Authors' Response to the Review of EMSE-D-20-00300: "Fixing Vulnerabilities Potentially Hinders Maintainability"

Sofia Reis, Rui Abreu, Luis Cruz

Editor

> Overall, I would like to give you a chance to respond to these issues in a revision of your manuscript but I have to make it clear that it does not guarantee a path towards acceptance. Your rebuttal to #1 will be of critical importance for me judge the validity of the use of the BCH tool and in making a final decision. Methodology is, of course, very important for ESE journal and #2 & #3s comments on methodological details need to be carefully addressed, too.

Response:
We thank the editor and reviewers for their valuable feedback. In the following, we address the concerns raised by the reviewers, and consequently to your concerns too.

Reviewer 1

Reviewer comment 1.1:

> This paper investigates the impact of patches to improve security on the maintainability of open-source software. The paper is well written and easy to read. I really appreciated the replication package: complete, and well described. Really a good work!
>
> However, I found a crucial issue in this paper. The authors adopted as static analysis tool Better Code Hub. Better Code Hub' model includes 10 guidelines that can help the developers to write better code. Unfortunately, these guidelines are not related to maintainability. Avoiding introducing the issues associated to these guidelines do not imply increasing the maintainability of the code, since the tools has no possibility to measure "maintainability".

Response:
Thank you for your positive feedback on the paper presentation and replication package.

We would like to rebut your concern about Better Code Hub (BCH)'s guidelines not being related to maintainability. First, as mentioned in the paper, BCH checks GitHub codebases against 10 engineering guidelines as devised by Software Improvement Group (SIG). These guidelines are the result of many years of experience: analyzing more than 15 million lines of code every week, SIG maintains the industry's largest benchmark, containing more than 10 billion lines of code across 200+ technologies; SIG is, according to their website, the only lab in the world certified by TÜViT to issue ISO 25010:2011 certificates for Trusted Product Maintainability[1]. TÜViT is

[1] Information available here: `https://www.softwareimprovementgroup.com/methodologies/iso-iec-25010-2011-standard/`

an IT Security company. Being certified by TÜViT means that the company checked if BCH actually respects and follows the ISO 25010 guidelines.

Second, the relation of the entire set of guidelines to maintainability is well detailed in the O'Reilly's book "Building Maintainable Software: Ten Guidelines for Future-Proof Code" by Professor Joost Visser [2]. In fact, the underlying maintainability model for BCH is documented with several (non-academic) reports linked at this page: https://www.softwareimprovementgroup.com/methodologies/iso-iec-25010-2011-standard/ as well as validated by the academic community in the following paper:

```
Heitlager, Ilja, Tobias Kuipers, and Joost Visser.
''A practical model for measuring maintainability.''
6th international conference on the quality of information and
        communications technology (QUATIC 2007).
IEEE, 2007.
```

BCH's compliance criterion is derived from the requirements for 4-star level maintainability (cf. ISO 25010). The concrete values of the thresholds used in BCH are also documented in their book Building Maintainability Software https://www.softwareimprovementgroup.com/resources/ebook-building-maintainable-software/.

Further, the methodology used to derive the thresholds is documented in the following literature:

```
Tiago L. Alves, Christiaan Ypma and Joost Visser.
''Deriving metric thresholds from benchmark data.''
IEEE International Conference on Software Maintenance (ICSME 2010).
IEEE, 2010.

Tiago L. Alves, Jose Pedro Correia and Joost Visser.
''Benchmark-Based Aggregation of Metrics to Ratings.''
2011 Joint Conference of the 21st International Workshop on Software
Measurement and the 6th International Conference on Software Process
and Product Measurement.
IEEE, 2011.

Robert Baggen, Jose Pedro Correia, Katrin Schill and Joost Visser.
''Standardized code quality benchmarking for improving software
maintainability.''
Software Quality Journal.
Springer, 2011.
```

SIG performs the threshold calibration yearly on a proprietary data set to satisfy the requirements of TUViT to be a certified measurement model. However, please note that this data set cannot be shared due to non-disclosure agreements between SIG and its clients.

[2] Available here: https://www.softwareimprovementgroup.com/resources/ebook-building-maintainable-software/

All in all, we argue that BCH's guidelines are a good proxy for building maintainable software. This is backed up by the observations of SIG in the field, as it was told us in personal communications:

> We saw the value of a tool like BCH in helping in writing clean and maintainable code and most importantly, it offers us so much more: over time, the guidelines in BCH will become part of your coding standards, and the overall quality of your work will improve, slowly, but surely.

We changed the flow of the text and clarified these points in the *Introduction*. We also added a section motivating Better Code Hub called *Better Code Hub* in Section 3.

Reviewer comment 1.2:

> As far as I know, the company that developed Better Code Hub developed also the Delta Maintainability Model [diBiase2019] that aims at measuring the maintainability of a code change and a score and compare change-based maintainability measurements.
>
> [diBiase2019] M. di Biase and A. Rastogi and M. Bruntink and A. van Deursen. The Delta Maintainability Model: Measuring Maintainability of Fine-Grained Code Changes. IEEE/ACM International Conference on Technical Debt (TechDebt) in 2019.

Response:
Thank you for suggesting Di Biase et al.'s paper. We evaluate the impact of security patches in software maintainability using an alternative maintainability model [Cruz2019].

Di Biase et al.'s work was developed in parallel with the one by Cruz et al [Cruz2019]. Both models attempt to measure maintainability, but Di Biase et al.'s is not as complete since it only considers 5 of the same set of guidelines. Both models build their maintainability model over the SIG-MM model—the model behind BCH.

Our model was validated by DiBiase in several discussions between the authors of both papers.

We also mention Di Biase et al.'s work in our *Related Work* section:

> Recent work proposed a new maintainability model to measure fine-grained code changes by adapting/extending the BCH model [diBiase2019]. Our work uses the same base model (SIG-MM), but considers a broader set of guidelines. Moreover, we solely focus on evaluating the impact of security patches on software maintainability.

The literature that supports this response is the following:

```
Luis Cruz, Rui Abreu, John Grundy, Li Li, Xin Xia.
‘‘Do Energy-oriented Changes Hinder Maintainability?’’
International Conference on Software Maintenance and
Evolution (ICSME 2019).
IEEE, 2019.
```

```
M. di Biase and A. Rastogi and M. Bruntink and A. van
Deursen.
‘‘The Delta Maintainability Model: Measuring Maintainability
of Fine-Grained Code Changes.’’
IEEE/ACM International Conference on Technical Debt (TechDebt).
IEEE, 2019.
```

Reviewer comment 1.3:

> Moreover, I am not sure that Better Code Hub is the best static analysis tools to detect security issues. One of the most adopted in security domain is Coverity Scan for examples, but also other tools might be a better choice (e.g. Kiuwan).

Response:
We do not think that the concern of the reviewer is warranted. Note that the goal of our work is not to use static analysis to detect security issues. We have already a dataset of security patches (pairs of vulnerable versions and non-vulnerable versions) and our objective is to understand what the impact of those patches on the maintainability guidelines/metrics is. We argue that these guidelines/metrics may in the future complement static analysis tools by assisting developers with more information on the risks associated with their patches.

Reviewer comment 1.4:

> The authors could have adopted both approaches, Better Code Hub guidelines or another one first and then measure the code maintainability with the Delta Maintainability Model since the goal of the paper is to improve security on the maintainability.

Response:
As we explain above, namely while addressing comment 1.2, both models assess maintainability. We consider the Delta Maintainability Model to be an alternative to our model. However, it only uses 5 guidelines of the set of guidelines our study considers. This has been discussed in the related work section.

Reviewer comment 1.5:

> In my opinion, the idea behind this work is very interesting, but the research questions cannot be answered with the tools and metrics selected.
>
> My recommendation is to select a specific static analysis tool for security issues and to include a maintainability model to evaluate the maintenance.

Response:
Thank you for the positive feedback on the work.

We would like to rebut your concern about the tools and metrics we are using. As we explained previously, in comment 1.1, BCH checks GitHub codebases against 10 maintainability guidelines as devised by Software Improvement Group (SIG). SIG has devised these guidelines after

many years of experience: analyzing more than 15 million lines of code every week, SIG maintains the industry's largest benchmark, containing more than 10 billion lines of code across 200+ technologies; SIG is the only lab in the world certified by TÜViT to issue ISO 25010:2011 certificates[3].

The guidelines and their relation to maintainability is well detailed in the SIG's book "Building Maintainable Software: Ten Guidelines for Future-Proof Code" [4].

Note that our goal is to measure the impact of security patches on software maintainability and not finding vulnerabilities—vulnerabilities are already identified and located. Thus, for now, we do not need to leverage static analysis tools.

For future research, it is indeed interesting to understand how BCH can be fused with a static analysis tool and how the BCH results can help security engineers assess the risk of their patches and guide them on performing better patches. But, for now, we only try to understand the impact of security patches on those metrics.

All in all, we argue that BCH is appropriate to perform the current analysis.

Reviewer 2

Reviewer comment 2.1:

> Question: Is your dataset available?

Response:
Thank you for your question. The dataset is fully available at `dataset/db_security_changes.csv` in the figshare package provided in the contributions of the paper: `https://figshare.com/s/4861207064900dfb3372`. If the paper is accepted, the package will be available on GitHub.

This information is also available in Section 1 where we present the contributions of our work:

> This research performs the following main contributions:
>
> - Evidence that supports the trade-off between security and maintainability: developers may be hindering software maintainability while patching vulnerabilities.
> - An empirical study on the impact of security patches on software maintainability (per guideline, severity, weakness and programming language).
> - A replication package with the scripts and data created to perform the empirical evaluation, for reproducibility. Available online: `https://figshare.com/s/4861207064900dfb3372`.

[3] Information available here: `https://www.softwareimprovementgroup.com/methodologies/iso-iec-25010-2011-standard/`

[4] Available here: `https://www.softwareimprovementgroup.com/resources/ebook-building-maintainable-software/`

Reviewer comment 2.2:

> Abstract: Throughout the paper, ”hypothesize” is probably a better word than ”suspect” for sounding more scientific.
>
> Your results should be more specific than ”show evidence of trace-off”. Briefly tell us about what metrics you used and what the numerical results indicate.

Response:
We thank the reviewer for the suggestions. We replaced “suspect” by “hypothesize”. For the latter issue, we added to the abstract the results for maintainability and results for the 2 guidelines with a more negative impact on software maintainability (software complexity and unit size), in particular:

> Results show evidence of a trade-off between security and maintainability for 41.90% of the cases, i.e., developers may hinder software maintainability. Our analysis shows that 38.29% of patches increased software complexity and 37.87% of patches increased the percentage of LOCs per unit.

Reviewer comment 2.3:

> Intro:
>
> - First sentence - quality is not ONLY related to cost but also to security and safety.
> - Provide a URL to Software Improvement Group and Better Code Hub.
> - Page 3, Line 1 - Application Security Verification Standard (ASVS).
> - Page 3, Line 7 - instead of a ”broad number of code metrics” - tell us exactly the number of metrics.
> - Page 3, Line 13 - ”suggest” sounds more scientific than ”hint at”
> - Page 3, Line 24 - ”we intend to highlight the need . . . ”. Is that the broad goal of your paper? The goal should be explicitly stated in the abstract and intro.

Response:
Thank you for reporting these issues. All of them were addressed in the new version of the paper. In the last point, where the reviewer asks “Is that the broad goal of your paper?”, we want to clarify that the goal of our study is to show the impact of security patches on software maintainability and highlight the need for solutions, which are described at the end of the abstract, Section 1 (*Introduction*) and Section 5 (*Study Implications*).

Reviewer comment 2.4:

> Intro:
> You should check out this paper: [Li2017] Li, Frank and Paxson, Vern. A large-scale empirical study of security patches. ACM SIGSAC Conference on Computer and Communications Security in 2017.

Response:
Thank you for bringing our attention to this paper. We now discuss this paper in the Related Work section and used it to support some of our claims and findings.

We added the following paragraph to Section 7 (*Related Work*):

> Researchers performed a large-scale empirical study to understand the characteristics of security patches and their differences against bug fixes [Li2017]. The main findings were that security patches are smaller and less complex compared to bug fixes and usually performed at function-level. Our study compares the impact of security patches on software maintainability with the impact of regular changes.

We also used the paper to suport our research in different Sections of our paper:

Section 1 (*Introduction*)

> Our hypothesis is that some of these patches may have a negative impact on the software maintainability and, possibly, even be the cause of the introduction of new vulnerabilities—harming software reliability and introducing technical debt. Research found that 34% of the security patches performed introduce new problems and 52% are incomplete and do not fully secure systems [Li2017].

Section 3.2 (*Security Patches vs. Regular Changes*)

> Previous studies attempted to measure the impact of regular changes on open-source software maintainability. However, there is no previous work focused on comparing the impact of security patches with regular changes on maintainability, only with bug-fixes [Li2017].

Section 5 (*Study Implications*)

> ***Prioritize High and Medium Severity:*** Previous research exhibits proof that developers prioritize higher impact vulnerabilities [Li2017]. Our study shows that vulnerabilities of high and medium severity should be prioritized in software maintainability tasks.

Reviewer comment 2.5:

> Section 2
>
> - The first paragraph is redundant with Section 1.

- Page 4, Line 15 - You define the OCSP acronym late in the paragraph.
- Page 5, Line 6 - how many new branch points?
- Page 6, Line 19 - I think you want to say ”In this study, maintainability is ....” Since it currently implies this information is available in the CWE.
- Page 6, line 47 - ”regular commits” non-security commits to be more clear; maybe a few more words to explain ”randomly collected.” You also call them ”baseline commits” in Figure 1 which gives a different phrase for the same concept. Pick one and use it everywhere.

Response:
Thank you for pointing out all of these issues. We addressed all of them. We decided to use regular changes/regular commits instead of baseline and non-security commits.

Reviewer comment 2.6:

Section 3

- Sometime you say 1330 patches (e.g. page 7, line 16) and sometimes 1300 (e.g. page 8, line 11, and the abstract). In science it's better to say the exact number and use it always and not round.
- You say the dataset had 1330 (or 1300) patches and 1282 commits though you also say one patch can have multiple commits assigned. So how did you end up with less than one commit/patch?

Response:
Thank you for raising this issue. The right number of security patches is 1300 (624 from Ponta et al. and 676 from Secbench). The 1330 value is a typo, we fixed the issue. Regarding the second point, 1282 commits is the equivalent to the 624 patches that integrate the Pontas et al. dataset where one patch can have multiple commits. The 1282 commits are referring to the Ponta et al. dataset and not to the combined dataset.

> We use a combined dataset of 1300 security patches which is the outcome of mining and manually inspecting a total of 312 GitHub projects.

Reviewer comment 2.7:

Section 3

- I would really like a better picture of what SIG is to establish credibility. I was at first thinking it had something to do with ACM SIG's but now after going to the website, it seems like a commercial/consulting organization? I also feel irritated that I don't understand what you really mean by ”running against the BCH toolset” (page 8, line 13) since you have repeatedly mentioned BCH like it was an industry standard - but you have not told me if

> it's a static analysis tool or what it is. The explanation and link to introduce SIG and BCH should be on Page 2 lines 41-42 rather than Page 7 - though an explanation of the BCH toolset is best a paragraph in the methodology.

Response:
Thank you for raising the concern regarding SIG's credibility. SIG is a company that has more than 20 years of experience and research in software quality production. Their models are scientifically proven and certified. We added this information and the links the reviewer refer in the comment to the Introduction section.

> As ISO does not provide any specific guidelines/formulas to calculate maintainability, we resort to Software Improvement Group (SIG[5])'s web-based source code analysis service Better Code Hub (BCH)[6] to compute the software compliance with a set of 10 guidelines/metrics to produce quality software based in ISO/IEC 25010 [Visser2016]. SIG has been helping business and technology leaders drive their organizational objectives by fundamentally improving the health and security of their software applications for more than 20 years. Their models are scientifically proven and certified [Alves2010, Alves2011, Baggen2012].

We rephrased "running against the BCH toolset" to "analyzed using the BCH toolset". We mean that we used BCH to calculate the metrics presented on Table 1.

The literature that supports this response is the following:

```
J. Visser.
‘‘ Building Maintainable Software, Java Edition: Ten Guidelines
for Future-Proof Code’’
O’Reilly Media, Inc. 2016.

Tiago L. Alves, Christiaan Ypma and Joost Visser.
‘‘Deriving metric thresholds from benchmark data.’’
IEEE International Conference on Software Maintenance (ICSME 2010).
IEEE, 2010.

Tiago L. Alves, Jose Pedro Correia and Joost Visser.
‘‘Benchmark-Based Aggregation of Metrics to Ratings.’’
2011 Joint Conference of the 21st International Workshop on Software
Measurement and the 6th International Conference on Software Process
and Product Measurement.
IEEE, 2011.

Robert Baggen, Jose Pedro Correia, Katrin Schill and Joost Visser.
‘‘Standardized code quality benchmarking for improving software
maintainability.’’
```

[5]SIG's website: `https://www.sig.eu/` (Accessed on April 30, 2021)
[6]BCH's website: `https://bettercodehub.com/` (Accessed on April 30, 2021)

Software Quality Journal.
Springer, 2011.

Reviewer comment 2.8:

> Section 3
>
> - Page 7, line 4 - I don't think "resemble" is the word you are looking for - but I don't know what you are trying to say so I can't make a suggestion.

Response:
Thank you for raising this issue. In the sentence "The codebases that resemble to the commits of our datasets are collected before the analysis performed by BCH.", we mean that the codebases of each commit in our dataset were analyzed by BCH. We improved the text for:

> BCH evaluates the codebase available in the default branch of a GitHub project. We created a tool that pulls the codebase of each commit of our dataset to a new branch; it sets the new branch as the default branch; and, runs the BCH analysis on the codebase; after the analysis is finished, the tool saves the BCH metrics results to a cache file.

Reviewer comment 2.9:

> Section 3
>
> - Please explain your repeatable methodology to "clean the dataset and select the most relevant and compatible projects" (page 8, line 14) that brings the patches from 1282 to 969.
> - From 969 to 866 patches - you just could not classify? Please explain.

Response:
As explained in comment 2.6, the 1282 commits refer only to the Ponta et al. dataset (a total of 624 patches). In total, we analyzed 1300 patches (624 patches from Ponta et al. and 676 from Secbench). However, our study only considers 969 patches. We detected a total of 23 patches with floss-refactorings. In addition, BCH has some limitations, in particular, the lack of language support and project size, and was uncapable of analyzing the rest of the 308 data points. All of this is explained in more detail later in Section 3.4.

> Security patches can be performed through one commit (single-commit); several consecutive commits (multi-commits); or, commit(s) interleaved with more programming activities (floss-refactoring). Only 10.7% of the data points of our dataset involve more than one commit, the other 89.3% of the cases are single-commit patches. We manually inspected 25.1% ($244/969$) of the security patches—122 from each

dataset—and, identified a total of 23 floss-refactorings. Many of these are patches with many changes where it is hard to understand which parts involve the security patch. These were disregarded to minimize the impact of measuring other programming activities rather than solely security patches can have in our results.

For projects with large codebases, the results calculated for the *Keep Your Codebase Small* guideline were way above the limit set by BCH (20 Person-years). We suspect this threshold may not be well calibrated, and hence biasing our results. Thus, we decided not to consider this guideline in our research. The *Automated Tests* guideline was also not considered since the tool does not contemplate two of the most important techniques to security testing: vulnerability scanning and penetration testing. Instead, it only contemplates unit testing. In total, we detected 308 data points suffering from these two limitations. Those data points were disregarded from the study.

Those are the cases that were tossed from the initial 1300 patches.

Regarding the second question: while manually inspecting the patches, we were not able to map the issue to any CWE with confidence due to the lack of quality information on the vulnerability/patch for 103 patches. We clarified this in the paper.

A total of 103 patches were not classified because we were not able to map the issue to any CWE with confidence due to the lack of quality information on the vulnerability/patch.

Reviewer comment 2.10:

> Section 3
>
> - Page 8, line 33 ”regular commit” - pick baseline or regular (or non-security) commits and use it always. I don’t understand the sentence ”The baseline dataset is generated from the security commits dataset.” Do you mean you extract these commits from the projects in the security commits dataset?

Response:
Yes, for each security commit we collected a random regular change from the same project. We have clarified it in the paper.

The baseline dataset is generated from the security commits dataset, i.e., for each security commit in the dataset, we collect a random regular change from the same project.

Reviewer comment 2.11:

Section 3

- Section 3.2 is confusing enough that I'm not sure of the analysis - I will need to review the revision that you would submit to gain confidence. Are you saying you want your non-security commits (regular commits) to be about the same size as a given security commit so you are using this criteria to choose your regular commits? Justify this reasoning and make the methodology clear. I would have expected that you randomly collected 1300 non-security commits to compare with the 1300 security commits. Why constrain to make them be of "similar" size? Why not allow the characteristics of regular versus security commits to be what they are without such curating?

Response:
Yes, we are looking for regular changes with the same size as security commits from the same project. However, for some cases, it was difficult to find the exact same size. Thus, we searched for an approximation. Every 10 attempts of failling to get a commit with the same size, we spanned the range size. We clear the methodology in the paper in Section 3.2 *(Security Patches vs. Regular Changes)*.

"I would have expected that you randomly collected 1300 non-security commits to compare with the 1300 security commits.": Although we started with 1300 commits, 331 cases were tossed due to BCH limitations. In addition, BCH takes substantial time to process this amount of cases. Since the comparison is between the final dataset of 969 patches and the baselines, we produced baselines with the final number of viable results obtained for the security dataset.

We have collected a baseline of random changes only (no size restrictions) in the past and performed the same evaluation. We now present the results for both baselines: *size-baseline*, where we consider size appproximation; and, *random-baseline*, where we consider all the regular changes characteristics.

Reviewer comment 2.12:

Section 3

- Since SIG seems to be "just another consulting organization" - I'd like to see about the acceptance of the 10 guidelines in the software engineering discipline. I would guess you could find peer-reviewed references to support these guidelines, or even books by Martin Fowler. That has more credibility than the reference by the SIG consulting group. For example on Page 9, line 42 you cite McCabe Complexity – that has more credibility than a consulting organizations guidelines that they use to make money on their tool. Tell us more about the BCH data experience since that is your comparison point for compliance.

Response:
All the guidelines are fully described on their book "Building Maintainable Software: Ten Guidelines for Future-Proof Code" [7].

[7] Available here: `https://www.softwareimprovementgroup.com/resources/ebook-building-maintainable-software/`

We found other papers using some of the metrics measured by BCH. But we did not find extra theoretical papers explaining metrics in detail as we found for McCabe Complexity. However, metrics like Unit Size, Duplication, Unit Interfacing, Volume, Testability and Code Smells are well-known metrics in the Software Engineering field.

We would like to address your concern regarding the credibility of BCH. BCH checks GitHub codebases against 10 engineering guidelines as devised by Software Improvement Group (SIG). SIG has devised these guidelines after many years of experience: analyzing more than 15 million lines of code every week, SIG maintains the industry's largest benchmark, containing more than 10 billion lines of code across 200+ technologies; SIG is the only lab in the world certified by TÜViT to issue ISO 25010:2011 certificates[8].

The underlying maintainability model for BCH is documented with several (non-academic) reports linked at this page: `https://www.softwareimprovementgroup.com/methodologies/iso-iec-25010-2011-standard/` as well as validated by the academic community in the following paper:

```
Heitlager, Ilja, Tobias Kuipers, and Joost Visser.
‘‘A practical model for measuring maintainability.’’
6th international conference on the quality of information and
        communications technology (QUATIC 2007).
IEEE, 2007.
```

BCH's compliance criterion is derived from the requirements for 4-star level maintainability (cf. ISO 25010). The concrete values of the thresholds used in BCH are documented also in their book Building Maintainability Software `https://www.softwareimprovementgroup.com/resources/ebook-building-maintainable-software/`.

Further, the methodology for arriving at the thresholds is documented in the following literature:

```
Tiago L. Alves, Christiaan Ypma and Joost Visser.
‘‘Deriving metric thresholds from benchmark data.’’
IEEE International Conference on Software Maintenance (ICSME 2010).
IEEE, 2010.

Tiago L. Alves, Jose Pedro Correia and Joost Visser.
‘‘Benchmark-Based Aggregation of Metrics to Ratings.’’
2011 Joint Conference of the 21st International Workshop on Software
Measurement and the 6th International Conference on Software Process
and Product Measurement.
IEEE, 2011.

Robert Baggen, Jose Pedro Correia, Katrin Schill and Joost Visser.
‘‘Standardized code quality benchmarking for improving software maintainability.’’
Software Quality Journal.
Springer, 2011.
```

[8] Information available here: `https://www.softwareimprovementgroup.com/methodologies/iso-iec-25010-2011-standard/`

SIG performs the threshold calibration yearly on a proprietary data set to satisfy the requirements of TUViT to be a certified measurement model. However, please note that this data set cannot be shared due to non-disclosure agreements between SIG and its clients. BCH has a free plan available that allows the analysis of public projects with a size limit of 100.000 lines of code.

Reviewer comment 2.13:

> Section 3
>
> - Page 10, line 49 ”mainly single-commit patches” . . . ”small percentage of data points” - give us the exact numbers for each of these.
> - Page 11, line 38 - define ”floss refactoring” and justify how these were objectively and repeatably identified in your set. You say ”these cases” which makes it seems that you were including the set of patches that had more than one commit as your (only) floss candidates.

Response:
Our dataset integrates 89.3% of single-commit patches and 10.7% of patches involving more than one commit. We replaced line 49 by the following sentence:

> Only 10.7% of the data points of our dataset involve more than one commits, the other 89.3% of the cases are single-commit patches.

We define floss-refactoring with the paragraph below. To mitigate the impact of floss-refactorings, we extracted and manually inspected a random sample with 25% of security patches from each dataset. From this sample, we identified 23 floss-refactorings. Most floss-refactoring patches include many changes making it difficult to understand which parts involve the security patch. Although we suspect that more floss-refactorings may occur, we argue that they occur in a small portion of the data. We clarified this in the paper:

> Security patches can be performed through one commit (single-commit); several consecutive commits (multi-commits); or, commit(s) interleaved with more programming activities (floss-refactoring). Only 10.7% of the data points of our dataset involve more than one commit, the other 89.3% of the cases are single-commit patches. To mitigate the impact of floss-refactorings, we extracted and manually inspected a random sample with 25% of security patches from each dataset. From this sample, we identified 23 floss-refactorings. Most floss-refactoring patches include many changes making it difficult to understand which parts involve the security patch. Although we suspect that more floss-refactorings may occur, we argue that they occur in a small portion of the data.

Reviewer comment 2.14:

Section 3

- Page 11, Line 42 - I think you are saying you removed these two guidelines from the analysis of all projects (as shown in Figure 4) but this paragraph seems to say this was only for projects with large code bases.

Response:
We started with all the guidelines but after analyzing the BCH results, we found that some of the results for the *Keep Your Codebase Small* guideline were way above the limit set by BCH (20 Person-years). We suspect this threshold may not be well calibrated, and hence biasing our results. Thus, we decided not to consider this guideline in our research. We did not want to lose more data points. Thus, we decided to not consider the guideline for all projects.

We also did not consider the *Automated Tests* guideline because this guideline does not contemplate vulnerability scanning and penetration testing which are the two most important testing techniques in security.

We improved the paper in the following paragraph:

> Due to BCH limitations, in particular, lack of language support and project size by BCH, 308 data points were not analyzed and automatically disregarded from our study. After performing the BCH analysis and the maintainability calculations, we found the following limitations regarding two of BCH's guidelines:
>
> 1) For projects with large codebases, the results calculated for the *Keep Your Codebase Small* guideline were way above the limit set by BCH (20 Person-years). We suspect this threshold may not be well calibrated, and hence biasing our results. Thus, we decided not to consider this guideline in our research.
>
> 2) The *Automated Tests* guideline was also not considered since the tool does not contemplate two of the most important techniques to security testing: vulnerability scanning and penetration testing. Instead, it only contemplates unit testing.

Reviewer comment 2.15:

Section 3

- Page 11 line 48. How many were disregarded?

Response:
Due to BCH limitations, in particular, lack of language support and project size by BCH, 308 data points were disregarded. We clarified it in the paper (Section 3.4):

> Due to BCH limitations, in particular, lack of language support and project size by BCH, 308 data points were not analyzed and automatically disregarded from our study.

Reviewer comment 2.16:

> Section 4
>
> - Page 13, Line 3 - 969 security commits and 969 baseline commits? It's hard to know how to parse that.

Response:
We mean that our study is an empirical evaluation of 969 security patches and 969 regular changes. We improved the sentence.

> This study evaluates a total of 969 security patches and 969 regular changes from 260 distinct open-source projects.

Reviewer comment 2.17:

> Section 4
>
> - Page 13, Line 31 "overall patches . . . " rather than having us rely on looking at the Figure, tell us the number or percentage of when the impact is positive like you do with the negative in the next sentence. [You do provide the specifics starting on page 15 line 49. I wanted it here.]

Response:
We replaced overall by the number of patches where maintainability increases (38.7%).

> Regarding the impact of security patches per guideline, we observe that 38.7% of the security patches have positive impact on software maintainability.

Reviewer comment 2.18:

> Section 4
>
> - Page 19, Line 7 - for the unindoctrinated - I'd explain the concept of Research Concepts. Also explain your choice of 7a and 7b and how representative they are from the several hundred choices. Similar for your explanations in the second two paragraphs on this page. How far down the concept/vulnerability hierarchy tree are these CWE choices or are they lower level vulnerabilities chosen from the 700+ CWE types?

Response:
We added an explanation of the Research Concepts list to the paper in a footnote:

> Research Concepts is a tree-view provided by the Common Weakness Enumeration (CWE) website that intends to facilitate research into weaknesses. It is organized

according to abstractions of behaviors instead of how they can be detected, their usual location in code, and when they are introduced in the development life cycle. Available here: `https://cwe.mitre.org/data/definitions/1000.html`

In Figure 7a, we present the entire distribution of the 969 vulnerabilities per each node of the 1st-level of the Research Concepts tree. Looking at those results, we detected that the percentage of cases for CWE-707 (30.4%) and CWE-664 (32.8%) were considerably higher in the dataset than the rest of the CWEs. Thus, we decided to look at lower levels of behavior for both CWEs: CWE-707 results are presented in Figure-7b and CWE-664 results are presented in Figure-7c. In Figure-7b, all the CWEs presented are sub levels of the CWE-707. The same happens for Figure-7c, but the CWEs are sub levels of the CWE-664.

We added the previous numbers to the paper and clarified that we are considering lower levels of vulnerabilities.

> In *RQ2*, we report/discuss the impact of security patches on software maintainability per weakness (CWE). We use the weakness definition and taxonomy proposed by the *Common Weakness Enumeration* (cf. Section 2). Figure 7 shows three different charts. Figure *7-a*, presents the impact of the 969 patches grouped by the first level weaknesses from the *Research Concepts* list. While the Figures *7-b* and *7-c* present the impact on maintainability for lower levels of weaknesses for the most prevalent weaknesses in Figure *7-a*: *Improper Neutralization* (CWE-707) and *Improper Control of a Resource Through its Lifetime* (CWE-664), respectively.
>
> In Figure 7-*a*, there is no clear evidence of the impact on maintainability per weakness. Yet, it is important to note that overall there is a very considerable number of cases that hinder maintainability—between 30% and 60%. The CWE-707 and CWE-664 weaknesses integrate the higher number of cases compared to the remaining ones: 295 (30.4%) data points and 318 (32.8%) data points, respectively. Thus, we present an analysis of their sub-weaknesses on Figure 7-*b* and Figure 7-*c*, respectively.

Reviewer comment 2.19:

> Section 4
>
> - Page 20, Line 42 - how many is ”a considerable number of cases”?

Response:
In total, we inspected around 25 regular changes with no impact on software maintainability to understand what was leading to the higher number of cases. This point was clarified in Section 4, with the following paragraph:

> Overall, the results for both baselines, show that regular changes are less prone to hinder the software maintainability of open-source software. However, the *size-baseline* integrates a larger number of cases with no impact on software maintainability. We manually inspected a total of 25 cases from that distribution of regular changes with no impact on maintainability, and, found that identifying regular

changes with the same size as the security-related commit is limiting the type of regular commits being randomly chosen: input patches, variables or functions, type conversion (i.e., changes with no impact on the software metrics analyzed by BCH).

Reviewer comment 2.20:

> Section 4
>
> - Page 20, Line 45 - it seems choosing commits of similar size to the security commits ended up causing a limitation. I'm still wondering why you chose that methodology anyway.

Response:
As we described in comment 2.19, we manually inspected 25 cases. Many of those cases were changes of small size (< 10 LOCs) where refactorings were usually improving names of variables, functions, cases of type conversion, etc. Changes that have no impact on the metrics we are evaluating.

Our concern by presenting a total random baseline was that comparing regular changes with security changes with very different sizes—a scenario that could happen—was unfair. Thus, we chose to create this baseline. However, we also curated a total random baseline in the past and we now present both baselines in the paper.

Reviewer comment 2.21:

> Section 4
>
> - RQ3 is answered to succinctly and with too little detail to be convincing to me that the differences between security and regular changes is different.

Response:
We improved the discussion of the comparison between the results for security patches and regular changes. We now consider two baselines: *size-baseline*, where we collected random changes with the same size as security patches; and, *random-baseline*, where we collected random changes without any restrictions. The overall results show that regular changes are less prone to hinder software maintainability than security patches. Security patches results have more cases with negative impact on software maintainability than any baseline of regular changes. By providing results for both baselines, we now clearly see that the limitation found for the *size-baseline* does not change the need to pay more attention to security patches in software maintenance tasks.

Reviewer comment 2.22:

> Section 5
>
> - Line 32 - since RQ3 was not convincing, I don't feel a separate effort for maintainable

> security is justified as separate from general focus on maintainability.

Response:
As explained in the previous response, security patches hinder more software maintainability than regular changes with size restriction (*size-baseline*) or no size restriction (*random-baseline*). Thus, we argue that special attention should be given to maintainable security when teaching software maintainability. We improved the analysis provided in Section 4 for the research question 3 (RQ3). We added an extra baseline for comparison with no size restrictions (*random-baseline*) and improved the discussion over our results.

Reviewer comment 2.23:

> Section 6
>
> - It might not be a threat - but I feel the curation of the non-security commits is a limitation to the generalizability of the characteristics of non-security commits

Response:
We agree with the reviewer. Our concern by presenting a total random baseline was that comparing regular changes with security changes with very different sizes—a scenario that could happen—was unfair. Nevertheless, we also feel that considering size may be limitating the generalizability of the characteristics of regular changes. Thus, we added to RQ3 our initial random baseline and completed the work by presenting now two baselines (size-baseline and random-baseline).

Reviewer 3

Reviewer comment 3.1:

> The paper addresses a relevant and up to date topic. The authors have analyzed a large set of project wrt to quality assessment focused on vulnerabilities in line with ISO25000 standard. There are some points worth considering: - authors should better motivate the choice for BCH as web based source code analysis service with respect to other options that are available in literature. Tools such as Kiuwan, sonar cloud, codify etc are also able to provide specific metrics on maintainability and security, vulnerability features of code. Furthermore Kiuwan is also an example designed based on the ISO25000 standard. Authors discuss the need for tools for risk assessment in the study implications (section5) but fail to assess why their choice falls on BCH. Furthermore all tools provide different conceptualization and evaluation criteria for calculating quality characteristics such as Vulnerabilities and Security. Authors should also explicit how these quality characteristics are measured in BCH.

Response:
We kindly thank the reviewer for the comments.

As mentioned in the paper, BCH checks GitHub codebases against 10 engineering guidelines as devised by Software Improvement Group (SIG). SIG has devised these guidelines after many

years of experience: analyzing more than 15 million lines of code every week, SIG maintains the industry's largest benchmark, containing more than 10 billion lines of code across 200+ technologies; SIG is the only lab in the world certified by TÜViT to issue ISO 25010:2011 certificates for Trusted Product Maintainability[9].

Furthermore, the relation of the entire set of guidelines to maintainability is well detailed in the O'Reilly's book "Building Maintainable Software: Ten Guidelines for Future-Proof Code" by Professor Joost Visser [10].

In this study, to clarify, we only focus on measuring software maintainability, we do not focus on vulnerability detection or measuring security. Instead, we want to understand how software maintainability is impacted by security patches. However, as we explained before, BCH is based on the ISO 25010 which considers *Security* as one of the main software product quality characteristics since 2011.

There are other tools that measure the same metrics such as as Kiuwan; or, even provide a model to calculate maintainability such as SonarCloud. In this study, it is not our goal to study other tools. However, we argue that BCH is a better tool for the following reasons:

- Kiuwan measures 4 guidelines that by name appear to be similar to some of the guidelines measured by BCH: Complexity, Duplicated Code, Size and Coupling[11]. However, we did not find the full description of kiuwan's metrics anywhere. Thus, we are ensure that the metrics are the same but we are able to conclude that BCH integrates a larger set of metrics. In contrast, SIG provides a full description of the 10 guidelines/metrics behind better code hub plus their relation with maintainability on the book and several scientific papers.
- SonarCloud measures 5 similar metrics to the ones measured by BCH: Complexity, Duplication, Issues, Size and Tests[12]. The tool rates maintainability with a letter based on the technical debt ratio which is calculated as the ratio between the cost to develop the software and the cost to fix it. But no information is provided on how the remediation cost and developmet cost are calculated. Better Code Hub contemplates a larger set of maintainability metrics and our formula for maintainability is not only fully explained in our paper but also based on metrics that are fully and publicly described.

We do not know codify and did not find the tool anywhere in static analysis tools collections or even through an organic search at Google.

We added a Section to the paper called *3.3 Better Code Hub* where we motivate the Better Code Hub tool and compare it with Kiuwan and SonarCloud.

Reviewer comment 3.2:

[9] Information available here: `https://www.softwareimprovementgroup.com/methodologies/iso-iec-25010-2011-standard/`

[10] Available here: `https://www.softwareimprovementgroup.com/resources/ebook-building-maintainable-software/`

[11] List available here: `https://www.kiuwan.com/docs/display/K5/Metrics`

[12] List available here: `https://sonarcloud.io/documentation/user-guide/metric-definitions/`

> In section 3a better state/formulate the hypothesis authors analyze wrt to the Research questions in previous sections. In general the empirical study is not structured according to any of the guidelines or explicit states any of them.

Response:
In our work, we follow the same methodology as the previous work mentioned below but instead of measuring the impact of energy-oriented changes on software maintainability, we measure the impact of security patches.

```
Luis Cruz, Rui Abreu, John Grundy, Li Li and Xin Xia.
‘‘Do Energy-oriented Changes Hinder Maintainability?.’’
IEEE International Conference on Software Maintenance and
Evolution (ICSME).
IEEE, 2019.
```

Improving software security is not a trivial task and requires implementing patches that might affect software maintainability. Our hypothesis is that some of these patches may have a negative impact on the software maintainability. With this study, we want to understand which maintainability metrics can guide security engineers on producing patches with better quality—which we evaluate with ***RQ1: What is the impact of security patches on the maintainability of open-source software?***. In this research question, we present the results by metric, overall score, severity and programming language to have a better understanding of the impact of each of these parameters in software maintainability.

Different weaknesses may require patches with different characteristics. Our hypothesis is that security patches for different weaknesses can have different impacts on software maintainability. Thus, we evaluate this in ***RQ2: Which weaknesses are more likely to affect open-source software maintainability?***. The results can help security engineers prioritizing weaknesses and bring awareness to the ones that need more attention.

Performing a regular change/refactoring, for instance, to improve the name of a variable or function, is different than performing a security patch. Therefore, we created ***RQ3: What is the impact of security patches versus regular changes on the maintainability of open-source software?*** to explore if there is a need to pay more attention to security patches than regular changes during software maintenance tasks.

The previous information is already included in the paper. However, we clarified these points in Section 2, where we present the research questions and our hypothesis.

Reviewer comment 3.3:

> The findings of the research should better emphasize what the lessons learned and what better practices should be put in place by developers to assure better quality of software. The take away message remains hindered.

Response:
We agree with the reviewer that we should better emphasize the lessons learned and what best practices can be followed by developers to assure better vulnerability patches. We improved the discussion of these points in Section 5.