greater detail.

5.1 Indicator Metric

The central theme in our study is the investigation of the relationship between certain metrics and the likelihood of a file requiring a fix for a post-release vulnerability. The most important aspect in such an investigation is the ability to identify the files that were historically vulnerable. We used a binary valued metric, called **vulnerable**, to separate source code files into "vulnerable" and "neutral" populations. In the context of a specific Chromium release, vulnerable is true if a file was fixed for a post-release vulnerability within a year *after* the particular release, false otherwise. A file was considered to have been fixed for a post-release vulnerability if a commit to the file contained an identifier to a code review report that facilitated the inspection of a vulnerability-fixing change to the file.

5.2 Bug Metrics

We collected several file level bug metrics that quantify the number of pre-release bugs in a source code file. The pre-release bug count of a specific file is the number of commits to the file having a bug report identifier in the commit message. Only bug reports opened within a year prior to a specific release date were considered when collecting pre-release bug count

Table 2 Description of the file level bug metrics

Metric	Description
num-pre-bugs	Number of bugs reported within a year *before* a selected release.
num-pre-build-bugs	Number of bugs labeled as *"build"* reported within a year *before* a selected release.
num-pre-compatibility-bugs	Number of bugs labeled as *"type-compat"* reported within a year *before* a selected release.
num-pre-features	Number of bugs labeled as *"type-feature"* reported within a year *before* a selected release.
num-pre-regression-bugs	Number of bugs labeled as *"type-bug-regression"* reported within a year *before* a selected release.
num-pre-security-bugs	Number of bugs labeled as *"type-bug-security"* reported within a year *before* a selected release.
num-pre-stability-bugs	Number of bugs labeled as *"stability-crash"* reported within a year *before* a selected release.
num-pre-tests-fails-bugs	Number of bugs labeled as *"cr-tests-fails"* reported within a year *before* a selected release.

of a file. Additionally, we made use of the Chromium bug labels to compute the bug count of different types of bugs. We created seven bug categories corresponding to seven most commonly used vulnerability-relevant labels. The bug categories used in our study are build, compatibility, features, regression, security, stability, and test failure. The bug metrics and their respective descriptions are presented in Table 2.

Files that were neither fixed for a bug nor a vulnerability were removed from the data set since no inferences could be made about their quality or security. The total number of files (and the percentage of files that were vulnerable) in each of the five Chromium releases before (Pre-process) and after (Post-process) the removal of files with no history of bug or vulnerability fix are shown in Table 3.

Table 3 Number of C/C++ source code files (percentage of which were vulnerable) in five Chromium releases before and after removal of files that were neither fixed for a bug nor a vulnerability

Release	Num. of Files (% vulnerable)		% Removed
	Pre-process*	Post-process*	
Release 5.0	9,142 (2.55 %)	2,276 (11.08 %)	75.10 %
Release 11.0	17,005 (3.35 %)	4,760 (12.80 %)	72.01 %
Release 19.0	21,818 (1.40 %)	6,826 (4.61 %)	68.71 %
Release 27.0	30,087 (0.98 %)	8,451 (3.58 %)	71.91 %
Release 35.0	35,871 (0.28 %)	8,125 (1.23 %)	77.35 %

*Pre- and Post-process correspond to before and after the removal of files that were neither fixed for a bug nor a vulnerability, respectively

Table 4 Spearman's ρ for the strongest correlation between pre-release bug category variables

Metric	Spearman's ρ in Release					
	5.0	11.0	19.0	27.0	35.0	Overall
num-pre-build-bugs	−0.12	−0.18	−0.23	−0.29	−0.13	−0.23
num-pre-compatibility-bugs	0.15	0.08	0.08	0.05	0.03	0.07
num-pre-features	−0.12	−0.18	−0.23	−0.29	−0.32	−0.23
num-pre-regression-bugs	0.19	0.29	0.19	0.13	−0.32	0.13
num-pre-security-bugs	0.15	0.18	0.15	0.11	−0.12	0.10
num-pre-stability-bugs	0.19	0.29	0.19	0.13	0.13	0.13
num-pre-tests-fails-bugs	0.05	0.09	0.06	0.04	0.03	0.04

On average, about 73 % of the files had no history of a bug or a vulnerability fix. As seen in the Table, the pre-process number of files between subsequent releases is increasing by an average rate of 43 %. Given such a rapid growth in the project there may have not been enough time for a bug to be uncovered in most files.

The validity of our analysis depends on the ability to distinguish between the various (seemingly) independent metrics/variables. If two or more variables are highly correlated (i.e. explained the same aspect of the data), the analysis may be affected by the redundancy. Therefore, we performed correlation analysis by computing the Spearman's rank correlation coefficient (or Spearman's ρ) between the seven pre-release bug category metrics to identify and remove any variables that presented high co-linearity. The results from the correlation analysis indicated that the pre-release bug category variables had a negligible-to-low correlation with one other. In Table 4, we present the value of Spearman's ρ for the *strongest correlation* between a particular bug category variable and the rest. Even the strongest correlation values were considerably small indicating weak correlation between the various bug category variables. Thus, we concluded that there was no co-linearity among the pre-release bug category variables and that each variable explained an aspect of the data that another variable did not.

5.3 Review Experience Metrics

Developers acquire different types of experience in their role as architects, designers, programmers, testers, and reviewers. Experience acquired in one role may become useful in another. For example, a developer who has resolved several stability bugs becomes familiar with the patterns that lead to this type of failure, making him/her a good asset in stability code reviews. We restrict our study to metrics that quantify the developers' experience in reviewing source code changes. We collected review experience metrics that identify the percentage of reviewers, experienced in reviewing a specific kind of bugs, involved in inspecting a file.

A developer is said to be experienced in reviewing a specific type of bug if she/he had participated in a code review that inspected the fix for that specific type of bug. For a given file, we aggregated the average number of reviewers experienced in reviewing different types of bugs over a release time span. We did not collect experience reviewing bugs labeled

Table 5 Description of the file level review experience metrics

Metric	Description
build-experience	Percentage of reviewers, experienced in reviewing fixes for bugs labeled *"build"*, involved in inspecting a file.
compatibility-experience	Percentage of reviewers, experienced in reviewing fixes for bugs labeled *"type-compat"*, involved in inspecting a file.
security-experience	Percentage of reviewers, experienced in reviewing fixes for bugs labeled *"type-bug-security"*, involved in inspecting a file.
stability-experience	Percentage of reviewers, experienced in reviewing fixes for bugs labeled *"stability-crash"*, involved in inspecting a file.
test-fail-experience	Percentage of reviewers, experienced in reviewing fixes for bugs labeled *"cr-tests-fails"*, involved in inspecting a file.

as either *feature* or *regression* as these may be considered as traditional changes, and we assumed that every reviewer would have some experience reviewing these types of bugs. The various review experience metrics and their respective descriptions are presented in Table 5.

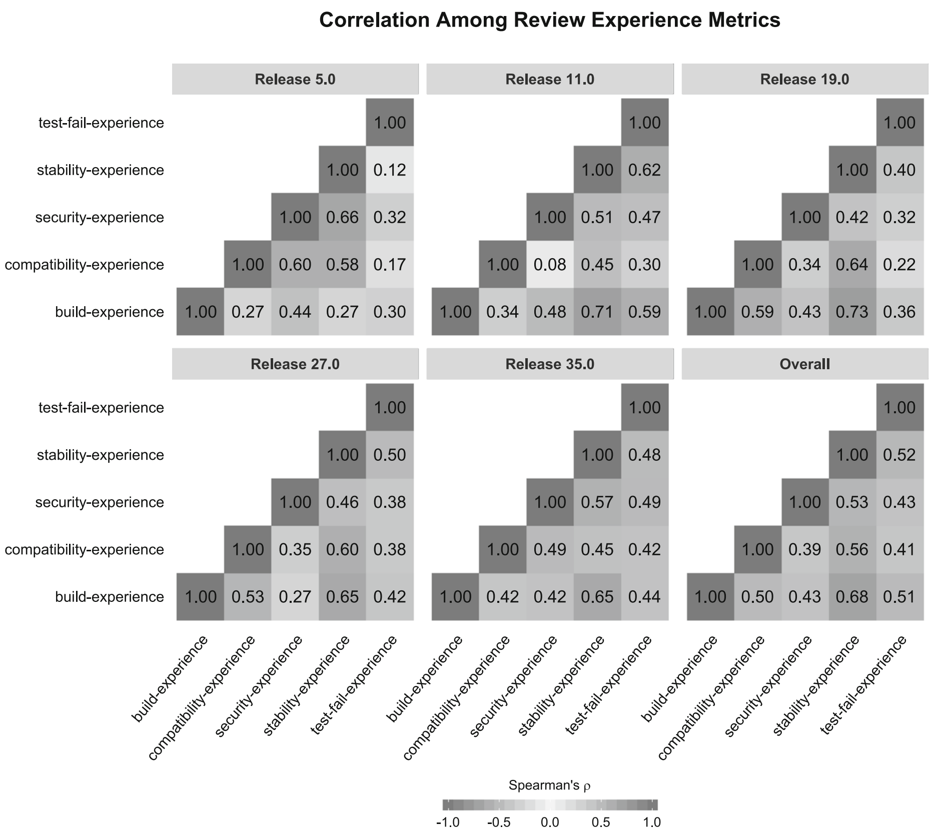
Files that were never subject to a code review were removed from the data set since the review experience metrics could not be collected from such files. The total number of files (and the percentage files that were vulnerable) in each of the five Chromium releases before (Pre-process) and after (Post-process) the removal of files with no history of being reviewed is presented in Table 6. An interesting observation from Table 6 is that the percentage of files without a code review is decreasing with each release, showing Chromium's commitment to ensuring that all changes are reviewed.

As with the bug metrics, we performed correlation analysis by computing Spearman's ρ between the review experience metrics. Unlike the bug category metrics, the correlation

Table 6 Number of C/C++ source code files (percentage of which were vulnerable) in five Chromium releases before and after removal of files that were never subject to a code review

Release	Num. of Files (% vulnerable)		% Removed
	Pre-process*	Post-process*	
Release 5.0	9,142 (2.48 %)	7,477 (2.97 %)	18.21 %
Release 11.0	17,005 (3.25 %)	14,919 (3.59 %)	12.27 %
Release 19.0	21,818 (1.38 %)	21,678 (1.39 %)	0.64 %
Release 27.0	30,087 (0.97 %)	29,900 (0.98 %)	0.62 %
Release 35.0	35,871 (0.28 %)	35,210 (0.28 %)	1.84 %

*Pre- and Post-process correspond to before and after the removal of files that were never subject to a code review, respectively



Release 5.0	build-experience	compatibility-experience	security-experience	stability-experience	test-fail-experience
test-fail-experience					1.00
stability-experience				1.00	0.12
security-experience			1.00	0.66	0.32
compatibility-experience		1.00	0.60	0.58	0.17
build-experience	1.00	0.27	0.44	0.27	0.30

Release 11.0	build-experience	compatibility-experience	security-experience	stability-experience	test-fail-experience
test-fail-experience					1.00
stability-experience				1.00	0.62
security-experience			1.00	0.51	0.47
compatibility-experience		1.00	0.08	0.45	0.30
build-experience	1.00	0.34	0.48	0.71	0.59

Release 19.0	build-experience	compatibility-experience	security-experience	stability-experience	test-fail-experience
test-fail-experience					1.00
stability-experience				1.00	0.40
security-experience			1.00	0.42	0.32
compatibility-experience		1.00	0.34	0.64	0.22
build-experience	1.00	0.59	0.43	0.73	0.36

Release 27.0	build-experience	compatibility-experience	security-experience	stability-experience	test-fail-experience
test-fail-experience					1.00
stability-experience				1.00	0.50
security-experience			1.00	0.46	0.38
compatibility-experience		1.00	0.35	0.60	0.38
build-experience	1.00	0.53	0.27	0.65	0.42

Release 35.0	build-experience	compatibility-experience	security-experience	stability-experience	test-fail-experience
test-fail-experience					1.00
stability-experience				1.00	0.48
security-experience			1.00	0.57	0.49
compatibility-experience		1.00	0.49	0.45	0.42
build-experience	1.00	0.42	0.42	0.65	0.44

Overall	build-experience	compatibility-experience	security-experience	stability-experience	test-fail-experience
test-fail-experience					1.00
stability-experience				1.00	0.52
security-experience			1.00	0.53	0.43
compatibility-experience		1.00	0.39	0.56	0.41
build-experience	1.00	0.50	0.43	0.68	0.51

Fig. 3 Spearman's ρ among review experience metrics

among the review experience metrics was nuanced warranting the need to present all values of Spearman's ρ individually. The results from the correlation analysis are presented in Fig. 3. The weak-to-moderate positive correlation among **compatibility-experience**, **security-experience**, and **stability-experience** is not surprising since there is an overlap between the skills required in reviewing compatibility-, security-, or stability-related bugs. The correlation analysis also revealed moderate-to-strong positive correlation between **build-experience** and **stability-experience**. In order to eliminate the redundancy, we chose to use **build-experience** to represent both metrics.

6 Analysis Approach and Results

In our investigation of the relationship between bugs and vulnerabilities, we conducted an in-depth exploratory statistical analysis using the file level metrics collected. In the subsections that follow, we describe the methodology used in analyzing the data in the context of each of our research questions, and present the results obtained from the analysis.

6.1 RQ1: *Are Source Code Files Fixed for Bugs Likely to be Fixed for Future Vulnerabilities?*

6.1.1 Motivation

Our goal in this research question was to evaluate, in a broad sense, if files that were frequently fixed for bugs also had a higher likelihood of being fixed for a vulnerability. This research question serves as an overall measure of the relationship between quality and security and does not delve into other properties of bugs such as its category (as RQ2 does).

6.1.2 Analysis

As an initial step in the analysis, we visualized the distribution of the num-pre-bugs metric collected from vulnerable and neutral source code files. The visualization, shown in Fig. 4, enables a *qualitative assessment* of the difference (or lack thereof) in the distribution of the metric values collected from vulnerable and neutral source code files. Intuitively, the higher the overlap between the distributions, the less powerful a metric is in differentiating between vulnerable and neutral files.

We used the MWW test to *quantitatively* investigate if a statistically significant association exists between the number of pre-release bugs fixed in a file and the likelihood of a file being fixed for a vulnerability. Furthermore, we examined the magnitude of effect of number of pre-release bugs on the populations using the Cohen's d statistic (Cohen 2013). The approach to computing the Cohen's d statistic assumes that the populations being compared are normally distributed. The distribution of num-pre-bugs is not normal. Therefore, we apply a logarithmic transformation on the data to approximate near-normality (as shown in Fig. 4 which has the metric value on a log scale) before computing the Cohen's d statistic.

The results from the MWW test in each of the five Chromium releases and in the overall data set are presented in Table 7 with all results being statistically significant at $p < 0.05$. Table 7 also contains the value of the Cohen's d statistic computed in each of the five Chromium releases and in the overall data set.

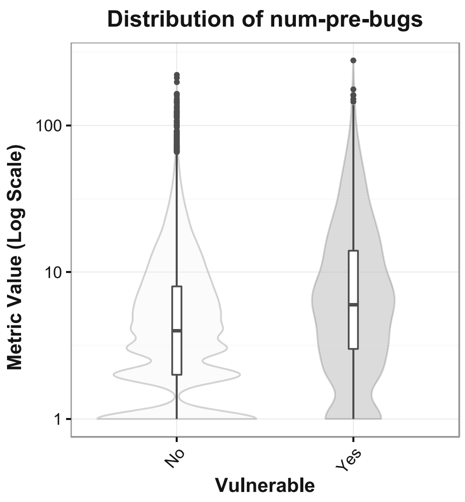


Fig. 4 Distribution of number of pre-release bugs collected from vulnerable and neutral files

Table 7 MWW test and Cohen's *d* effect size evaluation results for num-pre-bugs and SLOC in five Chromium releases (All results were statistically significant at $p < 0.05$)

Release	Metric	Median		Cohen's *d*
		Vulnerable	Neutral	
Release 5.0	num-pre-bugs	6	3	0.74
	SLOC	200	10	0.53
Release 11.0	num-pre-bugs	4	2	0.57
	SLOC	193.5	70	0.78
Release 19.0	num-pre-bugs	4	3	0.29
	SLOC	202	82	0.63
Release 27.0	num-pre-bugs	5	3	0.36
	SLOC	228.5	10	0.63
Release 35.0	num-pre-bugs	6	4	0.39
	SLOC	307	11	0.61
Overall	num-pre-bugs	5	3	0.34
	SLOC	205	96	0.58

The results show that vulnerable files have a higher median value for num-pre-bugs than neutral files. Although the difference in the median value of num-pre-bugs between vulnerable and neutral files is small, it is consistent across all releases. As mentioned earlier, the Cohen's *d* statistic is useful in gauging the strength of an association (if any as revealed by MWW). Applying the heuristics from the Cohen's *d* literature (Cohen 1992) to the value of Cohen's *d* statistic computed for num-pre-bugs, the effect observed was considered to be small-to-medium.

We know that vulnerable files tend to be larger in size (measured in terms of SLOC) than neutral files (Zimmermann et al. 2010; Shin et al. 2011). We used the MWW test to validate this assertion in the Chromium project. The results from the test are also presented in Table 7 with all results being statistically significant at $p < 0.05$. The results confirm the assertion from related and prior work that vulnerable files are larger in size than neutral files. Furthermore, the effect of the association observed, as quantified by Cohen's *d*, was consistently medium across all releases and in the overall data set.

We note that the heuristic used to interpret Cohen's *d* depends on the area of research and the research context. For instance, Zakzanis (2001) suggested that a value of $d \geq 3.0$ in neuropsychological research would correspond to a "large" effect. In any case, the Cohen's *d* values obtained in our study indicated that the effect of number of pre-release bugs in distinguishing the vulnerable and neutral populations was relatively weak. That is, a large number of neutral files had many bugs; and a large number of vulnerable files had few bugs.

Broadly speaking, files with a history of bugs are likely to be fixed for vulnerabilities in the future. However, there are many counterexamples to this notion leading to a weak association overall.

6.2 RQ2: *Are Some Types of Bugs (Including Features) more Closely Related to Vulnerabilities than Others?*

6.2.1 Motivation

In RQ1, we learned that there is a statistically significant yet weak association between number of pre-release bugs and the likelihood of a future vulnerability. However, bugs come in many different forms. While some bugs may be related to compatibility across operating systems, others may be related to overall stability of a system. Consequently, some bugs may have a higher likelihood of foreshadowing future vulnerabilities. For example, source code with a history of bugs related to compatibility between 32-bit and 64-bit operating system may be more likely to have integer overflow problems that could become exploitable in the future. In order to investigate the potential dissimilarity between bug categories, we analyzed the strength of association between multiple bug categories and the likelihood of future vulnerabilities. Our goal in this research question was to identify trends that may imply that some bug categories foreshadow future vulnerabilities more than others.

6.2.2 Analysis

Similar to RQ1, an initial step of the analysis was to visualize and compare the distribution of the various bug category metrics collected from population of vulnerable and neutral source code files. The visualizations are shown in Fig. 5. From the Figure, we observed that the metrics' distribution, even on a log scale, was not normal. Furthermore, the overlap between the metrics' distribution for vulnerable and neutral files is considerably high, *qualitatively* indicating metrics' limited ability in differentiating between the two types of files.

Although the distributions enable a visual interpretation of association (or lack thereof) between the bug category metrics and vulnerability likelihood, we performed quantitative investigation to substantiate the hypothesis. We used logistic regression analysis to build a series of models, each with one or more bug categories as independent variable(s), and compared their relative quality to ascertain the prominence of certain bug categories in the prediction of vulnerability likelihood. Since we know that SLOC of a file is a good predictor of the likelihood of the file being vulnerable (Zimmermann et al. 2010; Shin et al. 2011), we built a *base model* using only SLOC as the independent variable. The logistic regression models were compared using two statistical tests: **model goodness of fit** (how well an estimated regression model fits the data) and **model performance** (how well the estimated regression model performs when applied to out of sample prediction). The model goodness of fit test results were interpreted relative to the base model i.e. the goodness of fit of each candidate model was compared to that of the base model to gauge the improvement in fit (if any). The model performance test, on the other hand, was conducted using 10 repetitions of 10-fold cross-validation. We must point out that the logistic regression models were built to *compare* various bug categories and not to present an overall vulnerability prediction model as done in prior literature (Gegick et al. 2009; Shihab et al. 2011; Chen et al. 2012; Meneely et al. 2014).

We logically grouped the seven bug categories to create four bug category groups: build, features, security, and stability. Although features may not conceptually be considered as bugs, the Chromium project tracks features in the same tracking system as the bugs; hence,

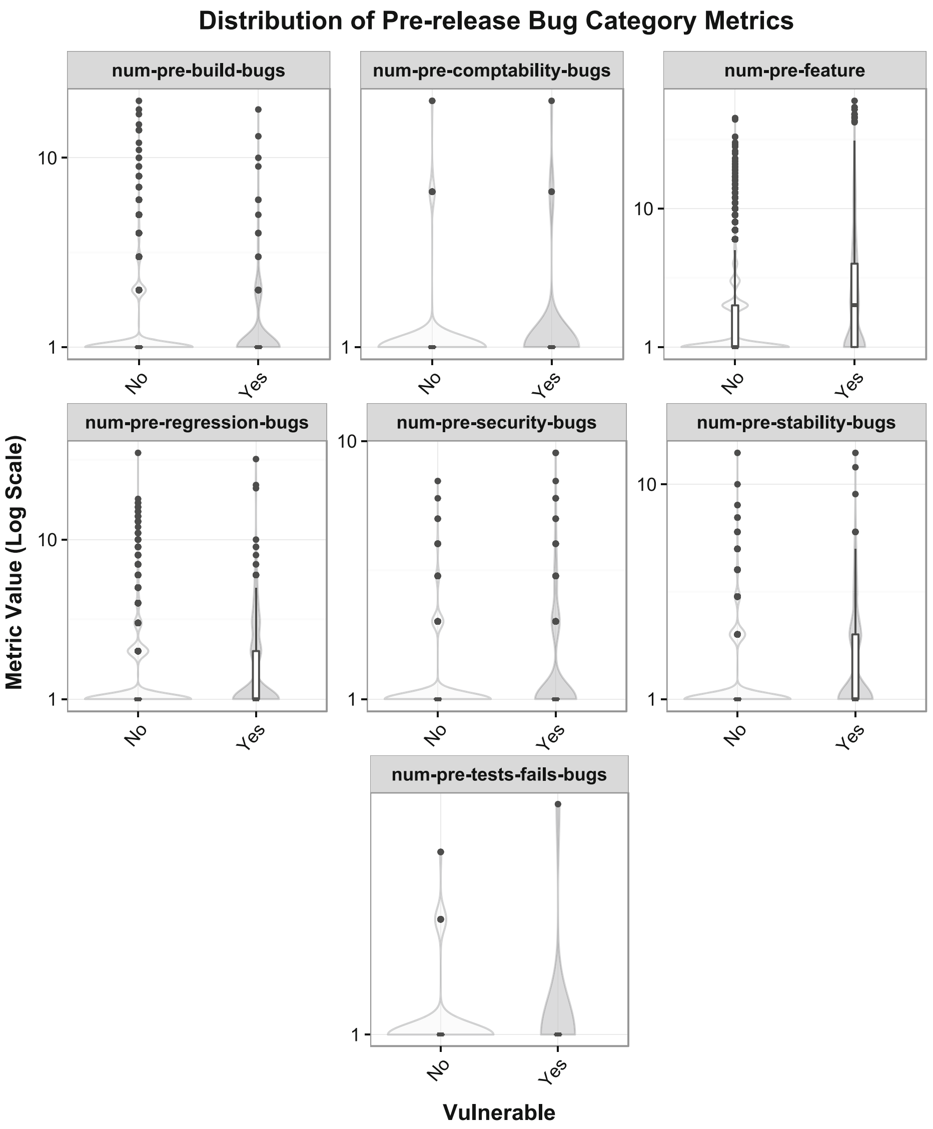


Fig. 5 Distribution of bug category metrics collected from vulnerable and neutral files

we consider features to be a bug category group. For each of these bug category groups, we built a logistic regression model by including the base model (i.e. SLOC predicting vulnerability likelihood) and metric(s) from a bug category group. In addition to the base model and the bug category group models, we also built a logistic regression model in which SLOC and the aggregate number of pre-release bugs (i.e. num-pre-bugs) was used to predict vulnerability likelihood. This last model (i.e. SLOC and num-pre-bugs predicting vulnerability likelihood) serves as a *reference model* when assessing if the categorization of bugs had any positive impact on vulnerability likelihood when compared to the aggregate metric—num-pre-bugs. The reference model was also used to reaffirm the result from RQ1 that the number of pre-release bugs is weakly associated with vulnerability likelihood.

Table 8 Specification of the base, reference, and four bug category group logistic regression models

Base Model	
Model	Specification
fit-sloc	`vulnerable ~ SLOC`
Reference Model	
Model	Specification
fit-num-pre-bugs	`vulnerable ~ SLOC + num-pre-bugs`
Bug Category Group Models	
Model	Specification
fit-build	`vulnerable ~ SLOC + num-pre-build-bugs + num-pre-tests-fails-bugs`
fit-features	`vulnerable ~ SLOC + num-pre-features`
fit-security	`vulnerable ~ SLOC + num-pre-security-bugs`
fit-stability	`vulnerable ~ SLOC + num-pre-stability-bugs + num-pre-compatibility-bugs + num-pre-regression-bugs`

Table 8 lists the specification of the base model, the reference model, and the four bug category group models.

The coefficient of the variables in each of the six logistic regression models when fit to the entire data set (i.e. all five Chromium releases taken together) is shown in Table 9. All coefficients shown are statistically significant at $p < 0.05$ and those that were not are shown as dash (-) in the Table. We used **release** as an ordinal-valued variable in each of the six logistic regression models to understand the contribution of data originating from each of the five Chromium releases to vulnerability likelihood. In R, the ordinal-valued variables are replaced with multiple binary-valued dummy variables. In effect, the contribution of data originating from each of the five Chromium releases to vulnerability likelihood is encapsulated in the intercept and thus there are five different values of the intercept shown in Table 9.

6.2.3 Model Goodness of Fit

As mentioned earlier, the model goodness of fit is a way to evaluate how well an estimated regression model fits the data. By comparing the model goodness of fit of multiple regression models with varying independent variables, we may ascertain the set of independent variables that best describe the data. We assessed the model goodness of fit using two metrics: (a) AIC and (b) D^2. The absolute value of AIC and D^2 may not be used directly to assess the model goodness of fit. Instead, the values of AIC and D^2 computed for a candidate model is compared against those computed for a base model. The model that exhibits the maximum decrease in AIC and the maximum increase in D^2 is considered to fit the data better than the base model.

The value of AIC and D^2 computed for each of six logistic regression models are presented in Table 10. The results indicated an overall improvement in the model goodness of fit, measured relative to the base model (i.e. fit-sloc), with the inclusion of pre-release bug metrics. The models that exhibited the best improvement in model goodness of fit have their AIC and D^2 values shown in boldface. The **fit-security** model exhibited the

Table 9 Coefficient of variables in base, reference, and four bug category models (- is shown in place of coefficients that were not statistically significant at $\alpha = 0.05$)

Variable	Models					
	fit-sloc	fit-num-pre-bugs	fit-build	fit-features	fit-security	fit-stability
$(\text{intercept})_{5.0}$	−4.8918	−4.8036	−4.9201	−4.7963	−4.6699	−4.8523
$(\text{intercept})_{11.0}$	−4.6194	−4.5233	−4.6370	−4.5265	−4.3751	−4.5570
$(\text{intercept})_{19.0}$	−5.7246	−5.6483	−5.6521	−5.7062	−5.4284	−5.6684
$(\text{intercept})_{27.0}$	−6.0671	−6.0011	−6.0508	−6.0129	−5.7984	−5.9882
$(\text{intercept})_{35.0}$	−7.1905	−7.1362	−7.2480	−7.1785	−6.9153	−7.1025
sloc	0.5380	0.4770	0.5543	0.5724	0.4614	0.5187
num-pre-bugs		0.1385				
num-pre-build-bugs			−0.4842			
num-pre-tests-fails-bugs			–			
num-pre-features				−0.3724		
num-pre-security-bugs					1.1935	
num-pre-stability-bugs						–
num-pre-compatibility-bugs						0.8362
num-pre-regression-bugs						–

highest improvement in model goodness of fit when compared to other models with a 1.65 % decrease in AIC and a 12.66 % increase in D^2. Furthermore, the bug category group models—**fit-build**, **fit-features**, and **fit-security**—outperformed the reference model, **fit-num-pre-bugs**, indicating that the categorization of bugs may be necessary when attempting to understand vulnerability likelihood through the analysis of bugs.

The value of AIC and D^2 computed for each of the six logistic regression models built with data from each of the five Chromium releases are shown in Table 18 in the Appendix. In release 5.0, no bug category group model outperformed the reference model **fit-num-pre-bugs**. In releases 11.0 and 19.0, **fit-features** exhibited a better goodness of fit when compared to the base and reference models. The later releases (specifically 27.0 and 35.0) is where **fit-security** begins to improve in goodness of fit.

6.2.4 Model Performance

As mentioned earlier, model performance is a way to evaluate how well an estimated regression model performs when applied to out of sample prediction. We assessed model

Table 10 Model goodness of fit metrics for base, reference, and bug category group models

Model	AIC	AIC Decrease	D^2	D^2 Increase
fit-sloc	10360	–	0.1165	–
fit-num-pre-bugs	10345	−0.14 %	0.1179	1.25 %
fit-build	10326	−0.33 %	0.1197	2.78 %
fit-features	10302	−0.56 %	0.1216	4.38 %
fit-security	**10189**	−1.65 %	**0.1312**	12.66 %
fit-stability	10348	−0.12 %	0.1180	1.30 %

Table 11 Model performance metrics for base, reference, and bug category group models

Model	Precision	Recall	F-measure	AUC
fit-sloc	0.1189	0.6672	0.2019	76.34 %
fit-num-pre-bugs	0.1054	**0.7308**	0.1843	76.21 %
fit-build	0.1130	0.6967	0.1945	76.80 %
fit-features	0.1094	0.7151	0.1898	77.01 %
fit-security	**0.1230**	0.6874	**0.2087**	**77.82 %**
fit-stability	0.1121	0.6997	0.1932	76.33 %

performance using four metrics: (a) precision, (b) recall, (c) F-measure, and (d) Area under the ROC (AUC). We used 10 repetitions of 10-fold cross-validation to compute the model performance metrics. Similar to the model goodness of fit assessment, we compared the value of model performance metrics computed for a candidate model to those computed for the base model.

The value of precision, recall, F-measure, and AUC computed for each of six logistic regression models are presented in Table 11. The models that exhibited the best improvement in model performance have their precision, recall, F-measure, or AUC values shown in boldface. Firstly, the *reference model*, **fit-num-pre-bugs**, exhibited better recall when compared to the *base model*, **fit-sloc**. Among the bug category group models, **fit-security** exhibited the best performance when compared to the *base model*. Moreover, **fit-build**, **fit-features**, and **fit-stability** exhibited the same or marginally better performance when compared to the *reference model*.

The value of precision, recall, F-measure, and AUC computed for each of six logistic regression built with data from each of the four Chromium releases are shown in Table 19 in the Appendix. Although we built a model with data from release 35.0, we could not assess its performance since we did not have data from a release that was chronologically adjacent to release 35.0 in our data set. The models that exhibited the best improvement in model performance have their precision, recall, F-measure, or AUC values shown in boldface. Firstly, the *reference model*, **fit-num-pre-bugs**, exhibited better recall when compared to the *base model*, **fit-sloc**, in all but release 5.0. Among the bug category group models, **fit-security** exhibited the best performance when compared to the *base model*. Moreover, **fit-build**, **fit-features**, and **fit-stability** exhibited the same or marginally better performance when compared to the *reference model*.

6.2.5 Interpretation

Aggregating the observations from model goodness of fit and model performance tests, we observed traditional non-security bugs (i.e. build, features, and stability) to have a positive yet weak association with vulnerabilities. Among the bug category groups, features and security bugs had the strongest positive effect on the overall model quality, however, the improvement was marginal. Moreover, the predictive power of the various bug category group models were limited when compared to prior literature (Shin et al. 2011; Gegick et al. 2009; Cruz and Ochimizu 2009; Neuhaus et al. 2007). A particularly interesting aspect of the result is that some of the bug categories with weaker association with vulnerabilities are indeed known to be related to security properties. For example, stability is key to ensuring availability (i.e. preventing denial-of-service attacks). One might assume that stability problems in files' past may lead to security problems in the future, however, we did not find empirical evidence supporting this assumption.

The weakness in association indicates that bugs and vulnerabilities are empirically dissimilar groups and that, even with a granular categorization of bugs into types, it may be difficult to identify future vulnerabilities based solely on the pre-release bug history.

History of pre-release bugs, even at a finer level of granularity, has limited ability in indicating the vulnerability likelihood of files.

6.3 RQ3: *Do Source Code Files with Most Bugs Also Have the Most Vulnerabilities?*

6.3.1 Motivation

Our goal in this research question was to simulate and understand the use of pre-release bugs in practice, specifically as an aid in security audits. Stated informally, if we had a perfect bug prediction model that would rank files by defect density, how good would this model be at finding vulnerabilities?

Consider the situation of a last-minute security audit. The thoroughness of the code inspection in such expedited audits may be limited by the resources available. Our goal in this research question is to understand if the "bugginess" of files could be used to prioritize code inspections. In other words, we wanted to evaluate the effectiveness of using "bugginess" as an indicator of files that would later need to be fixed for vulnerabilities.

6.3.2 Analysis

We estimated the "bugginess" of a file using bug density, which we define as the number of pre-release bugs per SLOC (i.e. num-pre-bugs/SLOC). We used *lift curves* to visualize the percentage of vulnerable files accounted for when files were ranked by bug density. The lift curves obtained from each of the five Chromium release are shown in Fig. 6. As an example, in release 11.0, we observed a little over 37.5 % of the post-release vulnerable files were found in the top 50 % of the buggiest files. The ranking scheme is not even better than random guessing (i.e. roughly the diagonal line) and therefore the prediction capabilities of bug density is significantly limited in comparison to many of the other metrics found in the vulnerability prediction literature (Shin et al. 2011; Neuhaus et al. 2007).

Consider these results in the context of our scenario: suppose we have the resources that permit inspecting 20 files that are non-trivial, say 25 lines of code or more (these numbers may be arbitrary, however, we chose them as reasonable simulations of an expedited security audit). If we ranked the files by bug density, then those files would, at best, contain 1.4 % of the vulnerabilities. An optimal top 20 file ranking would account for an average of 12 % of vulnerabilities in Chromium; thus, ranking by bug density is far from optimal.

We conclude that the buggiest files had some, but not many, vulnerabilities. We also attempted an inverse analysis by ranking files based on their vulnerability density (i.e. number of post-release vulnerabilities per SLOC, num-post-vulnerabilities/SLOC) to understand

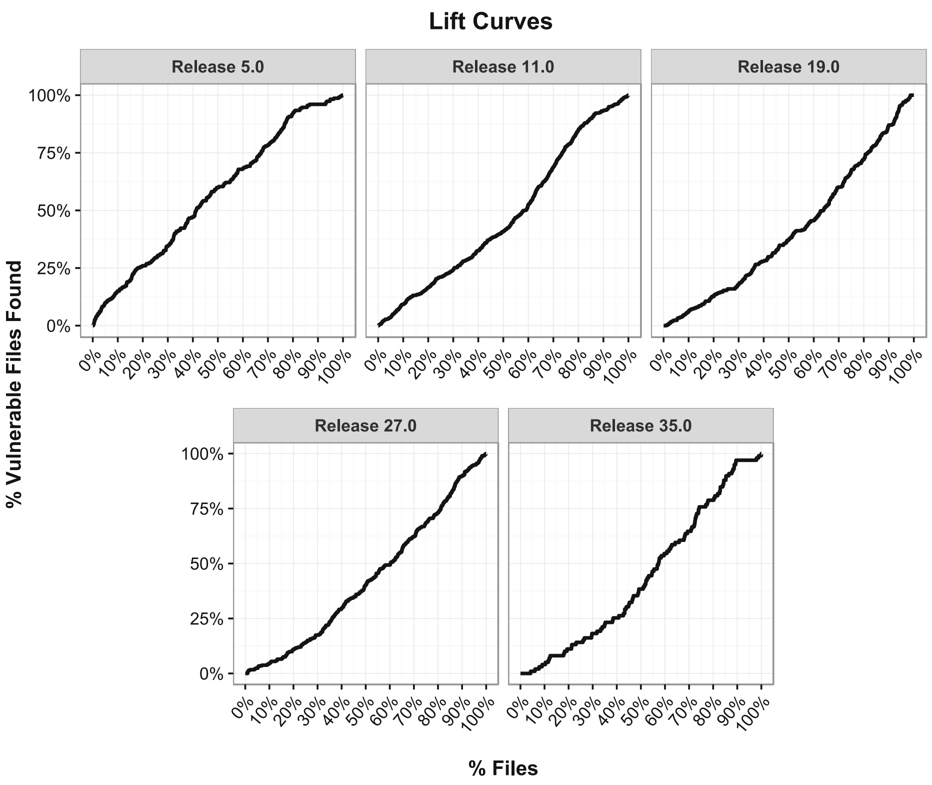


Fig. 6 Lift curve of percentage of vulnerable files found when files are ranked by "bugginess" (i.e. num-pre-bugs/SLOC) in five Chromium releases

if the most vulnerable files were also the most buggiest. We observed that none of the top 20 files with highest vulnerability density appeared in the top 20 files with highest bug density.

A "perfect" bug prediction model would be a poor vulnerability prediction model.

6.4 RQ4: *Are Bugs More Likely to be in Files with High Vulnerability Severity?*

6.4.1 Motivation

Not all vulnerabilities are the same. The *severity* of a vulnerability depends on several contextual details of a system such as its architectural layout, relationship to sandboxes, access requirements, etc. Consequently, some vulnerabilities may be of a higher value to attackers than others. For example, a vulnerability that allows unauthorized access to the contents of a server's memory may be more valuable to an attacker than a vulnerability that causes a software system to crash when a particular kind of file is opened.

We used two measures of severity in this study: Chromium Bounty Award (Bounty) and the Common Vulnerability Scoring System (CVSS) base score. Our goal in this research question was to understand if the files with severe vulnerabilities were also buggy. Our motivation for analyzing two severity metrics comes from our view that CVSS, while widely used, is severely flawed (see CVSS discussion below).

6.4.2 Analysis

Bounty Currently, markets exist on the dark web to buy and sell information on vulnerabilities (Miller 2007; Algarni and Malaiya 2014). In response to this, companies have opted to reward ethical hacking and responsible disclosure of vulnerabilities discovered in their systems with monetary benefits. In the case of Chromium, the Google Application Security team started a bounty program ("Chrome Rewards") to reward security researchers for finding vulnerabilities.

Depending on the severity of the vulnerability, these rewards range from a few hundred (U.S.) dollars to hundreds of thousands of dollars. Not every vulnerability is given a bounty—only ones that are considered to have significant, widespread impact are awarded a bounty. The complex rules for bounty assignments are published and constantly revised by the Chromium development team. Factors that raise the severity of a vulnerability include "sandbox escape", "remote code execution", and "information leak". Higher rewards are also given out for better reports, such as including a proof-of-concept exploit. This element of the bounty program lends credence to bounty reflecting the notion of severity, as an exploit can be proof by existence that a security threat is valid.

Google awarded a total of \$264,805.90 to researchers for 204 of the 703 vulnerabilities in our data set. The maximum bounty awarded for a single vulnerability was \$12,147 and the average reward (when given out) was \$1,298. In cases where a single bounty was awarded for a set of vulnerabilities, we evenly divided the bounty amount among the vulnerabilities. In the case of files fixed for multiple vulnerabilities for which a bounty was awarded, we summed the bounty amount of the vulnerabilities. As a consequence, a vulnerability impacting multiple files could be counted twice.

CVSS CVSS is a severity scoring system that is required for entry into the National Vulnerability Database to get a CVE identifier. Every vulnerability we evaluated in this study had a CVSS score. CVSS is typically entered by the person reporting the vulnerability, but can be revised by the development team over time. The CVSS uses various contextual factors of a vulnerability and its impacts (e.g. "is authentication required for exploit?").

As a caveat, CVSS has been criticized by researchers (Bozorgi et al. 2010; Allodi and Massacci 2012a, b; Allodi et al. 2013; Allodi and Massacci 2014; Younis et al. 2015; Younis and Malaiya 2015), many of the concerns we also share. Our main criticisms are: (a) CVSS scores are reported by the original researcher who may not know enough of the system to make an accurate assessment of severity, (b) the subjectivity of the scoring levels, and (c) the weighting scheme for CVSS is (at the time of this writing) not transparent in its origin. Despite its limitations, CVSS is the de facto standard for rating vulnerability severity. Therefore, we report the CVSS scores owing to its popularity alone.

The average CVSS score across vulnerabilities in our data set was 7.1. In the case of files fixed for multiple vulnerabilities, we averaged the CVSS base score of the vulnerabilities.

Table 12 Spearman's ρ between num-pre-bugs and vulnerability severity metrics—Bounty and CVSS–in five Chromium releases and in the entire Chromium data set

Release	Spearman's ρ	
	Bounty	CVSS
Release 5.0	−0.0214	−0.2061
Release 11.0	–	–
Release 19.0	0.0734	−0.2954
Release 27.0	0.0054	0.1025
Release 35.0	−0.0223	−0.0611
Overall	0.0256	−0.1594

Bounty vs. CVSS Since we have two competing metrics for severity, we examined how they aligned with each other per-vulnerability before translating them to files. The Spearman's rank correlation coefficient between CVSS and bounty (where both existed) was 0.28, which we interpret as considerably weak. Many severe vulnerabilities according to CVSS did not have a significant impact on Chromium's security posture because of sandboxing and various other features. Interestingly, the vulnerability with the highest bounty awarded had a CVSS base score of 6.8 out of 10. Given the complex rules surrounding bounty valuation for Chromium, we found this weak correlation result to be another deterrent for using CVSS base scores.

(Bounty, CVSS) vs. Bugs We computed the Spearman's rank correlation coefficient between number of pre-release bugs (i.e. num-pre-bugs) and the vulnerability severity metrics—bounty and CVSS base score. We only examined files that have more than zero dollars in vulnerability bounties (635 in total). The results from the Spearman's correlation analysis are presented in Table 12 with all results being statistically significant at $p < 0.05$. Release 11.0 had only a single file with vulnerability bounty, so we excluded it from our analysis and is shown as a dash (-) in the Table 12.

Per-release, and overall, these results show that the correlations between the most economically severe vulnerabilities and number of pre-release bugs are extremely weak. When examining this per-release, we see inconsistent results: weak correlations that oscillate directions depending on the release.

Files with vulnerabilities that may be valuable to attackers did not have a corresponding increase in number of bugs.

6.5 RQ5: *Do Source Code Files Reviewed by More Bug Review Experienced Developers Have Fewer Vulnerabilities?*

6.5.1 Motivation

As was shown in Fig. 2, we used code reviews to link vulnerabilities to their respective vulnerability-fixing commits. A typical code review entails a group of developers inspecting potential commits for various types of coding mistakes such as traditional bugs, maintainability issues, design flaws, and security problems.

In the research questions considered thus far, we observed that buggy files and vulnerable files are empirically dissimilar groups, and that only a marginal overlap existed between the two groups. Although there is an overlap, reviewers may still miss vulnerabilities if they are not attuned to recognizing them. Our goal in this research question was to understand if developers' experience in reviewing bugs can be a factor in determining whether or not a vulnerability is prevented by detection in code reviews.

Meneely et al. (2014) have shown that vulnerable files tend to have fewer security-experienced reviewers than neutral files. We re-evaluated their hypothesis in the context of an expanded experience data set. We also collected the review experience of a developer in reviewing specific types of bugs (e.g. build, stability, etc.) to understand if the granular metrics were related to lowered vulnerability likelihood. The review experience metrics used in our study were described in Section 5.3.

6.5.2 Analysis

As with RQ1 and RQ2, the initial step of the analysis was to compare the distribution of the various review experience metrics collected from population of vulnerable and neutral files. The distribution is shown in Fig. 7. The overlap between the distribution of each of the review experience metrics for vulnerable and neutral populations is *qualitative indication* that no single review experience metric is capable of delineating the two populations.

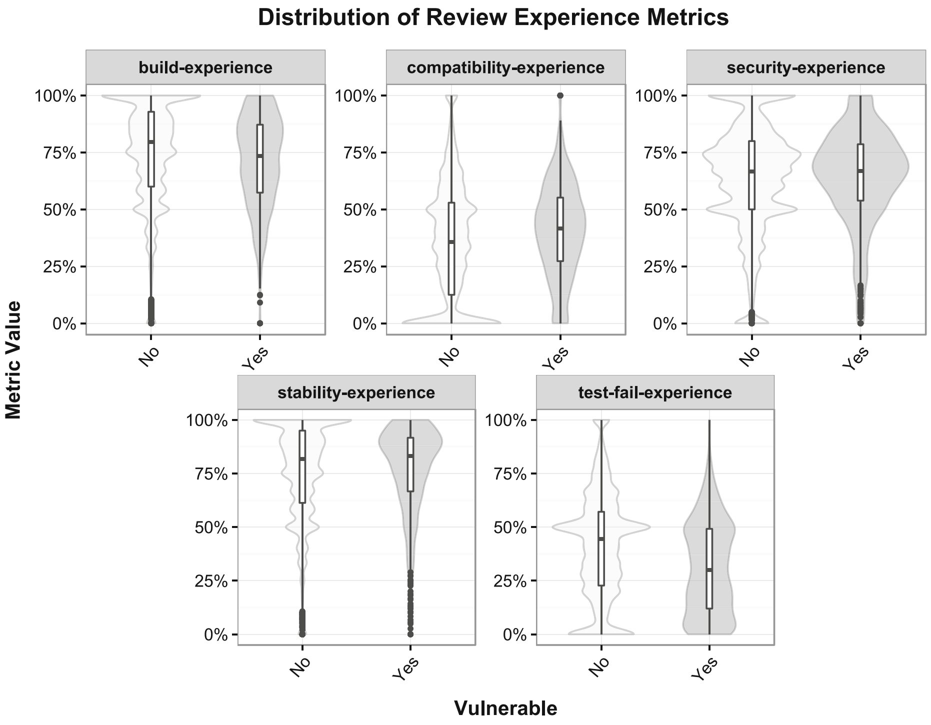


Fig. 7 Distribution of review experience metrics collected from vulnerable and neutral files

We substantiate the qualitative inference with quantitative evidence in a way similar to RQ1 and RQ2. We used MWW test to investigate if a statistically significant association existed between the review experience metrics and the likelihood of a file being fixed for a post-release vulnerability. We used logistic regression to investigate if the inclusion of the review experience metrics improved the goodness of fit and/or performance of a model when compared to a *base model* (i.e. SLOC predicting vulnerability likelihood). We also compared the results obtained from these analyses with the results from RQ2 to understand which of the corresponding review experience and bug category metrics were better predictors of vulnerability likelihood.

We separated the files into vulnerable and neutral populations and used the MWW test to evaluate the difference in the median value of review experience metrics between the two populations. Since we were using MWW test to evaluate four different hypotheses, there is a possibility of multiple hypothesis testing errors. We mitigated this problem in a way similar to prior work by Meneely et al. (2014), in which, a conservative Bonferroni correction was applied to the significance level ($\alpha = 0.05$) by dividing it by the number of independent variables. Thus, the significance level of each individual hypothesis test in our study was $\alpha = 0.05/4 = 0.125$. The MWW results are presented in Table 13 with the statistically significant results having their p-value shown in boldface.

We observed that vulnerable files had fewer build (and, by correlation, stability) and test failure experienced developers that neutral files indicating that having more developers experienced in reviewing build, stability, or test failure type of bugs decreases the likelihood of future vulnerabilities. We also observed a counter-intuitive result that vulnerable files tend to have *more* compatibility experienced developers than neutral files. The counter-intuitiveness may be explained by the fact that achieving compatibility between different processor architectures and operating systems is complex and prone to vulnerabilities. A mitigation strategy may have been to increase the number of compatibility experienced reviewers, however, from the results, we notice that the mitigation may have not been as successful as one would hope.

The results also revealed an interesting insight: neutral files had more build (and, by correlation, stability) experienced developers than security experienced developers. Although intriguing, the result is sound in that a person experienced in different environments gains an awareness of several aspects of the environment (file systems, interfaces, etc.), and with this awareness comes a better understanding of vulnerability patterns.

Further to our association analysis, we used logistic regression analysis to evaluate if the inclusion of review experience metrics improved the quality of a model relative to the base model. We built a series of models, each with SLOC and a review experience metric as the independent variables, and compared the relative quality of the models to ascertain

Table 13 MWW test results for review experience metrics

Metric	Mean		Median		p-value
	Vulnerable	Neutral	Vulnerable	Neutral	
build-experience	0.710	0.751	0.735	0.796	**2.20E-16**
compatibility-experience	0.408	0.353	0.417	0.357	**2.20E-16**
security-experience	0.644	0.637	0.669	0.667	6.83E-01
test-fail-experience	0.310	0.413	0.300	0.444	**2.20E-16**

Table 14 Specification of the base and four review experience logistic regression models

Base Model	
Model	Specification
fit-sloc	`vulnerable ∼ SLOC`
Review Experience Models	
Model	Specification
fit-build	`vulnerable ∼ SLOC + build-experience`
fit-compatibility	`vulnerable ∼ SLOC + compatibility-experience`
fit-security	`vulnerable ∼ SLOC + security-experience`
fit-test-fail	`vulnerable ∼ SLOC + test-fail-experience`

the prominence of certain review experience metrics in the prediction of vulnerability likelihood. We built four logistic regression models with each model including SLOC and one of the four review experience metrics. Table 14 lists the base model and four review experience models with their respective model specifications.

The coefficient of the variables in each of the five logistic regressions models when fit to the entire data set (i.e. all five Chromium releases taken together) is shown in Table 15. All coefficients shown are statistically significant at $p < 0.05$ and those that were not are shown as dash (-) in the Table. Similar to the bug category models from Section 6.2, we used **release** as an ordinal-valued variable in the logistic regression models. The contribution of data originating from each of the five Chromium releases to vulnerability likelihood is encapsulated in the intercept and thus there are five different values of the intercept shown in Table 15.

Similar to RQ2, we used **AIC** and D^2 in evaluating the improvement in the quality of the logistic regression models when compared to the base model (i.e. fit-sloc). The value of

Table 15 Coefficient of variables in base and four review experience models (- is shown in place of coefficients that were not statistically significant at $\alpha = 0.05$)

Variable	Models				
	fit-sloc	fit-build	fit-compatibility	fit-security	fit-test-fail
$(\text{intercept})_{5.0}$	−3.4998	−3.6583	−3.9598	−3.7901	−3.4915
$(\text{intercept})_{11.0}$	−3.3000	−3.5320	−3.7830	−3.6590	−3.2610
$(\text{intercept})_{19.0}$	−4.2742	−4.5186	−4.8750	v4.6378	−4.2111
$(\text{intercept})_{27.0}$	−4.6300	−4.8930	−5.3110	−4.9990	−4.5580
$(\text{intercept})_{35.0}$	−5.8910	−6.1630	−6.6650	−6.2810	−5.8120
sloc	4.75E−05	4.75E−05	5.08E−05	4.76E−05	4.74E−05
build-experience		0.3314			
compatibility-experience			1.6100		
security-experience				0.5638	
test-fail-experience					–

Table 16 Model goodness of fit metrics for review experience models

Model	AIC	AIC Decrease	D^2	D^2 Increase
fit-sloc	14425	–	0.0649	–
fit-build	14418	−0.05 %	0.0654	0.83 %
fit-compatibility	**14160**	−1.84 %	**0.0822**	26.62 %
fit-security	14398	−0.19 %	0.0668	2.85 %
fit-test-fail	14425	0.00 %	0.0650	0.15 %

AIC and D^2 computed for each of five logistic regression models are presented in Table 16. The models that exhibited the best improvement in model goodness of fit have their AIC and D^2 values shown in boldface.

We found **fit-compatibility** to exhibit the best improvement in model goodness of fit over base model, **fit-sloc**, when compared to other models, with a 1.84 % decrease in AIC and a 26.62 % increase in D^2. The other review experience models exhibited the same or marginally better goodness of fit over the base model.

The value of AIC and D^2 computed for each of the five logistic regression models built with data from each of the five Chromium releases are shown in Table 21 in the Appendix. With the exception of release 35.0, **fit-compatibility** exhibited the best improvement in model goodness of fit over the base model when compared to the other models. In release 35.0, **fit-security** exhibited marginally better improvement over the base model than **fit-compatibility** did.

Overall, the results from the logistic regression analysis indicated that the inclusion of the review experience metrics did improve the relative quality of the regression model, albeit, marginally. The limited improvement is not sufficient to conclude that the metrics quantifying the percentage of bug review experienced developers involved in reviewing a file has a significant impact on the vulnerability likelihood of the file.

Finally, we contrasted the performance, quantified using the F-measure, of bug category group models from RQ2 (Table 8) with that of the corresponding review experience models to evaluate the set of metrics that led to a better performing model when predicting future vulnerabilities. The results from the comparison are presented in Table 17. We have used F-measure only because (a) F-measure encapsulates precision and recall and (b) AUC was commensurate with F-measure in our analysis results. We observed that the bug category group models performed better that corresponding review experience models. The result is consistent even when the models were built with data from each of the four Chromium releases as shown in Table 22 in the Appendix.

Table 17 Model performance comparison between bugs and review experience models

Model	F-measure		Variation
	Bug	Experience	
fit-build	0.1945	0.0466	76.06 %
fit-compatibility	0.1932	0.0506	73.80 %
fit-security	0.2087	0.0452	78.34 %

The bug review experience metrics had limited effect on the occurrence of future vulnerabilities, and outperformed other, more broadly available, metrics like file size (SLOC) to a very minimal extent.

6.6 RQ6: *Does the Relationship Between Bugs and Vulnerabilities Generalize Beyond Chromium?*

6.6.1 Motivation

Our goal in this question was to investigate the external validity of our conclusions by examining an additional project: Apache httpd. The data sets between Apache and Chromium differ significantly, so we translated as many of our analyses as was possible to see if the results generalize. We applied our analysis from RQ1, RQ2, RQ3, and RQ4 to the Apache data set and report those results here.

6.6.2 Analysis

The Apache httpd is, and has been, the most popular open-source web server since its inception in 1995. A considerable number of websites on the World Wide Web are served by hosts running Apache httpd. In addressing this research question, we have considered two of the most recent releases of httpd, namely 2.2.0, and 2.4.0.

The analysis approach is similar to that employed for Chromium with a change to the way number of pre-release bugs is computed: we do not restrict the time frame of the commits to one year before the release but instead consider all commits between a release and its immediate major predecessor release. For example, the number of pre-release bugs of a file in release 2.4.0 is the number of bug-fixing commits (i.e. commits with reference to a bug identifier) to the file after release 2.2.0 but before 2.4.0. Another limitation of the httpd data set is that the types of bug did not span a wide range as Chromium bugs did. We were able to identify only two types of bugs in the httpd's bug tracking system, they were regression bugs and enhancements. num-pre-enhancements and num-pre-regression-bugs represent the number of pre-release bugs of type enhancement and regression, respectively. As with Chromium, where we considered features as a type of bug, we considered enhancements as a type of bug in the context of httpd.

As in the Chromium analysis, the analysis data set was filtered to remove all files that lacked a history of both bugs and vulnerabilities. On average, approximately 56 % of files were removed from our httpd data set.

The distribution of num-pre-bugs metrics collected from vulnerable and neutral source code files in httpd is shown in Fig. 8. The Figure reveals that the num-pre-bugs of vulnerable and neutral files may have different distributions, however, the overlap between the two is considerably large and may prevent the clear delineation of vulnerable and neutral files. The MWW test corroborates what the Figure reveals, i.e. vulnerable files tend to have higher median num-pre-bugs (= 5) than neutral files (= 3). Although statistically significant, the association is weak as indicated by the overlap in distribution, the magnitude of the difference in median, and Cohen's d value of 0.50 which is regarded as medium effect.

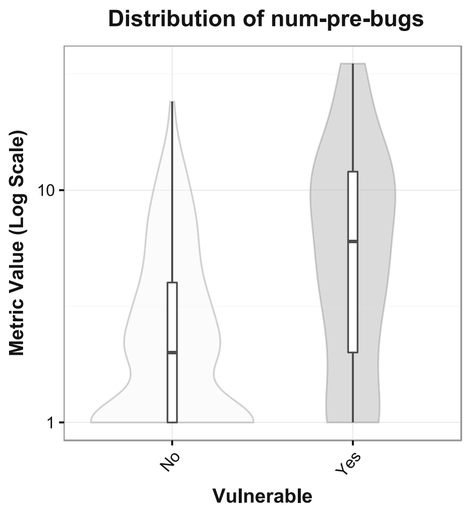


Fig. 8 Distribution of number of pre-release bugs collected from vulnerable and neutral files

To assess if categorization of bugs into bug types aids in improving the association between bugs and vulnerabilities, we built four logistic regression models—**fit-sloc**, **fit-num-pre-bugs**, **fit-enhancements**, **fit-regression**—similar to RQ2 and computed their goodness of fit measures (AIC and D^2) and performance measures (precision, recall, F-measure, and AUC). We used the entire data set (i.e. all httpd releases taken together) in building these models and, as a result, used **release** as a ordinal valued variable. The coefficients of the variable num-pre-enhancements in **fit-enhancements** and num-pre-regression-bugs in **fit-regression** were not statistically significant. As a result, we did not assess their goodness of fit or performance. **fit-num-pre-bugs** exhibited a negligible improvement in goodness of fit with a 0.06 % increase in AIC and a 4.85 % increase in D^2 but it did not outperform **fit-sloc** in performance. The insignificant results led us to conclude that the categorization of bugs into their types did not improve the association between bugs and vulnerabilities.

As with Chromium, files with most pre-release bugs per SLOC did not account for a lot of vulnerabilities. The lift curves shown in Fig. 9 reveal this fact with the curve being very close to, if not under, the diagonal that represents random guessing. Therefore, using "bugginess" (i.e. num-pre-bugs/SLOC) of a file as a metric to prioritize security audits may be a futile endeavor.

Finally, we evaluated if the files with severe vulnerabilities had a correspondingly high number of pre-release bugs. Bounties were not awarded for the vast majority of httpd bugs, so we relied upon (in our opinion questionable) CVSS score. The Spearman's ρ between num-pre-bugs and CVSS base score was -0.1231. Although statistically significant, the correlation is negligible and it is interesting to see that the correlation was negative implying that files with severe vulnerabilities had *fewer* bugs than files with less severe vulnerabilities.

The weak association between bugs and vulnerabilities observed in the Apache httpd project is consistent with that observed in Chromium project.

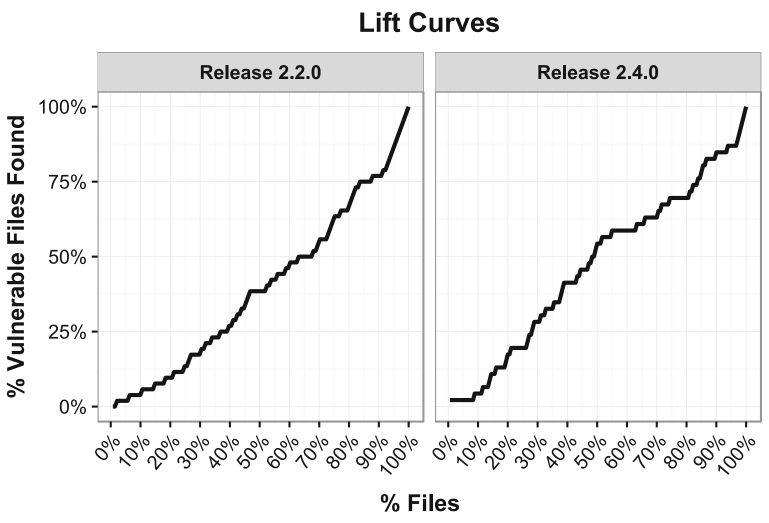


Fig. 9 Lift curve of percentage of vulnerable files found when files are ranked by "bugginess" (i.e. num-pre-bugs/SLOC) in two httpd releases

7 Discussion

The results of this study broadly demonstrates that bugs and vulnerabilities are simply not in the same places. This means that even a perfect vulnerability model would be a bad bug prediction model, and vice versa. While bugs may technically foreshadow vulnerabilities, connection is empirically and historically weak.

This results poses an interesting concern about metrics for software security. In past research (Shin et al. 2011), we have found that metrics such as churn and SLOC are correlated with vulnerabilities, as they are known to be correlated with bugs too. In the case of churn, many different problems in the team can cause problems with churn, so perhaps that metric is not specific enough to adequately target vulnerabilities. Thus, while some metrics may relate to both vulnerabilities and bugs, these metrics will be inherently limited for finding vulnerabilities in practice.

As a result, we believe that the current body of evidence suggests that vulnerability prediction requires security-specific metrics. These metrics must take into account the unique nature of security. For example, the skill of abusing functionality instead of wrong functionality comes from a different set of scenarios than a developer naturally thinks about. In everyday work, developers are asked to "build" much more often than "break". From our own experience, developers must cultivate this attacker mentality to engineer secure software, so metrics in that area show more promise than conventional metrics.

Finally, as a side note, we do not view this work's main contribution to be demonstrating the predictability of vulnerabilities. By presenting the variance in the model fitness and performance metrics, we caution the reader about drawing conclusions about the overall predictability of vulnerabilities. Other studies (even our own Shin et al. (2011)) have shown prediction models of vulnerabilities that outperform the models here. The multiple regression models in this study are purely for comparison across multiple groups of metrics.

8 Threats to Validity

We chose the Chromium project as a representative sample of a large, open source project. As with any other empirical studies, the results obtained in our study may be specific to the Chromium project. For example, Chromium project team may use the label *stability* to tag stability bugs, while another development team may choose to use *stability bug* as a tag to refer to the same type of bugs. To mitigate this, we conducted an additional study of Apache httpd in RQ6 and found that the results were consistent with Chromium.

Additionally, we cannot account for vulnerabilities that have yet to be discovered in the future. We define files without a known vulnerability as "neutral" (not "invulnerable") for this reason. In the future, historical vulnerabilities may be found that could alter these results. This validity concern is true of most bug-related research in mining software repositories.

In the logistic regression analysis of the metrics, we employed two approaches to building the models using (a) data collected from a single release and (b) data collected from all releases taken together. Both these approaches have their limitations. Models built with data from a single release may fail to capture bug patterns that take longer to reveal themselves as vulnerabilities and models built with data from all releases may be affected by duplication of data in cases where a file has not changed across releases. In the interest of completeness, we present results from both these approaches and found the results to be consistent.

We did not investigate the interplay between bug types, review experience types, and severity because, individually, these have a weak association with vulnerabilities anyway. As a sanity check, we examined models with multiple categories of metrics together, and we did not observe any noteworthy results.

9 Summary

In this study, we evaluated the relationship between pre-release bugs and post-release vulnerabilities in the Chromium project. While there existed an empirical association between bugs and vulnerabilities, the association was considerably weak. We found past security-related bugs and new features, neither of which are non-security bugs, to be the strongest indicators of vulnerabilities. We also found that the buggiest files did not intersect with files with many vulnerabilities. Factoring the severity of the vulnerabilities revealed that files with severe vulnerabilities did not have a corresponding increase in number of pre-release bugs. When an additional dimension—developers' bug review experience—was considered, we found that vulnerable files had fewer bug review experienced developers than neutral files. However, the bug review experience based metrics were only marginally better in predicting vulnerability severity when compared to traditional metrics such as SLOC. Lastly, our results generalized to a second project (Apache httpd) where data was available. In summary, the empirical evidence underscores the dissimilarity between bugs and vulnerabilities and indicates that additional empirical research must be directed at vulnerability data specifically.

Acknowledgments This research is supported by the National Science Foundation (grant CCF-1441444). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We thank the Software Archeology group at RIT for their valuable contributions to this work.

Appendix

Table 18 Model goodness of fit metrics for base, reference, and bug category group models built with data from five Chromium releases

Release	Model	AIC	AIC Decrease	D^2	D^2 Increase
Release 5.0	fit-sloc	1424.40	–	0.0384	–
	fit-num-pre-bugs	**1377.40**	−3.30 %	**0.0716**	86.39 %
	fit-build	1423.80	−0.04 %	0.0415	8.09 %
	fit-features	1423.90	−0.04 %	0.0401	4.34 %
	fit-security	1402.20	−1.56 %	0.0548	42.68 %
	fit-stability	1419.50	−0.34 %	0.0458	19.25 %
Release 11.0	fit-sloc	3098.40	–	0.0809	–
	fit-num-pre-bugs	3093.00	−0.17 %	0.0831	2.69 %
	fit-build	3099.20	0.03 %	0.0819	1.15 %
	fit-features	**3071.10**	−0.88 %	**0.0896**	10.73 %
	fit-security	3095.90	−0.08 %	0.0823	1.65 %
	fit-stability	3090.50	−0.25 %	0.0850	5.08 %
Release 19.0	fit-sloc	2365.60	–	0.0429	–
	fit-num-pre-bugs	2367.00	0.06 %	0.0432	0.60 %
	fit-build	2350.10	−0.66 %	0.0509	18.41 %
	fit-features	**2337.90**	−1.17 %	**0.0550**	28.06 %
	fit-security	2358.60	−0.30 %	0.0466	8.50 %
	fit-stability	2361.30	−0.18 %	0.0471	9.75 %
Release 27.0	fit-sloc	2432.50	–	0.0436	–
	fit-num-pre-bugs	2434.40	0.08 %	0.0436	0.02 %
	fit-build	2421.90	−0.44 %	0.0493	13.11 %
	fit-features	2434.10	0.07 %	0.0437	0.34 %
	fit-security	**2286.10**	−6.02 %	**0.1020**	134.10 %
	fit-stability	2427.50	−0.21 %	0.0479	9.88 %
Release 35.0	fit-sloc	1038.20	–	0.0330	–
	fit-num-pre-bugs	1040	0.17 %	0.0332	0.47 %
	fit-build	1038.70	0.05 %	0.0413	25.01 %
	fit-features	1015.50	−2.19 %	0.0561	69.82 %
	fit-security	**976.42**	−5.95 %	**0.0926**	180.41 %
	fit-stability	1038.70	0.05 %	0.0382	15.50 %

Table 19 Model performance metrics for base, reference, and bug category group models built with data from four Chromium releases

Release	Model	Precision	Recall	F-measure	AUC
Release 5.0	fit-sloc	0.2935	0.3516	0.3200	70.89 %
	fit-num-pre-bugs	0.2391	0.4926	0.3220	66.63 %
	fit-build	0.2451	0.4848	0.3256	70.86 %
	fit-features	0.2455	**0.4958**	**0.3284**	**71.97 %**
	fit-security	**0.2765**	0.3424	0.3059	70.41 %
	fit-stability	0.2646	0.4121	0.3223	69.92 %
Release 11.0	fit-sloc	0.1082	0.3125	0.1608	66.84 %
	fit-num-pre-bugs	0.0867	**0.4821**	0.1469	66.12 %
	fit-build	0.0947	0.4429	0.1561	67.61 %
	fit-features	0.0981	0.4560	0.1615	**68.77 %**
	fit-security	**0.1099**	0.3407	**0.1662**	67.06 %
	fit-stability	0.0902	0.4123	0.1480	66.40 %
Release 19.0	fit-sloc	0.1071	0.3215	0.1607	66.48 %
	fit-num-pre-bugs	0.0714	**0.5129**	0.1253	66.57 %
	fit-build	0.0796	0.4786	0.1365	68.42 %
	fit-features	0.0650	0.4970	0.1150	65.54 %
	fit-security	**0.1463**	0.3989	**0.2140**	**70.03 %**
	fit-stability	0.0781	0.4597	0.1335	66.35 %
Release 27.0	fit-sloc	0.0360	0.3453	0.0653	66.50 %
	fit-num-pre-bugs	0.0258	**0.5443**	0.0492	66.48 %
	fit-build	0.0341	0.4531	0.0634	67.41 %
	fit-features	0.0298	0.4667	0.0560	67.31 %
	fit-security	**0.0559**	0.5054	**0.1006**	**75.34 %**
	fit-stability	0.0297	0.4473	0.0557	65.90 %

Table 20 MWW test results for review experience metrics in five Chromium releases

Release	Metric	Mean		Median		*p*-value
		Vulnerable	Neutral	Vulnerable	Neutral	
Release 5.0	build-experience	0.480	0.464	0.467	0.500	4.11E-01
	compatibility-experience	0.325	0.234	0.342	0.200	**1.02E-14**
	security-experience	0.555	0.487	0.563	0.500	**2.69E-04**
	test-fail-experience	0.047	0.056	0.028	0.000	**8.91E-15**
Release 11.0	build-experience	0.691	0.689	0.680	0.697	7.84E-02
	compatibility-experience	0.336	0.244	0.353	0.200	**2.20E-16**
	security-experience	0.576	0.616	0.612	0.667	**1.84E-05**
	test-fail-experience	0.256	0.255	0.209	0.200	5.26E-02
Release 19.0	build-experience	0.757	0.731	0.785	0.750	1.73E-02
	compatibility-experience	0.454	0.315	0.460	0.316	**2.20E-16**
	security-experience	0.702	0.632	0.719	0.667	**1.18E-09**
	test-fail-experience	0.363	0.409	0.376	0.412	**1.68E-04**

Table 20 (continued)

Release	Metric	Mean		Median		*p*-value
		Vulnerable	Neutral	Vulnerable	Neutral	
Release 27.0	build-experience	0.829	0.790	0.847	0.818	2.01E-02
	compatibility-experience	0.500	0.369	0.500	0.384	**2.20E-16**
	security-experience	0.721	0.640	0.722	0.667	**1.62E-11**
	test-fail-experience	0.494	0.465	0.500	0.500	**1.08E-02**
Release 35.0	build-experience	0.833	0.816	0.841	0.857	6.69E-01
	compatibility-experience	0.576	0.432	0.607	0.451	**7.44E-11**
	security-experience	0.811	0.677	0.824	0.706	**1.01E-09**
	test-fail-experience	0.485	0.509	0.500	0.500	2.46E-01

Table 21 Model goodness of fit metrics for review experience models built with data from five Chromium releases

Release	Model	AIC	AIC Decrease	D^2	D^2 Increase
Release 5.0	fit-sloc	2001.4	–	0.0007	–
	fit-build	2002.7	0.06 %	0.0011	44.94 %
	fit-compatibility	**1975.3**	−1.30 %	**0.0148**	1912.19 %
	fit-security	1993	−0.42 %	0.0059	708.26 %
	fit-test-fail	2002	0.03 %	0.0014	94.81 %
Release 11.0	fit-sloc	4599.7	–	0.0049	–
	fit-build	4601.6	0.04 %	0.0049	0.12 %
	fit-compatibility	**4536.8**	−1.37 %	**0.0189**	287.13 %
	fit-security	4590.8	−0.19 %	0.0072	48.11 %
	fit-test-fail	4601.6	0.04 %	0.0049	0.03 %
Release 19.0	fit-sloc	3171.3	–	0.0016	–
	fit-build	3167.3	−0.13 %	0.0035	117.10 %
	fit-compatibility	**3092.8**	−2.48 %	**0.0270**	1556.93 %
	fit-security	3134.9	−1.15 %	0.0137	742.60 %
	fit-test-fail	3156.7	−0.46 %	0.0069	322.48 %
Release 27.0	fit-sloc	3277.2	–	0.0034	–
	fit-build	3263.6	−0.41 %	0.0081	139.61 %
	fit-compatibility	**3204.4**	−2.22 %	**0.0262**	669.82 %
	fit-security	3234.2	−1.31 %	0.0171	402.63 %
	fit-test-fail	3274.3	−0.09 %	0.0049	44.07 %
Release 35.0	fit-sloc	1363.4	–	0.0010	–
	fit-build	1364.6	0.09 %	0.0016	61.81 %
	fit-compatibility	1330.9	−2.38 %	0.0264	2524.28 %
	fit-security	**1326.4**	−2.71 %	**0.0297**	2856.74 %
	fit-test-fail	1364.4	0.07 %	0.0018	74.66 %

Table 22 Model performance comparison between bugs and review experience models built with data from four Chromium releases

Release	Model	F-measure		Variation
		Bug	Experience	
Release 5.0	fit-sloc	0.3200	0.1349	57.85 %
	fit-build	0.3256	0.0718	77.94 %
	fit-compatibility	0.3059	0.0849	72.26 %
	fit-security	0.3223	0.0582	81.95 %
Release 11.0	fit-sloc	0.1608	0.0692	56.96 %
	fit-build	0.1561	0.0550	64.74 %
	fit-compatibility	0.1662	0.0458	72.45 %
	fit-security	0.1480	0.0296	80.00 %
Release 19.0	fit-sloc	0.1607	0.0820	48.98 %
	fit-build	0.1365	0.0254	81.38 %
	fit-compatibility	0.2140	0.0272	87.30 %
	fit-security	0.1335	0.0254	80.98 %
Release 27.0	fit-sloc	0.0653	0.0346	46.96 %
	fit-build	0.0634	0.0061	90.38 %
	fit-compatibility	0.1006	0.0090	91.09 %
	fit-security	0.0557	0.0085	84.75 %

References

Algarni A, Malaiya Y (2014) Software vulnerability markets: Discoverers and buyers. Inter J Comp Inf Scie Engin 8(3):71–81

Allodi L, Massacci F (2012a) A Preliminary Analysis of Vulnerability Scores for Attacks in Wild. In: Proceedings of the 2012 ACM Workshop on Building analysis datasets and gathering experience returns for security - BADGERS '12, p 17. doi:10.1145/2382416.2382427

Allodi L, Massacci F (2012b) A Preliminary Analysis of Vulnerability Scores for Attacks in Wild: The EKITS and SYM Datasets. In: Proceedings of the 2012 ACM Workshop on Building analysis datasets and gathering experience returns for security, ACM, pp 17–24

Allodi L, Massacci F (2014) Comparing vulnerability severity and exploits using case-control studies. ACM Trans Inf Syst Secur 17(1):1

Allodi L, Shim W, Massacci F (2013) Quantitative assessment of risk reduction with cybercrime black market monitoring. In: Security and Privacy Workshops (SPW), 2013 IEEE, IEEE, pp 165–172

Bird C, Menzies T, Zimmermann T (2015) The Art and Science of Analyzing Software Data: Analysis Patterns. Elsevier Science

Bozorgi M, Saul LK, Savage S, Voelker GM (2010) Beyond Heuristics: Learning to Classify Vulnerabilities and Predict Exploits. In: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, pp 105–114. doi:10.1145/1835804.1835821

Burnham KP, Anderson DR (2004) Multimodel inference understanding aic and bic in model selection. Sociol Methods Res 33(2):261–304

Chen TY, Kuo FC, Merkel R (2004) On the statistical properties of the f-measure. In: Quality Software, 2004. QSIC 2004. Proceedings. Fourth International Conference on, pp 146–153. doi:10.1109/QSIC.2004.1357955

Chen TH, Thomas S, Nagappan M, Hassan A (2012) Explaining software defects using topic models. In: Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on, pp 189–198. doi:10.1109/MSR.2012.6224280

Cohen J (1992) Statistical power analysis. Curr Dir Psychol Sci:98–101

Cohen J (2013) Statistical power analysis for the behavioral sciences. Academic press

Cruz A, Ochimizu K (2009) Towards logistic regression models for predicting fault-prone code across software projects, pp Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on, pp 460–463. doi:10.1109/ESEM.2009.5316002

Finifter M, Akhawe D, Wagner D (2013) An Empirical Study of Vulnerability Rewards Programs. In: USENIX Security, vol 13

Gegick M, Williams L, Osborne J, Vouk M (2008) Prioritizing software security fortification throughcode-level metrics. In: Proceedings of the 4th ACM workshop on Quality of protection, ACM, pp 31–38

Gegick M, Rotella P, Williams L (2009) Predicting attack-prone components. In: Software Testing Verification and Validation, 2009. ICST '09. International Conference on, pp 181–190. doi:10.1109/ICST.2009.36

Guisan A, Zimmermann NE (2000) Predictive habitat distribution models in ecology. Ecol Model 135(2):147–186

Krishnamurthy S, Tripathi AK (2006) Bounty programs in free/libre/open source software. BITZER Jurgen, The Economics of Open Source Software Development, Lavoisier, Paris

Krsul IV (1998) Software vulnerability analysis, PhD thesis, Purdue University

Meneely A, Srinivasan H, Musa A, Rodriguez Tejeda A, Mokary M, Spates B (2013) When a patch goes bad: Exploring the properties of vulnerability-contributing commits. In: Empirical Software Engineering and Measurement, 2013 ACM / IEEE International Symposium on, pp 65–74. doi:10.1109/ESEM.2013.19

Meneely A, Tejeda ACR, Spates B, Trudeau S, Neuberger D, Whitlock K, Ketant C, Davis K (2014) An empirical investigation of socio-technical code review metrics and security vulnerabilities. In: Proceedings of the 6th International Workshop on Social Software Engineering, ACM, New York, NY, USA, SSE 2014, pp 37–44. doi:10.1145/2661685.2661687

Miller C (2007) The legitimate vulnerability market: Inside the secretive world of 0-day exploit sales. In: In Sixth Workshop on the Economics of Information Security, Citeseer

Mitropoulos D, Gousios G, Spinellis D (2012) Measuring the occurrence of security-related bugs through software evolution. In: Informatics (PCI), 2012 16th Panhellenic Conference on, pp 117–122. doi:10.1109/PCi.2012.15

Mitropoulos D, Karakoidas V, Louridas P, Gousios G, Spinellis D (2013) Dismal code: Studying the evolution of security bugs. In: Proceedings of the LASER 2013 (LASER 2013), USENIX, Arlington, VA, pp 37–48. https://www.usenix.org/laser2013/program/mitropoulos

Mukaka M (2012) A guide to appropriate use of correlation coefficient in medical research. Malawi Med J 24(3):69–71

Neuhaus S, Zimmermann T, Holler C, Zeller A (2007) Predicting vulnerable software components. In: Proceedings of the 14th ACM conference on Computer and communications security, ACM, pp 529–540

Poncin W, Serebrenik A, van den Brand M (2011) Process mining software repositories. In: Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on, pp 5–14. doi:10.1109/CSMR.2011.5

R Core Team (2015) R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. http://www.R-project.org/

Radianti J, Gonzalez JJ (2007) Understanding hidden information security threats: The vulnerability black market. In: System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on, IEEE, pp 156c–156c

Raftery AE (1995) Bayesian model selection in social research. Sociol Methodol 25:111–164

Ruscio J (2008) A probability-based measure of effect size: Robustness to base rates and other factors. Psychol Methods 13(1):19

Schneidewind NF (1992) Methodology for validating software metrics. Software Engineering. Trans IEEE 18(5):410–422

Shihab E, Mockus A, Kamei Y, Adams B, Hassan AE (2011) High-impact defects: a study of breakage and surprise defects. In: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, ACM, pp 300–310

Shin Y, Meneely A, Williams L, Osborne J (2011) Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. IEEE Trans Softw Eng 37(6):772–787. doi:10.1109/TSE.2010.81

Tantithamthavorn C, McIntosh S, Hassan AE, Ihara A, ichi Matsumoto K (2015) The impact of mislabelling on the performance and interpretation of defect prediction models. In: Proc. of the 37th Int'l Conf. on Software Engineering (ICSE), p To appear

Tegarden D, Sheetz S, Monarchi D (1992) Effectiveness of traditional software metrics for object-oriented systems. In: System Sciences, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference on, vol iv, pp 359–368 vol.4. doi:10.1109/HICSS.1992.183365

Younis AA, Malaiya YK (2015) Comparing and Evaluating CVSS Base Metrics and Microsoft Rating System. In: 2015 IEEE International Conference on Software Quality, Reliability and Security (QRS), IEEE, pp 252–261

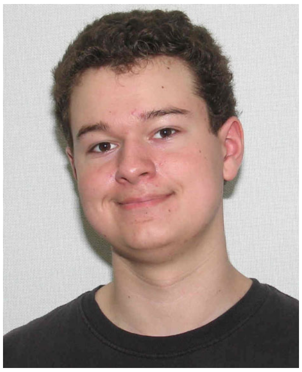
Younis A, Malaiya YK, Ray I (2015) Assessing vulnerability exploitability risk using software properties. Softw Qual J. doi:10.1007/s11219-015-9274-6

Zakzanis KK (2001) Statistics to tell the truth, the whole truth, and nothing but the truth: formulae, illustrative numerical examples, and heuristic interpretation of effect size analyses for neuropsychological researchers. Arch Clin Neuropsychol 16(7):653–667

Zimmermann T, Nagappan N, Williams L (2010) Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista. In: 2010 Third International Conference on Software Testing, Verification and Validation (ICST), IEEE, pp 421–428

Nuthan Munaiah is a Ph.D. student in the B. Thomas Golisano College of Computing and Information Sciences, Rochester Institute of Technology. He has a Bachelors of Engineering degree in Computer Science and Engineering from Visvesvaraya Technological University, India. Prior to starting Graduate school, Nuthan spent five years working as a Software Engineer developing web applications for a popular bank in the New England region of the United States of America. Nuthan's research is focused on analyzing historical vulnerability fixes to identify patterns that may be applied to understanding the engineering failures that led to the vulnerability.

Felivel Camilo was awarded a scholarship to study a Master in Software Engineering at RIT in 2013. His Masters Thesis "Do Bugs Foreshadow Vulnerabilities? A Study of the Chromium Project" focused on studying the relation between bugs and vulnerabilities. His work was presented in the MSR 2015 conference, wining various awards. Felivel now works in the industry as a Cloud Solutions Architect, focusing on AWS and Azure.



Wesley Wigham is currently pursuing a baccalaureate degree in Software Engineering at Rochester Institute of Technology.

Andrew Meneely Andy has been an assistant professor of Software Engineering at RIT since 2011. Prior to then, Andy received his PhD in Computer Science at North Carolina State University in Raleigh, North Carolina under Laurie Williams. He also earned his Masters at NCSU in 2008. Andy received his Bachelors of Arts at Calvin College, Grand Rapids, MI where he was a double-major in Computer Science and Mathematics.

Meiyappan Nagappan is an Assistant Professor in the Software Engineering department of Rochester Institute of Technology. Previously he was a postdoctoral fellow in the Software Analysis and Intelligence Lab (SAIL) at Queen's University, Canada. His research is centered around the use of large-scale Software Engineering (SE) data to address the concerns of the various stakeholders (e.g., developers, operators, and managers). He received a PhD in computer science from North Carolina State University. Dr. Nagappan has published in various top SE venues such as TSE, FSE, EMSE, and IEEE Software. He has also received best paper awards at the International Working Conference on Mining Software Repositories (MSR '12, '15). He is currently the editor of the IEEE Software Blog. He continues to collaborate with both industrial and academic researchers from the US, Canada, Japan, Germany, Chile, and India. You can find more at http://www.mei-nagappan.com.