On the Assessment of Security and Performance Bugs in Chromium Open-Source Project

Joseph Imseis[1], Costain Nachuma[1], Shaikh Arifuzzaman[1,2](✉), Minhaz Zibran[1], and Zakirul Alam Bhuiyan[3]

[1] Computer Science Department, University of New Orleans, New Orleans, LA 70148, USA
{jimseis, cnachuma, smarifuz, mzibran}@uno.edu
[2] Big Data and Scalable Computing Research (BDSC), UNO, New Orleans, LA 70148, USA
[3] Computer Science Department, Fordham University, Bronx, NY 10458, USA
mbhuiyan3@fordham.edu

Abstract. An individual working in software development should have a fundamental understanding of how different types of bugs impact various project aspects. This knowledge allows one to improve the quality of the created software. The problem, however, is that previous research typically treats all types of bugs as similar when analyzing several aspects of software quality (e.g. predicting the time to fix a bug) or concentrates on a particular bug type (e.g. performance bugs) with little comparison to other types. In this paper, we look at how different types of bugs, specifically performance and security bugs, differ from one another. Our study is based on a previous study done by Zaman et al. [1] in which the study was performed on the FireFox project. Instead of Firefox, we will be working with the Chrome web-browser. In our case study, we find that security bugs are fixed faster than performance bugs and that the developers who were assigned to security bugs typically have more experience than the ones assigned to performance bugs. Our findings emphasize the importance of considering the different types of bugs in software quality research and practice.

Keywords: Assessment of security · Performance bugs · Data mining · Software repository · Chromium project · Open-source project

1 Introduction

In past studies, researchers have noted that the majority of software development costs came from maintenance and evolutionary activities [2, 3]. Researchers have focused on detecting various code smells, and types of bugs in order to improve the quality assurance of software products. For example, CCFinder, a token-based code clone detection system, was used by researchers to determine how code clones affected the

The original version of this chapter was revised: The authorship information has been corrected by adding the name of Minhaz Zibran to the paper contributors. The correction to this chapter is available at https://doi.org/10.1007/978-981-15-1304-6_40 chances of faults occurring in a particular program [4]. Other prediction models exist that tend to focus on the time it takes to fix a bug [5–7] and which type of developer should fix that bug [8].

The problem is that most researchers working in quality assurance tend to treat bugs as generic, leaving out the differing aspects of each bug (i.e., there is no distinction between different bug types). This is concerning, because we could have a security bug that gives an intruder administrator access being grouped alongside with a simple syntax bug. Taking this example into consideration, we believe that it is vital for software quality researchers and developers to take into consideration the various aspects of different bug types. Based on a previous study by Zaman et al. [1], we study the characteristics and differences of security and performance bugs in the Chromium open-source project to demonstrate that these characteristics differ from one bug type to another. We aim to answer the following three research questions:

(Q1) **How fast are the bugs fixed?**
On average, security bugs are fixed 44% faster than performance bugs.

(Q2) **Which types of bugs are assigned to more experienced developers?**
The developers who work on security bugs are considered more experienced on average by ~8% when compared to developers working on performance bugs.

(Q3) **What are the characteristics of the bug fixes?**
On average, security bugs have significantly more lines of code added/deleted.

Section Summary: Section 2 includes our design and approach. Section 3 deals with answering the three research questions proposed. Section 4 mentions the limitations. Section 5 discusses the related works. Section 6 presents the conclusion to our paper.

2 Design and Approach

This section includes the design of our study in which we compare security and performance bugs of the Chrome web-browser. Figure 1 gives an overview of our approach. First, we scrape the data from the Chromium bug issue tracker website. The bugs obtained are then categorized based on security and performance. For all the bugs we then calculate several metrics and use them to make comparisons across the differing bug types. This section explains each of these steps in a generalized fashion.

2.1 Classification of Bugs

The chromium issue tracker has a built-in tagging feature that we were able to use. Specifically, we wanted to look at all issues which contained fixed bugs. We did this by selecting “All issues” from the dropdown menu and typing the keywords: “status: Fixed” in the omnibox provided.

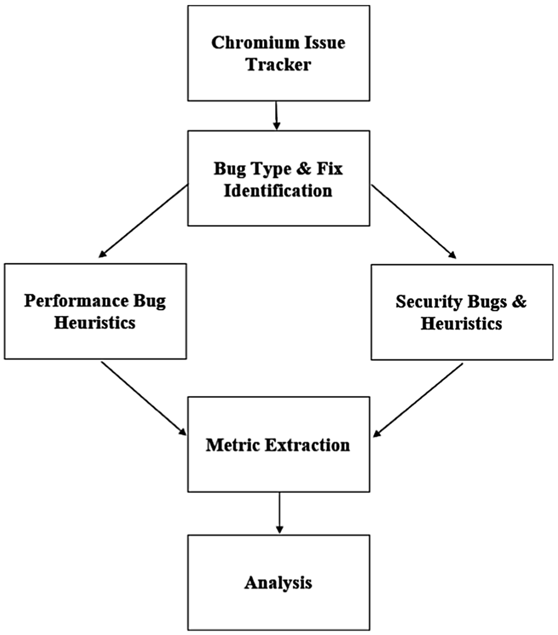

Chromium Issue Tracker
Bug Type & Fix Identification
Performance Bug Heuristics
Security Bugs & Heuristics
Metric Extraction
Analysis

Fig. 1. Overview of approach to analyze the differences in characteristics of security and performance bugs.

Luckily the issue tracker also has a “Type” parameter that allowed us to specifically search for security bugs (i.e., our input for the omnibox was “Type = Bug-Security”). However, there was no category via the Type parameter that pertained to performance. Hence, in order to obtain bugs that are related to performance we used the “summary:” parameter which checks the description of the bug for the search term provided (e.g. the search “summary: performance” would return bugs with the word performance in their summary/description section). In addition to the word “performance”, we also used heuristic keywords to obtain more bugs related to performance. The heuristics agreed upon are as follows: “duration”, “execution time”, “fast”, “faster”, “memory”, “run-time”, “slow”, and “slower”. Using these heuristics, we found 4605 performance bugs. A similar method was followed for security bugs using the following terms: “exploit”, “safe”, “safety”, “secure”, “steal”, “vulnerable”, “vulnerability”, and “vulnerabilities. A total of 5575 security bugs were found of which 4899 bugs came from the “Type: Bug-Security” search and 676 bugs came from our heuristic terms.

Unfortunately, without the use of the Monorail API, downloading these bug reports becomes tedious, as the bug issue tracker only allows a user to download 1000 records at a time. To remedy this, we wrote a simple python script, **RecordScrapper**, which automatically sends download requests for the next 1000 records. RecordScrapper then iteratively sends these requests until all the records pertaining to a particular category are downloaded. Note that all bugs obtained via the method mentioned above were downloaded as CSV files between March 2019 and April 2019.

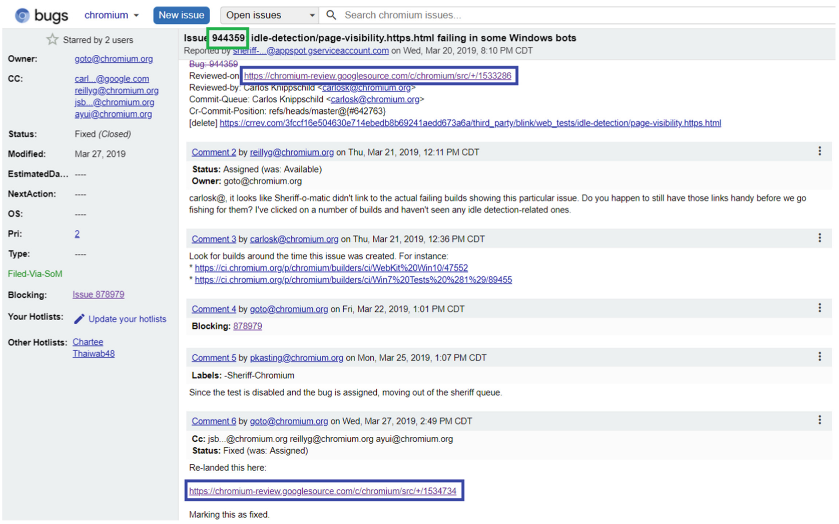
2.2 Tracking Bugs

To answer research question 1, we decided to determine the lifecycles for every category of bug. To do this we wrote a python script, **DateScrapper**, which uses the bug data, specifically the BugID, obtained from RecordScrapper as input. DateScrapper then iteratively takes a bugID and appends it to the end of this chromium issue tracker link: https://bugs.chromium.org/p/chromium/issues/detail_ezt?id=.

By doing so, the script now has access to the information associated with that particular bugID. From here, the script is able to obtain the start/assigned date and end/fixed date for each individual bugID. It does so by employing a python package which contains an HTML parser, known as Beautiful Soup. Beautiful Soup allows us to create a soup object which contains the HTML of an individual bug report. By using this soup object we can specifically target key areas of the website by their tags, classes, or several other web elements. This is how we are able to acquire the start/assigned date and end/fixed date. In the event that no start date is found, the date that the bug was assigned to a developer is used. If both the start date and the date assigned are not found, then the date that the bug was reported is considered as the start date. If the end date/date fixed is not found, then the date that the bug is closed is considered as the end date. We have had no instances where the closed date was not found, as this is a mandatory requirement for each bug enacted by the Chromium team.

The times obtained by DateScrapper are represented in UNIX time and need to be converted into a more readable format. In order to do so, we made another python script, **UnixConverter**.

UnixConverter generates the length of time taken to fix a bug in the typical month, day, year, hour format. In order to do so, it first converts the start and end dates acquired from DateScrapper into the previously mentioned standard form. It then takes the difference between the start and end dates which gives us the total length of time taken to fix a bug (i.e. the bugs life cycle). Note that for our purposes the output file of UnixConverter only contains the bugID, and the number of days and hours taken to fix a particular bug. We will generate the mean of these values and use them as part of our analysis in Sect. 3.



Starred by 2 users

Owner: goto@chromium.org

CC: carl...@google.com
reillyg@chromium.org
jsb...@chromium.org
ayui@chromium.org

Status: Fixed *(Closed)*

Modified: Mar 27, 2019

EstimatedDa... ----

NextAction: ----

OS: ----

Pri: 2

Type: ----

Filed-Via-SoM

Blocking: Issue 878979

Your Hotlists:

Other Hotlists: Chartee
Thaiwab48

Issue 944359: idle-detection/page-visibility.https.html failing in some Windows bots
Reported by sheriff-...@appspot.gserviceaccount.com on Wed, Mar 20, 2019, 8:10 PM CDT

~~Bug: 944359~~
Reviewed-on: https://chromium-review.googlesource.com/c/chromium/src/+/1533286
Reviewed-by: Carlos Knippschild <carlosk@chromium.org>
Commit-Queue: Carlos Knippschild <carlosk@chromium.org>
Cr-Commit-Position: refs/heads/master@{#642763}
[delete] https://crrev.com/3fccf16e504630e714ebedb8b69241aedd673a6a/third_party/blink/web_tests/idle-detection/page-visibility.https.html

Comment 2 by reillyg@chromium.org on Thu, Mar 21, 2019, 12:11 PM CDT

Status: Assigned (was: Available)
Owner: goto@chromium.org

carlosk@, it looks like Sheriff-o-matic didn't link to the actual failing builds showing this particular issue. Do you happen to still have those links handy before we go fishing for them? I've clicked on a number of builds and haven't seen any idle detection-related ones.

Comment 3 by carlosk@chromium.org on Thu, Mar 21, 2019, 12:36 PM CDT

Look for builds around the time this issue was created. For instance:
* https://ci.chromium.org/p/chromium/builders/ci/WebKit%20Win10/47552
* https://ci.chromium.org/p/chromium/builders/ci/Win7%20Tests%20%281%29/89455

Comment 4 by goto@chromium.org on Fri, Mar 22, 2019, 1:01 PM CDT

Blocking: 878979

Comment 5 by pkasting@chromium.org on Mon, Mar 25, 2019, 1:07 PM CDT

Labels: -Sheriff-Chromium

Since the test is disabled and the bug is assigned, moving out of the sheriff queue.

Comment 6 by goto@chromium.org on Wed, Mar 27, 2019, 2:49 PM CDT

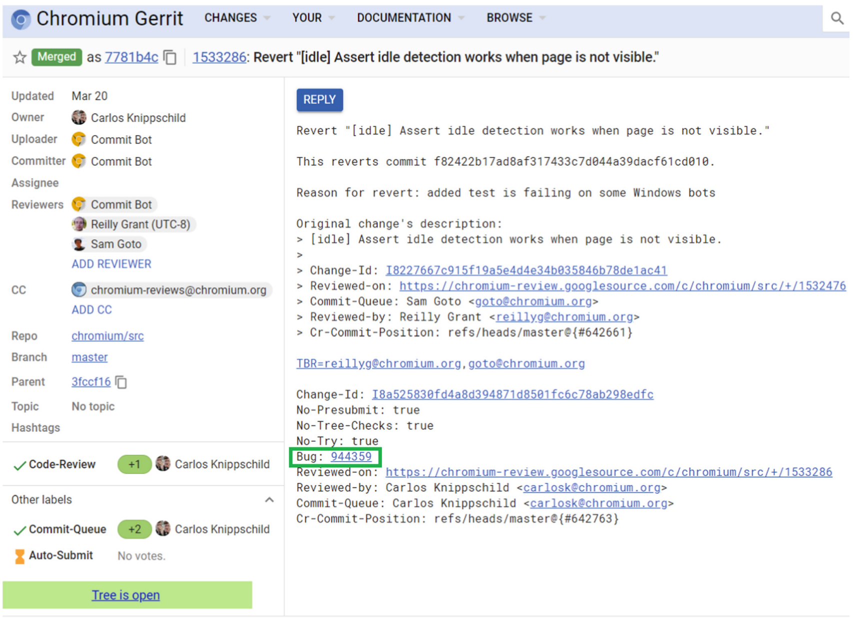
Cc: jsb...@chromium.org reillyg@chromium.org ayui@chromium.org
Status: Fixed (was: Assigned)

Re-landed this here:

https://chromium-review.googlesource.com/c/chromium/src/+/1534734

Marking this as fixed.

Fig. 2. This figure illustrates that one particular bug, represented by its bugID (green box), may have multiple change log links (blue boxes) associated with it. (Color figure online)



Merged as 7781b4c 1533286: Revert "[idle] Assert idle detection works when page is not visible."

Updated Mar 20
Owner Carlos Knippschild
Uploader Commit Bot
Committer Commit Bot
Assignee
Reviewers Commit Bot
Reilly Grant (UTC-8)
Sam Goto
CC chromium-reviews@chromium.org
Repo chromium/src
Branch master
Parent 3fccf16
Topic No topic
Hashtags

Code-Review +1 Carlos Knippschild

Other labels

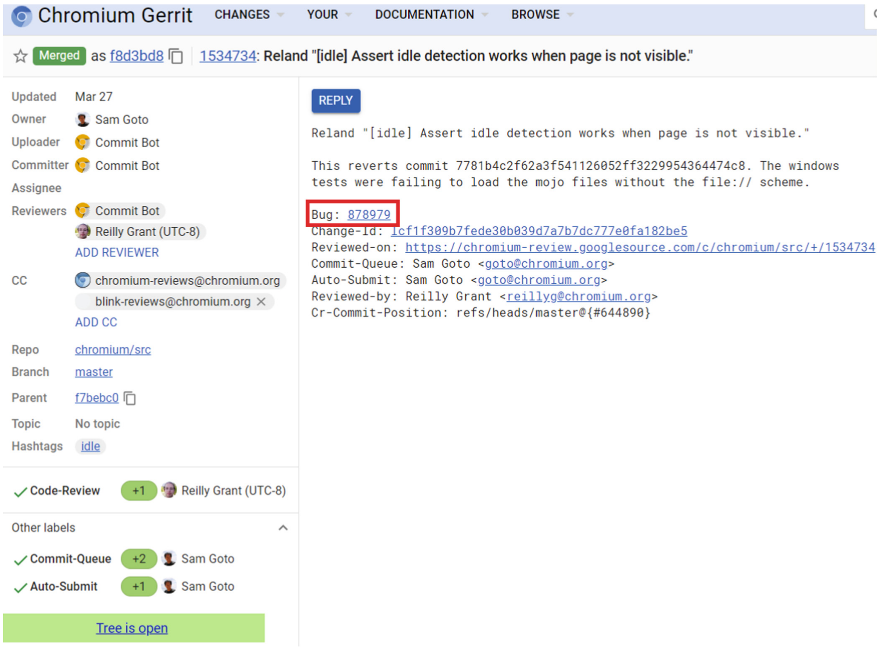
Commit-Queue +2 Carlos Knippschild
Auto-Submit No votes.

```
Revert "[idle] Assert idle detection works when page is not visible."

This reverts commit f82422b17ad8af317433c7d044a39dacf61cd010.

Reason for revert: added test is failing on some Windows bots

Original change's description:
> [idle] Assert idle detection works when page is not visible.
>
> Change-Id: I8227667c915f19a5e4d4e34b035846b78de1ac41
> Reviewed-on: https://chromium-review.googlesource.com/c/chromium/src/+/1532476
> Commit-Queue: Sam Goto <goto@chromium.org>
> Reviewed-by: Reilly Grant <reillyg@chromium.org>
> Cr-Commit-Position: refs/heads/master@{#642661}

TBR=reillyg@chromium.org,goto@chromium.org

Change-Id: I8a525830fd4a8d394871d8501fc6c78ab298edfc
No-Presubmit: true
No-Tree-Checks: true
No-Try: true
Bug: 944359
Reviewed-on: https://chromium-review.googlesource.com/c/chromium/src/+/1533286
Reviewed-by: Carlos Knippschild <carlosk@chromium.org>
Commit-Queue: Carlos Knippschild <carlosk@chromium.org>
Cr-Commit-Position: refs/heads/master@{#642763}
```

Fig. 3. This figure is the result of clicking on the first hyperlink (blue box) in Fig. 2. Notice that the bug ID (boxed in green) matches the bug ID shown in Fig. 2. (Color figure online)

2.3 Obtaining Final Metrics

In order to answer research questions 2 and 3, we need to determine the lines of code a single developer has altered (i.e. added or removed), the number of days taken to fix a bug, and the number of bugs they've fixed. We created a 4th python script, **MergeEmail,** which takes the bugIDs from the output of RecordScrapper (i.e. BugID, developer email, etc.…) and merges it with the output file of DateScrapper (BugID, Start and End time in UNIX format). This produces a file which contains the BugID, developer email, start time and end time [UNIX format]. So the whole purpose of MergeEmail is to just obtain the relative developer email, which is how we identify which developer worked on a particular bugID. It must be noted that if no email was provided for a bug, then the record is not considered for merging and analysis.



☆ Merged as f8d3bd8 1534734: Reland "[idle] Assert idle detection works when page is not visible."

Updated	Mar 27
Owner	Sam Goto
Uploader	Commit Bot
Committer	Commit Bot
Assignee	
Reviewers	Commit Bot, Reilly Grant (UTC-8)
CC	chromium-reviews@chromium.org, blink-reviews@chromium.org
Repo	chromium/src
Branch	master
Parent	f7bebc0
Topic	No topic
Hashtags	idle

✓ Code-Review +1 Reilly Grant (UTC-8)

Other labels

✓ Commit-Queue +2 Sam Goto

✓ Auto-Submit +1 Sam Goto

```
Reland "[idle] Assert idle detection works when page is not visible."

This reverts commit 7781b4c2f62a3f541126052ff3229954364474c8. The windows
tests were failing to load the mojo files without the file:// scheme.

Bug: 878979
Change-Id: Icf1f309b7fede30b039d7a7b7dc777e0fa182be5
Reviewed-on: https://chromium-review.googlesource.com/c/chromium/src/+/1534734
Commit-Queue: Sam Goto <goto@chromium.org>
Auto-Submit: Sam Goto <goto@chromium.org>
Reviewed-by: Reilly Grant <reillyg@chromium.org>
Cr-Commit-Position: refs/heads/master@{#644890}
```

Fig. 4. The figure is the result of clicking on the second hyperlink (second blue box from the top) in Fig. 2. Notice that the bug ID (boxed in red) DOES NOT match the bug ID shown in Fig. 2. Hence, this change log and it's added/deleted lines of code should not be considered when referring to bug 944359. (Color figure online)

The output of MergeEmail then becomes the input for a python script named **DaysBugs**. DaysBugs finds all the bugs for a single developer and then selects the lowest start date for a bug and the highest end date (fixed date) for that developer. This gives us the period between the first bug fixed by the developer and the last bug fixed by the developer in UNIX time which is then converted into days. DaysBugs will also keep track of the number of bugs fixed by each developer during this time period.

So, the output of DaysBugs will be as follows: Developer email/ID, total number of days the developer worked on the bugs, and number of bugs fixed.

To obtain the lines of code for each individual developer we use yet another python script, **LocBug**. This script takes the output of RecordScrapper (BugID, developer email, etc.) and iteratively appends the BugID to the chromium issue tracker link: https://bugs.chromium.org/p/chromium/issues/detail_ezt?id=similar to DateScrapper. Once on the bug report page, LocBug again creates a Soup object in order to target key areas of the website. In this case we are specifically targeting the 'comment-list' web-element as it contains the particular hyperlinks we need to retrieve the individual lines of code for each bug. These hyperlinks are structured as follows:

https://chromium-review.googlesource.com/c/chromium/src/+/
https://codereview.chromium.org/
https://chromium-review.googlesource.com/

We noted that the hyperlinks above are incomplete, in that each bug will contain a different variant of the link (i.e., a different set of numbers at the end). In other words https://chromium-review.googlesource.com/c/1475945 and https://chromium-review.googlesource.com/c/1471357 have the same format but different ending parameters (1475945 vs. 1471357). Hence, the two differing links point to two different bug change-logs. Since Beautiful Soup does not have a specific way to distinguish the differing links, we take the hyperlinks stated above, find the parameters (numbers after the links on the webpage) and then append to the end of the links.

Once this is done, LocBug will open the link it generated in order to view the change log. This change log will contain the lines of code added and deleted in a particular web-element referred to as "mainContent". To obtain access to this web-element, a tool similar to Beautiful Soup is used, called Selenium. Selenium is an automated web-browser python package that allows us to interact and manipulate web-objects/elements. In addition to scraping the lines of code added and removed, LocBug uses Selenium to ensure that the change-log link/hyperlink generated refers to the same bug on the chromium issue tracker bug report page. The reason this is needed is because there may be multiple references to other bugs within a particular bug report page as shown in Figs. 2, 3, and 4. After iteratively confirming the that change-log refers to the particular bugID, LocBug will then summate the lines of code together and save them to a file with the associated bugID (i.e., if a bug has multiple hyperlinks associated with it then LocBug will parse them all and summate the lines of code added and deleted). So, the output file generated by LocBug will contain all BugIDs and the associated lines of code added and deleted.

To obtain just the email and lines of code added/deleted, we use a python script called **DevLoc** which takes, as input, the results of RecordScrapper (BugIDs, developer email, etc.) and the output of LocBug (BugID and lines of code added/removed). By merging the input and pruning all non-needed features, we are left with the developer emails and the lines of code added and deleted for every single bug.

At this point, we need to take into account that a developer can work on more than one bug and hence their developer email will be seen multiple times in the output file of DevLoc. Hence, we want just one developer email that contains ALL the lines of code added/deleted for all the bugs that a particular developer worked on. This is done using

Apache Hadoop, which is a framework that allows for the distributed processing of large data sets [8]. We specifically use an altered Map Reduce function of Hadoop to obtain our results. A visual representation of the process can be seen in Fig. 5 below.

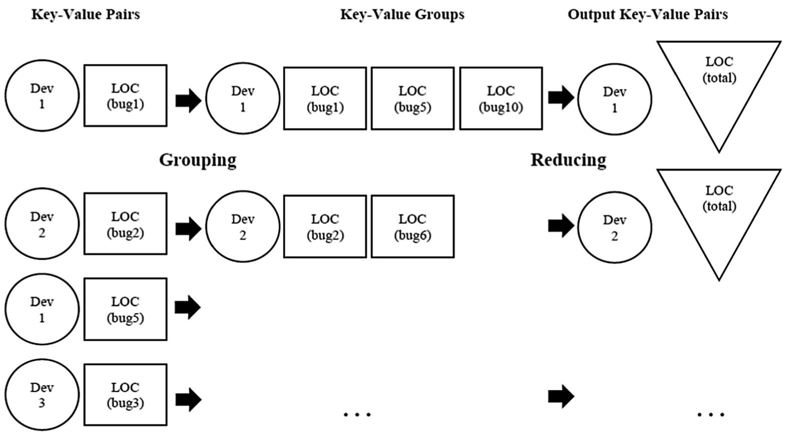


Fig. 5. We use Hadoop MapReduce framework. The lines of code (LOC) for every single bug worked on by the developer is summed and then assigned to that developer. Here, a circle denotes a single developer (key), a rectangle denotes lines of code (LOC) for a single bug (value), and the inverted triangle means total lines of code (LOC) for all bugs a developer worked on (output).

3 Answers to Our Research Questions

The subsections below pertain to one of the three research questions we studied using the data obtained from the Chromium Issue Tracker. Each question contains our motivation behind the question, our approach to answering that question and a discussion of our findings. Table 1 summarizes our results of each of the three research questions asked.

3.1 How Fast Are Bugs Fixed?

Motivation: Lead developers and project managers should be aware that security and performance bugs need to be handled differently, in that, one type of bug may hold higher precedence than another. For example, a project manager may want to quickly assign developers to a set of security bugs as those bugs may pose an immediate risk to the users/company. In addition, it would be ideal for the project manager to assign experienced developers to those higher risk bugs as to reduce the need for future maintenance. Our study measures which type of bug, performance or security, is fixed faster in terms of the Chrome web-browser.

Approach: The metric used to answer this question is the bug's life cycle. We specifically obtained the difference between the bug's start/assigned date and end/fixed date to determine a bug's life cycle.

Since the number of security bugs obtained was significantly more than the number of performance bugs, we needed to consider balancing the data in order to make accurate comparisons. To do this balancing, we first evaluated the mean for all security bugs. Next, we evaluated the mean for a randomized subset of security bugs. This randomized subset is the same size as the performance bugs. When comparing the mean of the total number of security bugs vs. the mean of the random set, we observed that the means were significantly different. This implies that the data is unbalanced and hence for our evaluation we used an equal number of security and performance bugs for our evaluations.

Findings: From Fig. 6 we can see that performance bugs take more time to fix. From Table 1, we can see that performance bugs take ~44% [i.e.|(111307.48/160691.11) − 111307.48 * 100|] longer to fix when compared to security bugs. Both findings suggest that security bugs are prioritized and hence are considered more important by the Chromium team to fix than performance bugs.

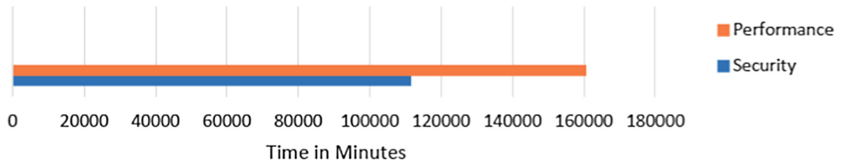


Fig. 6. Average time taken between bug assignment and fix: the chart demonstrates the average amount of minutes taken to fix a bug in its particular category. Security bugs are prioritized and fixed faster.

Table 1. Mean values for the three research questions we attempted.

Metric of interest	Security bugs	Performance bugs
Time between assignment and fix (minutes)	111307.48	160691.11
Person experience in days	598.28	646.33
Person experience in prior bugs fixed	5.69	3.56
Number of lines changed	1813.25	1559.4

3.2 Which Bugs Are Fixed by More Experienced Developers?

Motivation: Determining which developer has the most expertise to fix a bug is vital. In a previous study by Ackerman and Halverson [10] it was found that the experience of a developer was the primary criteria engineers used to determine expertise. For instance, performance bugs are complex in that they require comprehensive knowledge of the organization of the software system. Likewise, fixing a security bug requires sufficient understanding of the locations that security vulnerabilities could lie in the source code. In our study we are interested in finding the difference in experience between developers who work with security bugs vs. developers who work with performance bugs.

Approach: We measure the experience of the developer fixing a particular bug using the following two metrics:

– Experience in days, i.e. the number of days from the first bug fix of the developer to the current bug's fix date.
– Number of previously fixed bugs by the developer.

Findings: According to Table 1 above, the number of prior bugs fixed for developers that work on security bugs is greater by ~37% [i.e. |(5.68/3.55) - 5.68 * 100|] when compared to the developers that work on performance bugs. This suggests that developers who work on security bugs are more experienced. When we measure the experience based on the <u>number of days</u> from first bug fix to the current bug fix, we observed that the developers who worked on performance bugs have more experience (~8% more [i.e. |(598.28/646.33) − 598.28 * 100|]). The averages from Table 1 are also represented as bar graphs in Figs. 7 and 8 below.

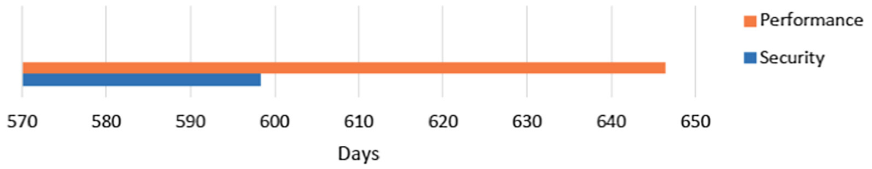


Fig. 7. Mean developer experience in days: the chart represents the average amount of days that a developer worked on bugs in a particular category.

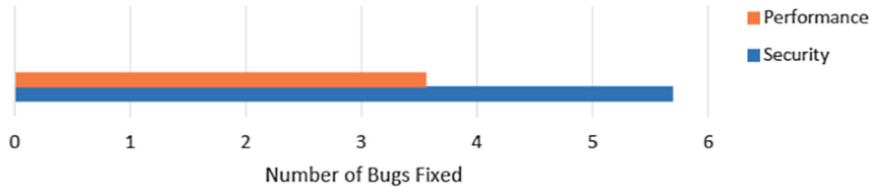


Fig. 8. Mean developer experience: the chart represents the average number of bugs that a developer fixed in a particular category. Security bugs are fixed by more experienced developers.

3.3 Characteristics of the Bug Fix

Motivation: Knowing the complexity of bug types can help guide project managers when they need to assign the right developers to fix complex bugs. Having this insight will lead to faster bugs fixes and less cost in terms of future maintenance. We hope to find a correlation similar to the one seen by Zaman et al. in which security bugs are more considered more complex than performance bugs.

Approach: To quantify the complexity of the bug fix, we used the total number of lines added/deleted as our metric.

Findings: According to Table 1 and Fig. 9, security bug fixes were found, on average, to be more complex than performance bugs. The lines of code altered in terms of security bugs was found to be ~14% [i.e. |(1813.25/1559.40) − 1813.25 * 100|] more than performance bugs.

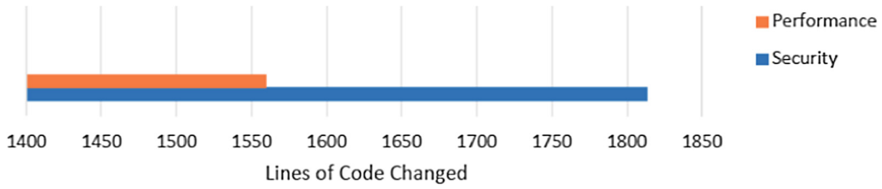


Fig. 9. Mean complexity of bug types: the bar chart represents the average LOC changed for a particular category. Security bugs are more complex and require writing or changing a higher number of LOCs.

4 Limitation of the Assessment

To obtain a fair amount of bug data for analysis, we used multiple heuristics keywords. The bugs obtained via these heuristics may not always give bug reports which pertain to performance or security. For example, our heuristic term "slow" may give us a bug which classified as a compatibility bug and not a performance or security bug. We also did not check if any of the bugs we obtained belonged to both the security and performance categories. In the future we hope to perform statistical sampling to estimate the precision and recall of our heuristics, so we can be confident that a particular bug belongs in its respective category.

It must be noted that a bug can be worked on by multiple developers at once, yet in our approach we only use what is labeled as the "Owner" of the bug report and hence only report a single developer per bug (i.e. we don't consider any other developers besides the owner). This was done to avoid distributing the lines of code added and deleted among multiple developers.

Out of the total bugs found for each category a significant amount was not considered due to missing fields. An example of some of the missing fields are as follows: no owner given, no start/fixed date given, and zero lines of code added or deleted.

5 Related Work

Our report is based upon a similar study by Zaman et al. in which the authors analyzed Firefox Bugzilla bug reports and found that security bugs are fixed and triaged much faster but are reopened and tossed more frequently than performance and other bugs [1]. They also found that security bugs involve more developers and impact more files in a project. Researcher's also use the IEEE Standard Classification for Software Anomalies to classify software defects [11]. This classification states that security and performance correspond to two out of the six different type of problems that are based on the effect of defects. Gegick et al. research identifies security bug reports via a text mining approach [12]. We, similar to the Zaman et al., suggest that developers and project managers should use the classifications of bug reports via bug type (security and performance) to improve software quality.

6 Conclusion

In this study, we analyzed the differences in time to fix, developer experience, and bug fix characteristics between security and performance bugs in order to validate our assumption that security and performance bugs are different and thus quality assurance should take into account the bug type. We found that on average, security bugs are fixed faster when compared to performance bugs. We also found on average that developers who worked on security bugs have fixed more bugs than the developers who worked on performance bugs. However, based on the number of days developers have been fixing bugs, we found that there is not a significant difference ($\sim 8\%$) between performance and security bugs. When compared to performance bugs, security bugs were considered more complex in terms of lines of code. In the future, we would hope to procure a way to fully automate our analysis, so that we may make more concise assumptions.

References

1. Zaman, S., Adams, B., Hassan, A.E.: Security versus performance bugs: a case study on firefox. In: Proceeding of the 8th Working Conference on Mining Software Repositories (2011)
2. Shihab, E., et al.: Predicting re-opened bugs: a case study on the eclipse project. In: Proceedings of the 17th Working Conference on Reverse Engineering, WCRE 2010, Washington, DC, USA, pp. 249–258 (2010)
3. Erlikh, L.: Leveraging legacy system dollars for e-business. IT Prof. **2**(3), 17–23 (2000)
4. Jaafar, F., Lozano, A., Gueheneuc, Y.-G., Mens, K.: On the analysis of co-occurrence of anti-patterns and clones. In: 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS) (2017)
5. Panjer, L.D.: Predicting eclipse bug lifetimes. In: Proceedings of the 4th International Workshop on Mining Software Repositories, MSR 2007, Washington, DC, USA (2007)
6. Weiss, C., Premraj, R., Zimmermann, T., Zeller, A.: How long will it take to fix this bug? In: Proceedings of the 4th International Workshop on Mining Software Repositories, MSR 2007, Washington, DC, USA (2007)
7. Kim, S., Whitehead Jr., E.J.: How long did it take to fix bugs? In: Proceedings of the 3rd International Workshop on Mining Software Repositories. ACM, New York (2006)
8. Anvik, J., Hiew, L., Murphy, G.C.: Who should fix this bug? In: Proceedings of the 28th International Conference on Software Engineering, ICSE 2006, pp. 361–370, New York, NY, USA (2006)
9. Apache Hadoop. https://hadoop.apache.org/. Accessed 30 Apr 2019
10. Ackerman, M.S., Halverson, C.: Considering an organization's memory. In: Proceedings of the 1998 ACM conference on Computer Supported Cooperative Work, CSCW 1998, pp. 39–48. ACM, New York (1998)

11. Draft Standard for IEEE Standard Classification for Software Anomalies. IEEE Unapproved Draft Std P1044/D00003, February 2009
12. Gegick, M., Rotella, P., Xie, T.: Identifying security bug reports via text mining: an industrial case study. In: Proceedings of the 7th International Workshop on Mining Software Repositories, pp. 11–20, May 2010