



---

<sup>b</sup>  
**UNIVERSITÄT  
BERN**

# **Exploring Security Issues in Open Source Software**

## **Bachelor Thesis**

Noah Bühlmann  
from  
Sursee LU, Switzerland

Faculty of Science  
University of Bern

19 June 2020

Prof. Dr. Oscar Nierstrasz  
Dr. Mohammad Ghafari  
Software Composition Group  
Institute of Computer Science  
University of Bern, Switzerland

# Abstract

**Background:** Identifying and resolving security issues in open source projects are paramount due to the large impact of these projects in the software industry. Previous work has mostly focused on the prediction of security issues, but how such issues evolve and are discussed is less well-known.

**Aims:** With this study we aim to understand how security issues evolve, learn what their characteristics are and pave the way for further, more detailed research on very specific elements of the security issue life cycle.

**Method:** We performed a large-scale analysis of metadata and quantitative features of security and non-security issues from 182 different GitHub projects. Additionally, we performed a more detailed manual analysis on a subset consisting of 333 security issues, where we also investigated the issue discussions and any related pull requests.

**Results:** We were able to identify differences and similarities between security and non-security issues and describe various characteristics of security issues. Furthermore, we were able to identify differences and similarities between accepted, pending and rejected security issues and speculate on factors that influence the probability of a security issue to be accepted. Finally, we were able to describe various characteristics of security issues that result in a different resolution time.

**Conclusion:** We were able to present an overview of security issues in Open Source Software that will be useful for both researchers and practitioners in understanding and further improving the security culture in Open Source Software engineering.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Related Work</b>	<b>4</b>
<b>3</b>	<b>Background</b>	<b>7</b>
<b>4</b>	<b>Methodology</b>	<b>16</b>
4.1	Selection of GitHub projects . . . . .	16
4.2	Classification of security issues . . . . .	17
4.3	Large-scale analysis . . . . .	18
4.3.1	Features in issue dimension . . . . .	18
4.3.2	Features in repository dimension . . . . .	20
4.3.3	Features in comment dimension . . . . .	20
4.4	Sampling procedure for manual analysis . . . . .	21
4.5	Manual analysis . . . . .	22
4.5.1	Features in issue dimension . . . . .	22
4.5.2	Features in comment dimension . . . . .	26
<b>5</b>	<b>Results</b>	<b>30</b>
5.1	Prevalence and emerging of security issues . . . . .	30
5.2	Reporters of security issues . . . . .	32
5.3	Reports of security issues . . . . .	34
5.4	Reaction to security issues . . . . .	35
5.5	Resolution of security issues . . . . .	37
5.6	Discussion of security issues . . . . .	38
5.6.1	Participants in discussion . . . . .	38
5.6.2	Comments in discussion . . . . .	39
5.7	Assignment of security issues . . . . .	45
5.8	Pull requests related to security issues . . . . .	46
5.9	Trends in security issues over the age . . . . .	51
5.10	Trends in security issues over the years . . . . .	57

5.11 Other general findings . . . . .	61
5.11.1 Role of bots . . . . .	61
5.11.2 Tools to detect security issues . . . . .	61
5.11.3 Rejected security issues . . . . .	61
5.11.4 CVE entries . . . . .	62
5.11.5 Unique issues . . . . .	62
<b>6 Discussion</b>	<b>63</b>
<b>7 Threats to validity</b>	<b>69</b>
7.1 External validity . . . . .	69
7.2 Internal validity . . . . .	70
<b>8 Conclusion and future work</b>	<b>71</b>
8.1 Conclusion . . . . .	71
8.2 Future work . . . . .	71
8.3 Acknowledgement . . . . .	72
<b>A Anleitung zum wissenschaftlichen Arbeiten</b>	<b>78</b>
A.1 GitHub GraphQL API . . . . .	78
A.2 Environment . . . . .	79
A.3 Basics . . . . .	80
A.4 Searching repositories . . . . .	82
A.5 Downloading issues and comments . . . . .	83

# 1

## Introduction

Open Source Software development has become impressively popular in recent years. For instance, GitHub, the leading software development platform worldwide, has more than 40 million developers who have closed 20+ million issues only in 2019.<sup>1</sup> The advancements in Open Source Software have encouraged the software industry and large companies such as Google and Facebook to open source their otherwise proprietary software for reasons such as an engaging, large community, obtaining prompt responses, and getting fast feedback [13]. With the increasing adoption of Open Source Software in the industry, the impact of security issues in such software increases as well.

HeartBleed [2] and the Equifax [15] Breach are two remarkable examples of vulnerabilities located in Open Source Software. The former vulnerability, located in the OpenSSL library (CVE-2014-0160), exposed an enormous number of secrets to the Internet, and the latter, located in the Apache Struts 2 (CVE-2017-5638), leaked the private records of more than 140 million customers of Equifax.

Timely reaction to security issues has received great attention recently. For instance, Jiang et al. developed a model to detect security bug reports from thousands of bug reports automatically [10].

Zhang et al. conducted a large-scale study to understand whether machine learning models can detect the abundance of vulnerabilities in an application [38]. They concluded that such models help, but no single feature, such as the code complexity, has reliable prediction power.

Younis et al. developed prediction models to discern vulnerabilities that have an

---

<sup>1</sup><https://octoverse.github.com>

exploit versus those without an exploit [34].

Wang et al. developed a toolset to identify secret security patches in Open Source Software [31]. These patches fix vulnerabilities that are neither reported to CVE<sup>2</sup>, nor are they mentioned in the software changelogs.

Mu et al. investigated the reproducibility of 368 crowd-reported vulnerabilities and found that missing information in such reports is prevalent, and that security professionals should heavily rely on manual debugging and speculation to infer the missed information [19].

Finally, it is the developer's responsibility to discuss security issues and resolve them. However, when such discussions happen and how they progress are mostly unexplored.

In this thesis we aim to understand how prevalent security issues are, how they evolve, and how developers discuss such issues in Open Source Software. In particular, we ask the following research questions.

1. How prevalent are security issues and how do they evolve?
2. What are the characteristics of security issues and their discussions?

In order to answer these questions, we collected a dataset of nearly 250 000 issues from 182 different GitHub Java projects and classified the issues into security issues and non-security issues using a labelling-based approach. We then performed a large-scale analysis of various features of security and non-security issues together with the comments of their discussions. Furthermore, we selected a small sample of 333 commented security issues from our full dataset in order to perform a more qualitative analysis of the security issue reports and their comments. During our study we investigated different areas of security issues such as prevalence, reporters, reports, reaction, resolution, discussion, assignment and pull requests of security issues.

We found that security issues are more often reported by core project members than non-security issues. Moreover, security issue have a faster initial reaction time and a slower proceeding discussion than non-security issues. Also, security issues are resolved significantly slower than non-security issues. Furthermore, we could identify that accepted security issues are more clearly explained, more often contain documentation and reproducibility information and have a significantly higher proportion of thematically relevant comments compared to rejected security issues. Also, security issues that have an assignee have a significantly higher acceptance rate than security issues which do not have an assignee. Finally, we found that security issues are resolved significantly faster when the issue report or the discussion contains CVE information, when the issue reporter is involved in the pull request process and when there is a fast initial reaction to them.

---

<sup>2</sup>Common Vulnerabilities and Exposures

Unexpectedly, we also found that the prevalence of CVE information has not changed significantly over the past six years and that the resolution time of security issues is longer if further documentation is provided in the comments of the discussion.

We believe that this work has several implications for researchers and practitioners. Practitioners should include detailed documentation or reproducibility information in their security issue reports and explain them clearly. Moreover, they should create a pull request for fixing security issues to have a more involving discussion on the changes to be made. Researchers, on the other hand, can leverage our findings regarding security vs. non-security issues, accepted vs. rejected security issues or the resolution time of security issues in order to build or refine prediction models that are used for security bug report prediction or to determine the outcome of security issues.

Further investigations revealed that there is no significant difference in the acceptance rate for male vs. female security issue reporters. We were also able to identify a small group of 328 developers who were assigned to security issues. We found that this group of developers was responsible for reporting more than 40% of all security issues and more than one-third of all comments in security issue discussions. These security assignees also had a significantly higher acceptance rate and lower resolution time for reported security issues than other developers.

The remainder of this thesis is organised as follows. In chapter 2 we give an overview of related work in the research field of security issues. In the following chapter 3 we provide background information on the concept of issue tracking on GitHub and the used terminology. Chapter 4 describes the methodology we followed throughout our study. Our results are presented in chapter 5 and discussed in chapter 6. We report possible threats that could impact the validity of our results in chapter 7. Finally, we deliver our conclusion in chapter 8.

# 2

## Related Work

There has been a lot of recent research in the area of security issues (mostly called security bug reports) especially in the field of security bug report classification, i.e. the classification of bug reports into security-related and non-security-related classes. In this chapter we present the state-of-the-art in security issue and GitHub research and how this study connects to previous work.

Several stages of the software development process have been addressed by various authors. Vasilescu et al. analysed quality and productivity outcomes relating to continuous integration on GitHub [30] and found that productivity of project teams is increased by the use of continuous integration functionality without an observable diminishment in code quality. Another study has proposed a data mining model that leveraged GitHub and other software repository services to predict the complexity of software bugs [21]. This model first uses existing techniques to estimate the fix time of a bug, and then maps it to a complexity cluster. Also concerning bugs, an empirical study by Zhang et al. from 2012 has identified concrete factors that influence bug fixing time and delays [37]. Among the most influential factors were the type, severity and description of a bug, but also the used operating system and the comments of a bug played an important role. Two related studies by the same research team investigated how, in the context of GitHub, social and technical factors on the one hand [28] and discussions on the other hand [29] could be used to evaluate contributions in GitHub. Pull requests, which are an essential functionality of GitHub, have also attracted the interest of research. Pull request discussion texts, project-specific information and developer-specific information from a large-scale dataset were used by Rahman and Roy in their paper in order to contrast between successful and unsuccessful pull requests [24]. They found an increasing failure



rate of pull requests in projects with more than 3000 forks or 4000 developers, and suggest that developers are most productive when they have between 20 and 50 months of experience. A more specific study from 2015 tried to identify determinants of pull request evaluation latency and found that latency is a complex issue, requiring many independent variables to be explained adequately, such as the size of the pull request, the delay to the first human response and the availability of continuous integration [35]. Finally, a large systematic mapping study of software development with GitHub has been conducted in 2017 that analysed 80 publications from 2009 until 2016. Consentino et al. raised concerns about how reliable those publications are “given that, overall, papers use small datasets, employ a scarce variety of methodologies and/or are hard to replicate” [5].

Multiple studies from the last decade also addressed the matter of using GitHub as a source for research. In 2012 a project called “GHTorrent” was set up with the aim to create a scalable offline mirror of GitHub’s data [8]. Unfortunately, we could not use this valuable tool as the project was discontinued in 2019. Later in 2014, Kalliamvakou et al. published “The Promises and Perils of Mining GitHub” [11], which proved to be very useful for our study because it provides peril avoidance strategies that we could apply. A long article by Munaiah et al. focused on selecting projects from GitHub through the use of a proposed framework in order to separate the signal (e.g. repositories containing so-called “engineered software projects”) from the noise (e.g. repositories containing personal experiments) [20]. Finally, there are two very recent papers concerning data collection. Wu et al. proposed a CVE-assisted large-scale security bug report dataset construction method that can be used to create high-quality datasets of security bug reports, which are needed for prediction models described in the next paragraph, in an automated and inexpensive way [32]. The other paper is a presentation of LibVCS4j, a Java library that can be used to mine various version control systems and issue trackers [27], which can be beneficial if the research focus lies on analysing the source code (e.g. for code smells) or commit history and their relationship with issues.

There has been a series of papers published on the topic of security bug report prediction in post-release phases, i.e. by using the bug reports themselves and not the source code to detect security vulnerabilities. Early examples include a 2010 industrial case study on identifying security bug reports via text mining [6] and a 2014 paper by Chawla and Singh on “Automatic Bug Labeling using Semantic Information from LSI [Latent Semantic Indexing]” [3]. In 2017 FARSEC<sup>1</sup> was proposed, a framework that greatly improves the performance of text-based prediction models for security bug reports by using a special filtering and ranking system to prevent mislabelling [22]. Goseva-Popstojanova and Tyo found that supervised learning approaches to classify security issues slightly outperform unsupervised learning methods, but in general, the performance varies between datasets due to differences in the amount of security-related

---

<sup>1</sup>a framework composed of a combination of **F**iltering **A**nd **R**anking methods to reduce the mislabelling of **SEC**urity bug reports by text-based prediction models

information [7]. Another study suggests that for text-based prediction models, one should be augmenting standard security keywords with domain- or project-specific security vocabulary in order to get more accurate prediction results [17]. In 2019 the performance of the proposed FARSEC method was further improved by the addition of hyperparameter optimisation of both the control parameters of the learner and the data preprocessing methods [26]. Finally, with the publication of the article “LTRWES<sup>2</sup>: A new framework for security bug report detection” [10] in April 2020 a new approach was proposed that was proven to outperform FARSEC and therefore superseded FARSEC as the state-of-the-art method for text-based security bug report prediction.

Of course, GitHub is not the only interesting data source for security vulnerability research. Other studies have leveraged other websites and social media for research in this area. A large-scale study of Stack Overflow posts has been carried out where Yang et al. used Latent Dirichlet Allocation to identify topics in those posts, analyse them and draw implications for researchers, educators and practitioners [33]. Another study makes comparisons between mentioning of security vulnerabilities on two social media conversation channels (Reddit and Twitter) and a collaborative software development platform (GitHub). The researchers identified characteristics of the three platforms and were able to show that Reddit and Twitter can be used to accurately predict activity on GitHub [9].

Finally, we are left with a group of publications that are most related to this thesis. In 2014 Pletea et al. already analysed security discussions on GitHub with the specific intent to extract emotions from the comments. They found that more negative emotions are expressed in security-related discussions than in other discussions, which confirms the anecdotal evidence that implementing application security can often lead to frustration and anger among developers [23]. The next study focused on perceived language complexity in security discussions on GitHub and how individuals adhere to project-specific or overall GitHub language. They found that lack of conformity to the project-specific language norm increases issue resolution times by a small amount [12]. For an extensive article Morrison et al. did an empirical analysis of three Open Source projects to study process-related differences between vulnerabilities and defects. It turned out that vulnerabilities are found later in the development cycle and are resolved more quickly than defects; however, the results indicate that “opportunities may exist for more efficient vulnerability detection and resolution” [18]. Another linguistic analysis was carried out by Meyers et al. who analysed five linguistic metrics (formality, informativeness, implicature, politeness and uncertainty) in security conversations in the Chromium project and found that they only play a small role [16]. Finally, there was an empirical study that tried to understand the key themes and topics of security issues by using topic modelling techniques and qualitative analysis in 2018 [36].

---

<sup>2</sup>Learning To Rank with Word Embedding for Security bug report prediction

# 3

## Background

The goal of this chapter is to briefly explain the main concepts of issue tracking on GitHub and clarify the terms that are used throughout this thesis.

Issues are the bug tracking functionality of GitHub. They have their own section in every repository that has issue tracking enabled. A typical overview of the issue section in a repository is shown in figure 3.1. The overview displays a chronological list of open issues in a repository and shows not only the title of the issue but also the creation date, the author, labels and further information such as the number of comments, linked pull requests or any assignees. Those concepts are explained in more detail in the following paragraphs.

A typical issue on GitHub is shown in figure 3.2. The top section contains the title of the issue, the consecutive number of the issue in the repository indicated by the “#” prefix. Below the title, we see the state of the issue, which can be either open or closed, and next to it the username of the issue author (reporter) together with the date of creation and the current number of comments. In the right section of the issue a sidebar is located, where assignees, labels, projects, milestones, linked pull requests, notification settings and participants for the issue are displayed. A more populated issue sidebar is displayed in detail in figure 3.3.

An assignee is a person who is responsible for the issue. One or multiple users can be assigned to an issue. They receive a notification in GitHub when they are assigned to a new issue. Labels are a way to organise issues in a project. The labels are predefined categories for a project that can be applied to issues and pull requests. An issue can be labelled with one or multiple labels. For this thesis, we were particularly interested in issues that have a security label, like the example in figure 3.4. Milestones are another

The screenshot displays a GitHub issues overview page. At the top, there are filters for '2,482 Open' and '21,491 Closed' issues, along with dropdown menus for 'Author', 'Label', 'Projects', 'Milestones', 'Assignee', and 'Sort'. Below this, a list of issues is shown, each with a title, labels, and metadata. The issues are:

- Issue 1:** "Document the limitations of using mlock with system swapping still possible" (enhancement, needs:triage). #58092, opened 1 hour ago by robin13.
- Issue 2:** "Avoid saturating GET thread pool with API key hash verification" (enhancement, Security/Authentication, Team:Security). #58088, opened 9 hours ago by ywangd. Includes 1 assignee and 1 comment.
- Issue 3:** "Use race-neutral terminology in Elasticsearch code" (enhancement, needs:triage). #58087, opened 9 hours ago by nyurik.
- Issue 4:** "SearchSourceBuilder not functional on Android" (bug, needs:triage). #58075, opened 2 days ago by jhudis.
- Issue 5:** "Wildcard field should return value as Strings not ByteRefs for scripts" (Search/Search, bug, Team:Search). #58044, opened 3 days ago by markharwood. Includes 1 assignee and 1 comment.
- Issue 6:** "Add an API to easily report why shard relocation isn't happening" (enhancement, Distributed/Allocation, Team:Distributed). #58030, opened 3 days ago by sebg1. Includes 2 comments.
- Issue 7:** "How does es realize the integration of SQL nested group aggregation?". #58022, opened 3 days ago by fl1312661385.
- Issue 8:** "Improve peer recovery performance in active indexing" (Distributed/Recovery, enhancement, Team:Distributed, team-discuss). #58011, opened 4 days ago by dnhatn. Includes 1 comment.

Figure 3.1: Typical GitHub issues overview

option on GitHub to group issues that correspond to a project, feature, or time period. Below the milestones, we see a list of linked pull requests for every issue. Those are pull requests that are associated with the issue and have been linked to the issue by users who used GitHub's referencing capabilities in either the pull request or the issue discussion. Next in the sidebar, we find the notifications settings. GitHub users can subscribe to issues they are interested in, to receive notifications of future updates. They are automatically notified if they are an assignee of the issue or if they already left a comment on the issue. Finally, we see a list of avatars that shows the participants in the issue. This list includes every user who performed any action with the issue, be it creating, labelling, commenting, assigning or closing the issue.

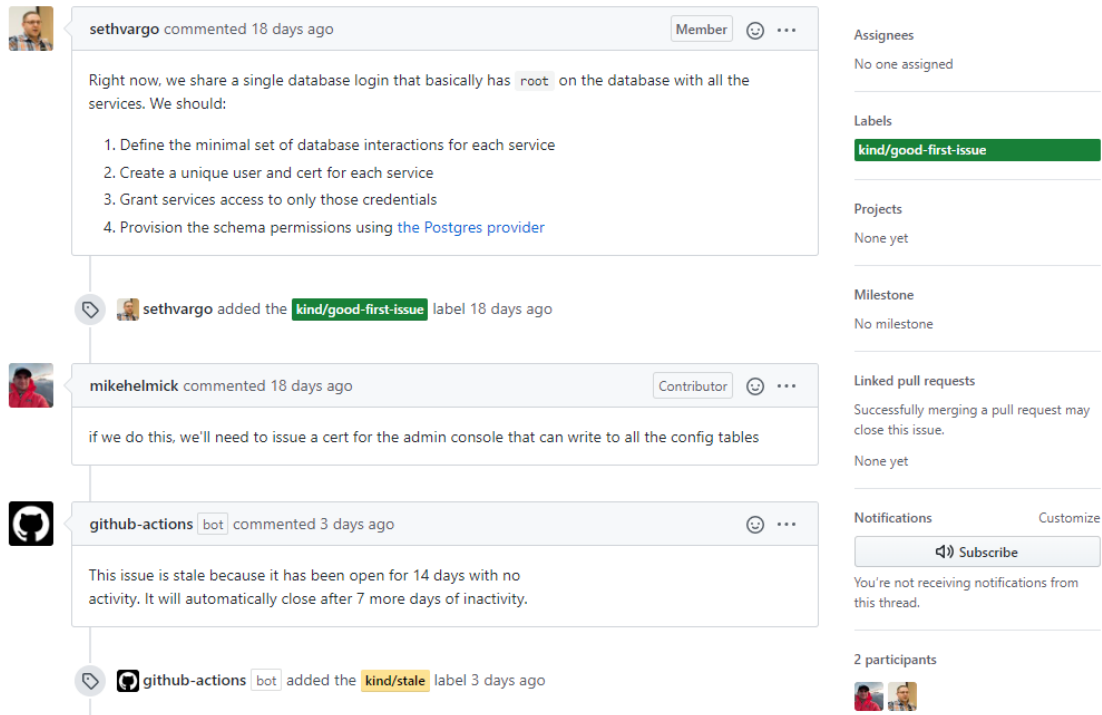
In the bottom right of figure 3.2, we see the main section of every GitHub issue, the issue timeline. The issue timeline is a log of everything that happens with a GitHub issue. Not only does the timeline show the initial issue report and all the comments by the community, but also the addition of new labels, any assignments, any references elsewhere on GitHub and of course if and how an issue gets closed. Examples of those additional timeline entries can be seen in figure 3.5.

Now that we saw all basic elements of a GitHub issue, we also take a closer at the comments that user can make on the issues. A typical GitHub issue comment is shown

## Move to per-service database credentials and POLP #471

New issue

 Open sethvargo opened this issue 18 days ago · 2 comments



The screenshot shows a GitHub issue page for "Move to per-service database credentials and POLP #471". The issue is marked as "Open" and was opened by "sethvargo" 18 days ago. It has 2 comments. The issue is labeled "kind/good-first-issue" and "kind/stale".

**Comments:**

- sethvargo** (Member) commented 18 days ago: "Right now, we share a single database login that basically has `root` on the database with all the services. We should:
  1. Define the minimal set of database interactions for each service
  2. Create a unique user and cert for each service
  3. Grant services access to only those credentials
  4. Provision the schema permissions using [the Postgres provider](#)
- sethvargo** added the `kind/good-first-issue` label 18 days ago
- mikehelmick** (Contributor) commented 18 days ago: "if we do this, we'll need to issue a cert for the admin console that can write to all the config tables"
- github-actions** (bot) commented 3 days ago: "This issue is stale because it has been open for 14 days with no activity. It will automatically close after 7 more days of inactivity."
- github-actions** (bot) added the `kind/stale` label 3 days ago

**Metadata:**

- Assignees:** No one assigned
- Labels:** `kind/good-first-issue`
- Projects:** None yet
- Milestone:** No milestone
- Linked pull requests:** Successfully merging a pull request may close this issue. None yet
- Notifications:** [Subscribe](#). You're not receiving notifications from this thread.
- Participants:** 2 participants

Figure 3.2: Typical GitHub issue

in figure 3.6. On the top of every comment, we see the author and the creation date of the comment. Furthermore, we are informed of the author's association with the project and can react to an issue using a single emoji. The author's association with a project can be one of the following categories:

- **Owner / Member:** The owner is the GitHub user who created the repository. If the repository was created and is maintained by an organisation, then all developers who belong to that organisation fall into the member category.
- **Collaborator:** Collaborators are users who were invited by the project owners to contribute to a project and have write access to the repository.
- **Contributor:** Contributors are users who do not have collaborator access to a repository but have contributed to a project and had a pull request they opened merged into the repository.
- **None:** All other users who do not belong to one of the first three categories fall

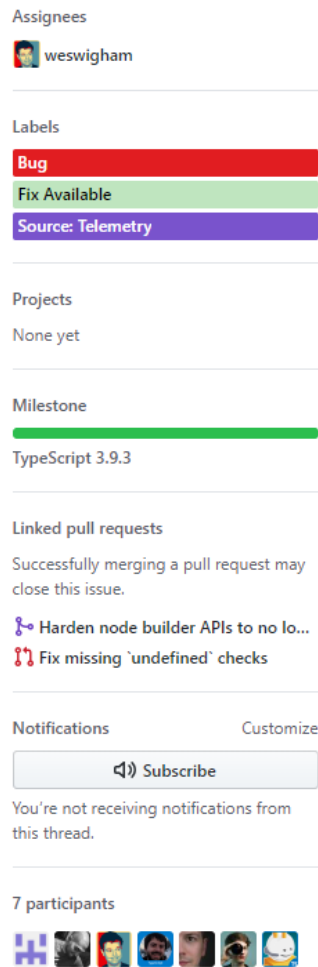


Figure 3.3: Issue sidebar

into this category, in which case no indication of their association is made in the top section of the comment.

As is visible in figure 3.7, an additional author label is displayed next to the association if the author of the comment is the initial author of the issue.

In order to style and format their comments, developers can make use of the GitHub Flavored Markdown syntax, which is a dialect of the “Markdown” markup language. It allows users to highlight text, include hyperlinks or include source code snippets (see figure 3.2). Most importantly though, it allows users to make references to other commits, issues, pull requests or repositories within the GitHub ecosystem. Additionally, users can attract the attention of other users by mentioning them using @-mentions. An example of this feature in use is shown at the very beginning of the comment in figure 3.7. When



Figure 3.4: Security label on an issue

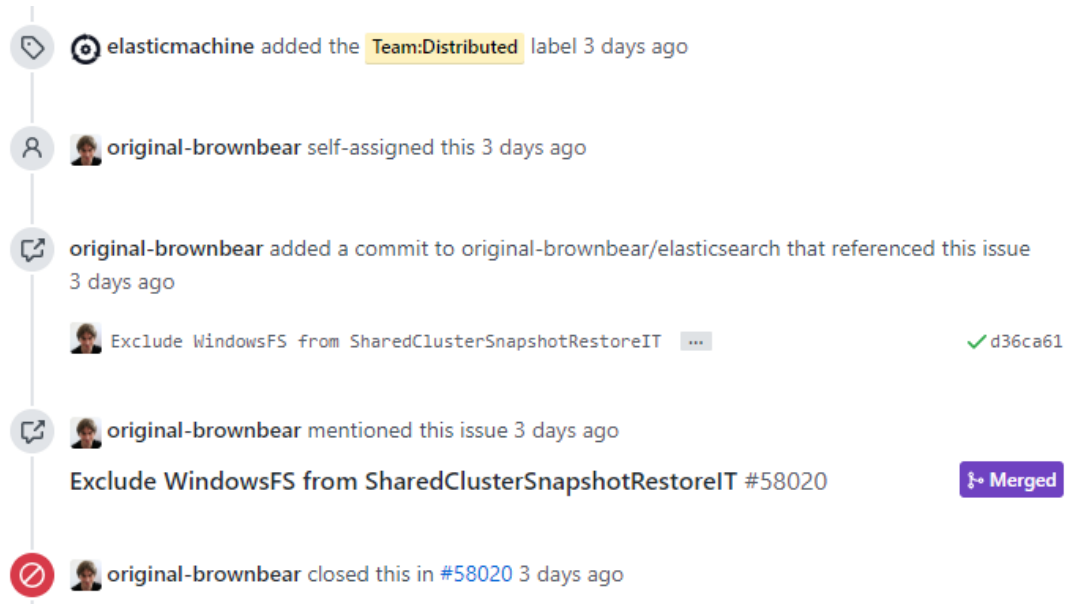


Figure 3.5: Additional issue timeline entries

users get mentioned this way, they receive a personal notification.

There are three ways to close an issue on GitHub. The author of an issue or entitled users with write access can close an issue manually. This mostly happens when no code changes have been implemented, and the issue is rejected. The second way is to automatically close an issue with a direct commit to the repository. If the commit message of a commit contains certain keywords like “closes” followed by “#” and the number of an issue, then the issue is automatically closed, and the commit gets linked to the issue. An example is shown in the bottom half of figure 3.8. The final and most common way to close an issue is to create a pull request that fixes the problem and link it to the issue as just described. The pull request then also gets linked to the issue as shown in the top half of figure 3.8 and the issue is automatically closed when the pull request gets merged.

Pull requests are a common concept in modern software engineering and part of many open source code hosting platforms. We will only explain a particular set of features in GitHub pull requests here that were relevant for our study. As we already saw, pull requests are closely related and integrated with issues on GitHub. It is therefore

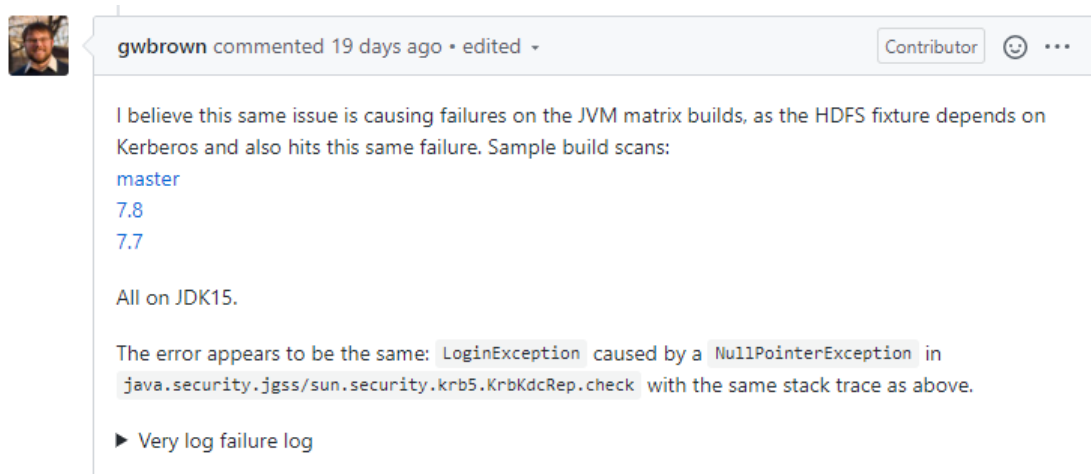


Figure 3.6: Typical GitHub issue comment

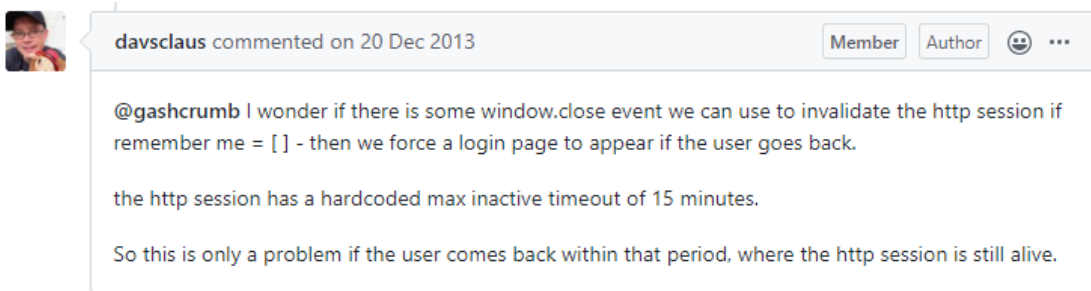


Figure 3.7: Mentioning in an issue comment

unsurprising that a typical pull request, shown in figure 3.9, looks very similar to an issue. In addition to issues, pull requests also contain information about commits and changed files that belong to them. Most importantly though, we can see at the top right a list of reviewers who have been added to the pull request. Feedback from one or multiple users can be requested by adding them as a reviewer to a pull request. They then receive a notification and have the ability to review the proposed code changes. They can leave comments in the discussion or at specific lines in the source code and either approve the pull request or request changes to the proposal. Their verdict is indicated by the icon next to their username under “Reviewers”.

Generally, any GitHub user who has read access to a repository can submit an issue. For public open source projects, this means every GitHub user can create an issue. In order to do so, one has to fill out a simple form shown in figure 3.10. The form only consists of the title and description field. For the description, the reporter can make use of the Markdown syntax mentioned above to style and format the issue report or



include source code snippets, mention other developers, reference existing issues and pull requests and more. A project may provide issue templates that have a predefined structure (i.e. a section for reproducibility information) that the issue reporter should follow because it is desired by the community. The author may also attach files to an issue. Labelling and assigning of issues only happens after the creation of an issue and can only be performed by GitHub users with the status owner, member or collaborator for the project, i.e. developers with write access to the repository.

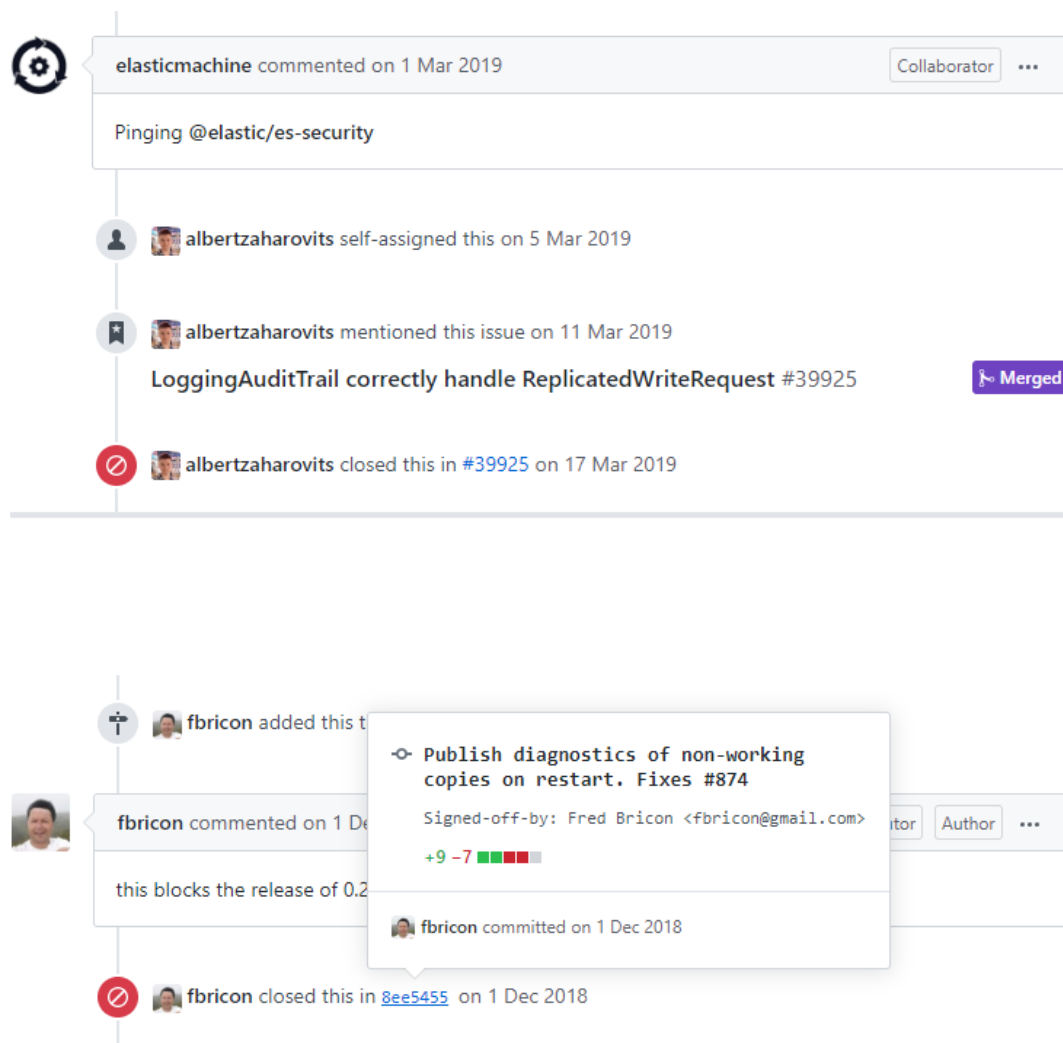


Figure 3.8: GitHub issues closed by linked pull request (top) and direct commit (bottom)

## [DOCS] Make ILM documentation data stream aware #58035

Open andreidan wants to merge 36 commits into elastic:master from andreidan:docs-ilm-data-stream

Conversation 54 Commits 36 Checks 0 Files changed 6 +291 -79

andreidan commented 3 days ago • edited

Relates to #53100  
Relates to #57905

Reviewer: dakrone ✓, jrodewig ✓, martijnvg ●

Assignees: No one assigned

Labels: :Core/Features/Data streams, :Core/Features/ILM+SLM, >docs, Team:Core/Features, Team:Docs, v7.9.0, v8.0.0

Projects: None yet

Milestone: No milestone

Linked issues: Successfully merging this pull request may close these issues. None yet

4 participants

andreidan added 2 commits 4 days ago

- [DOCS] Make ILM documentation data stream aware 2e46761
- [DOCS] ILM: Add managing indices using alias to tutorial ● fa37a17

andreidan added >docs, :Core/Features/ILM+SLM v8.0.0, :Core/Features/Data streams v7.9.0 labels 3 days ago

andreidan requested review from dakrone, martijnvg and jrodewig 3 days ago

elasticmachine commented 3 days ago

Pinging @elastic/es-docs (> docs)

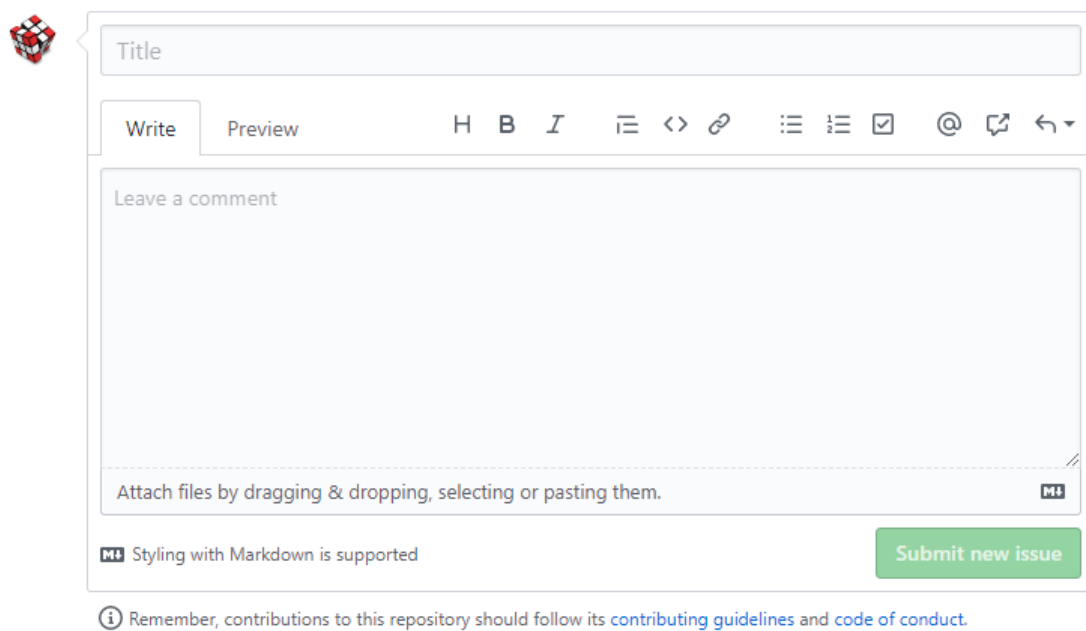
elasticmachine added the Team:Docs label 3 days ago

elasticmachine commented 3 days ago

Pinging @elastic/es-core-features (:Core/Features/ILM+SLM)

elasticmachine added the Team:Core/Features label 3 days ago

Figure 3.9: Typical GitHub pull request



The image shows a screenshot of an empty GitHub issue submission form. On the left side, there is a small red and white cube icon. The form itself is a white box with a light gray border. At the top, there is a text input field labeled "Title". Below this, there are two tabs: "Write" (which is active) and "Preview". To the right of the tabs is a rich text editor toolbar containing icons for bold (H), italic (I), list (≡), code (<>), link (🔗), checklist (☑), mention (@), reply (↩), and a dropdown arrow. Below the toolbar is a large text area with the placeholder text "Leave a comment". At the bottom of the text area, there is a dashed line and the text "Attach files by dragging & dropping, selecting or pasting them." with a small "M" icon. Below the text area, there is a message that says "Styling with Markdown is supported" with a small "M" icon. To the right of this message is a green button labeled "Submit new issue". At the very bottom of the form, there is a small information icon (i) followed by the text "Remember, contributions to this repository should follow its [contributing guidelines](#) and [code of conduct](#)."

Figure 3.10: Empty issue submission form

# 4

## Methodology

In this chapter we describe the methodology that we followed throughout our study from the selection of the dataset to the analysis of the data. In order to answer the questions outlined in the introduction, we decided to do our analysis in two parts. First, we selected a large number of GitHub projects according to the criteria outlined in section 4.1. After classifying all issues of the selected projects into security issues and non-security issues as described in section 4.2, we ended up with a dataset of projects, corresponding issues and their discussion comments that we could use to perform the large-scale analysis outlined in section 4.3. Because we were particularly interested in qualitative features of the issue discussions and pull requests, we also selected a significant sample of issues from the large-scale dataset in order to perform a manual, more qualitative analysis by hand as described in section 4.5. The sampling procedure is described in section 4.4.

### 4.1 Selection of GitHub projects

We had to make a selection of GitHub projects (repositories) for three reasons. Firstly, the dataset would have been too big to be practically handled if we wanted to analyse all GitHub projects. Secondly, not all GitHub projects are so-called “engineered software projects” [20] that we are interested in. Finally, not all “engineered software projects” were suitable for this analysis, because not all of them have issue tracking enabled and not all of them use English as their issue tracking language. Therefore we set up the following selection criteria on projects and incorporated some of the peril avoidance strategies proposed by Kalliamvakou et al. [11] in their analysis of the promises and

perils of mining GitHub:

1. Primary programming language is Java
2. Repository is not a fork of another repository
3. More than 10 forks
4. More than 10 stars
5. More than 2 kB in size
6. Pushed to at least once after 01.01.2019
7. More than 10 issues
8. More than 50 commits
9. Issue tracking language is English

Criterion (1) was mainly introduced to drastically reduce the scope of the search for projects, while still adhering to a fairly popular programming language. Criteria (1) - (8) could be automatically incorporated into the data collection process. This process resulted in 6 104 available repositories on GitHub. After semi-automatically applying criterion (9) we were left with 5 572 repositories. With those 5 572 repositories we proceeded to the next step of classifying the issues as described in section 4.2 and then later removing the projects from the dataset that turned out not to include any issues that were classified as security issues.

## 4.2 Classification of security issues

For the classification into security issues and non-security issues, we used a simple label-based approach. For all the 5 572 remaining repositories from the previous step we extracted any labels that contained the string “security”. We found 276 repositories that had at least one such label available, and in 182 instances such a label was actually used to tag at least a single issue in the project. We proceeded with those 182 projects because we only wanted to include projects in our analysis that had both security and non-security issues. We completed our dataset by downloading all the issues of those 182 GitHub projects and classifying those issues that were labelled with one of the aforementioned security labels as security issues and all other issues as non-security issues. In the rest of this thesis whenever we refer to security or non-security issues, we mean the classification described in this section.

We ended up with our final dataset of 182 GitHub projects containing a total of 249 043 issues, of which 3 493 were labelled as security issues, and a total of 860 948

Metric	commits	forks	stars	issues	pull requests	age [years]
Mean (Std. Error)	6 135 (679)	629 (123)	1 751 (335)	1 368 (178)	1 416 (222)	5.20 (0.18)
Std. Deviation	9 167	1 657	4 524	2 397	2 999	2.42
Minimum	79	11	12	12	0	0.49
25th percentile	931	41	75	229	122	3.51
Median	2 530	141	339	624	436	5.02
75th percentile	8 025	539	1 501	1 455	1 357	6.86
Maximum	68 171	16 270	47 805	23 100	1 357	11.75

Table 4.1: General statistics of the 182 repositories in the dataset

discussion comments. Key statistics of the 182 GitHub projects are summarised in table 4.1.

### 4.3 Large-scale analysis

For the large-scale analysis we used the full dataset established during the procedure described in section 4.2. Our main source of data was the GitHub GraphQL API<sup>1</sup>. To determine the gender of GitHub users based on their names, we used the services of Genderize.io<sup>2</sup>. The accuracy of Genderize.io has been evaluated in research [25]. We wanted to automatically download, calculate and derive as many features as possible in the repository, issue and comment dimension on the full dataset so that we would get a comprehensive overview of security issues and can also compare them to non-security issues. Our downloaded data represents a snapshot at the 11.04.2020 14:00 UTC.

#### 4.3.1 Features in issue dimension

We downloaded, calculated or derived the features prefixed by the letter *I* and listed in the middle section of table 4.2 for all 249 043 issues automatically. In the following paragraphs additional explanations to selected features in the issue dimension are provided.

**I.authorName:** GitHub users can choose to disclose their full name to the public. In the first case, the real name of the author was used; otherwise the GitHub username of the author was used.

**I.authorGender:** We populated the first word in *I.authorName* to the Genderize.io API and used the result. In case the GitHub user chose an invented username or had a gender-neutral first name the API could not provide a result, and the value “unknown” was used.

<sup>1</sup><https://developer.github.com/v4/>

<sup>2</sup><https://genderize.io/>

Feature name	Description	Data type	Possible values	Data source
R_name	The name of a repository.	String		GitHub API
R_createdAt	The exact date-time when a repository was created.	Datetime		GitHub API
R_age	The time interval for which a repository already has been existing.	Duration		GitHub API
R_commits	The number of commits of a repository.	Integer		GitHub API
R_releases	The number of releases of a repository.	Integer		GitHub API
R_forks	The number of forks of a repository.	Integer		GitHub API
R_stars	The number of stars of a repository.	Integer		GitHub API
R_pullRequests	The number of pull requests created for a repository.	Integer		GitHub API
R_issues	The number of issues created for a repository.	Integer		GitHub API
R_securityIssues	The number of security issues for a repository.	Integer		Calculation
R_ratioSecurityIssues	The quotient of R_securityIssues divided by R_issues.	Decimal		Calculation
R_timeUntilFirstIssue	The time interval between the creation of a repository and the creation of its first issue.	Duration		Calculation
R_timeUntilFirstSecurityIssue	The time interval between the creation of a repository and the creation of its first security issue.	Duration		Calculation
R_meanIssueInterval	The mean interval between the creation of the issues of a repository.	Duration		Calculation
R_meanSecurityIssueInterval	The mean number of days between the creation of the security issues of a repository.	Duration		Calculation
I_title	The title of an issue.	String		GitHub API
I_url	The GitHub URL of an issue.	String		GitHub API
I_createdAt	The exact date-time when an issue was created.	Datetime		GitHub API
I_authorName	The name of the author of an issue.	String		GitHub API
I_authorAssociation	The association of the author of an issue with the repository.	Enum	member, collaborator, contributor, none	GitHub API
I_authorGender	The gender of the author of an issue.	Enum	male, female, unknown	Genderize.io
I_state	The state of an issue.	Enum	open, closed	GitHub API
I_closedAt	The exact date-time when an issue was closed.	Datetime		GitHub API
I_numberOfComments	The number of comments in the discussion of an issue.	Integer		GitHub API
I_numberOfAssignees	The number of users who were assigned to an issue.	Integer		GitHub API
I_numberOfParticipants	The number of distinct participants in the discussion of an issue.	Integer		Calculation
I_assignmentActorName	The name of the user who performed the first assignment of an issue.	String		GitHub API
I_firstAssigneeName	The name of the first assignee of an issue.	String		GitHub API
I_timeUntilFirstComment	The time interval between the creation of an issue and the creation of the first comment.	Duration		Calculation
I_meanCommentInterval	The mean time interval between the creation the comments of an issue.	Duration		Calculation
I_timeUntilFirstAssignment	The time interval between the creation of an issue and the first assignment.	Duration		Calculation
I_timeUntilClosed	The time interval between the creation and closure of an issue.	Duration		Calculation
I_age	The time interval for which an issue has been existing.	Duration		Calculation
I_labeledAsSecurityIssue	If an issue has a label containing the word security.	Boolean	true, false	Calculation
I_status	If an issue is pending or has been accepted or rejected.	Enum	accepted, pending, rejected	Calculation
C_createdAt	The exact date-time when a comment was created	Datetime		GitHub API
C_authorName	The name of the author of a comment.	String		GitHub API
C_authorAssociation	The association of the author of a comment with the repository.	Enum	member, collaborator, contributor, none	GitHub API
C_authorGender	The gender of the author of a comment.	Enum	male, female, unknown	Genderize.io

Table 4.2: Automatically extracted features

**I\_numberOfParticipants:** This feature should not be confused with the number of participants in an issue that is displayed on every GitHub issue. Our feature only includes distinct people who left a comment on the issue, while GitHub’s number also includes people who merely performed an action on an issue (e.g. adding a label or closing the issue), even when they did not leave a comment. This is why we had to calculate this feature using the comments we downloaded.

**I\_timeUntilFirstComment:** This feature is often referred to as the reaction time of an issue.

**I\_age:** This feature is the same as *I\_timeUntilClosed* for closed issues but also expresses how long open issues have been pending until the moment of the analysis.

**I\_labeledAsSecurityIssue:** The details of this feature are explained in section 4.2.

**I\_status:** We introduced this feature to make comparisons between issues with different resulting outcomes. We derived the values “accepted”, “pending” and “rejected” as follows: Issues with *I\_state* “open” are classified as “pending”. Issues with *I\_state* “closed” are classified as “accepted”, if a change of code relating to this issue has been incorporated into the project via a linked and merged pull request or a linked direct commit to the repository, as described in chapter 3. Issues with *I\_state* “closed” that do not meet the criteria just mentioned are classified as “rejected”.

### 4.3.2 Features in repository dimension

We downloaded, calculated or derived the features prefixed by the letter *R* and listed in the top section of table 4.2 for all 182 repositories automatically. In the following paragraph additional explanations to a selected feature in the repository dimension are provided.

**R\_meanSecurityIssueInterval:** If a repository only included a single security issue, no value for this feature was assigned to it.

### 4.3.3 Features in comment dimension

We downloaded, calculated or derived the features prefixed by the letter *C* and listed in the bottom section of table 4.2 for all 852 341 comments automatically. In the following paragraphs additional explanations to selected features in the comment dimension are provided.

**C\_authorName:** GitHub users can choose to disclose their full name to the public. In the first case, the real name of the author was used; otherwise the GitHub username of the author was used.

**C\_authorGender:** We populated the first word in *C\_authorName* to the Genderize.io API and used the result. In case the GitHub user used an invented username or had a gender-neutral first name the API could not provide a result, and the value “unknown” was used.



## 4.4 Sampling procedure for manual analysis

Because doing a detailed qualitative analysis of all the security issues from the full dataset would have exceeded our time and resource constraints we decided to draw a significant sample from the full dataset and then perform our manual analysis with that sample. Because we were particularly interested in the discussions of the security issues and we wanted to focus our resources, we decided to exclude all issues with zero comments, where a meaningful analysis of the discussion was not possible anyway. With those restrictions our population of 3 493 security issues already shrank to 2 354 commented security issues. We analysed the two-dimensional distribution of this population along the features *I\_status* (accepted, pending, rejected) and *I\_numberOfComments* (separate quartiles per status group), because we wanted to perform a stratified-random sampling [1] in order to preserve the distribution of two features and get a broad insight into different security issue discussions, while still maintaining sufficient representativeness to draw conclusions for the population of commented security issues. In a bigger and more widespread dataset, we would not have to worry about the *I\_numberOfComments*-quartiles because quartiles are by nature already equally distributed, however, the distribution of the number of comments was very squeezed, which caused some quartiles to contain more entries than others. The composition of the population of 2 354 commented security issues can be seen in table 4.3. We determined we want to achieve a confidence interval of  $\pm 5\%$  at a confidence level of 95%. We then calculated the required sample size with those parameters using the formula proposed by Krejcie and Morgan [14]:

$$s = \frac{X^2 NP(1 - P)}{d^2(N - 1) + X^2 P(1 - P)}$$

$s$  = required sample size.

$X^2$  = the table value of chi-square for 1 degree of freedom at the desired confidence level (3.841).

$N$  = the population size (2 354).

$P$  = the population proportion (assumed to be .50 since this would provide the maximum sample size).

$d$  = the degree of accuracy expressed as a proportion (.05).

With our parameters this resulted in a minimum required sample size of 331. We distributed the samples to the strata according to table 4.3 and ended up using a sample of 333 security issues in order to accommodate for rounding imprecisions. The composition of this sample can be seen in table 4.4. Finally, we used a true random number generator<sup>3</sup>

<sup>3</sup><https://www.random.org>

	<b>Accepted</b>	<b>Pending</b>	<b>Rejected</b>	<b>Total</b>
<b>1</b>	321 (13.6%)	162 (6.9%)	267 (11.3%)	750
<b>2</b>	357 (15.2%)	120 (5.1%)	138 (5.9%)	615
<b>3</b>	165 (7.0%)	142 (6.0%)	122 (5.2%)	429
<b>4</b>	247 (10.5%)	139 (5.9%)	174 (7.4%)	560
<b>Total</b>	1 090	563	701	2 354

Table 4.3: Population of 2 354 commented security issues (issue status vs. number-of-comments-quartile)

	<b>Accepted</b>	<b>Pending</b>	<b>Rejected</b>	<b>Total</b>
<b>1</b>	45	23	38	106
<b>2</b>	50	17	19	86
<b>3</b>	23	20	17	60
<b>4</b>	36	20	25	81
<b>Total</b>	154	80	99	333

Table 4.4: Sample of 333 commented security issues (issue status vs. number-of-comments-quartile)

to randomly select the number of issues as specified in table 4.4 from our different strata of the population.

## 4.5 Manual analysis

For the manual analysis we used the sample of 333 security issues established during the procedure described in section 4.4. We qualitatively analysed the issue reports, the comments in the issue discussions and any related pull request discussions. The features we manually collected are listed in table 4.5. In order to test our questionnaire, refine the questions we wanted to answer and mitigate subjective influence, we first conducted a pilot study with 17 randomly selected security issues. Those issues for the pilot study were drawn from the remaining 2 021 commented security issues that were not used for the actual study. The results of the pilot study were compared and discussed by the author and the supervisor of this thesis until consensus was reached.

### 4.5.1 Features in issue dimension

In addition to the features already present from the large-scale analysis, we answered the questions for features listed in table 4.5 and prefixed by the letter *I* for the 333 selected issues. In the following paragraphs additional explanations to selected features in the

Feature name	Description	Data type	Possible values
I_clear	Is the issue clearly explained?	Boolean	true, false
I_info	What information does the issue provide or refer to via hyperlinks?	Enum	none, fix, documentation, cve, reproducibility
I_pullRequest	Is there a pull request and if yes is it reviewed?	Enum	no, no review, single review, multiple reviews
I_roleOfAuthorInPR	Is the author of an issue involved in the pull request?	Enum	no, creator, reviewer
I_numberOfCommentsInPR	The number of comments in the discussion of an associated pull request to the issue.	Integer	
I_numberOfParticipantsInPR	The number of distinct participants in the discussion of an associated pull request to an issue.	Integer	
I_prDiscussionChallenge	How challenging is the discussion of an associated pull request to an issue?	Enum	low, medium, high
C_mention	Is the comment referring to another person?	Enum	no, author of issue, member, collaborator, contributor, other
C_confident	Is the comment formulated confidently?	Enum	true, false, not applicable
C_info	What information does the comment provide or refer to via hyperlinks?	Enum	none, fix, documentation, cve, reproducibility
C_nature	What is the nature of the comment?	Enum	elaboration, ask/request, reminder, notification, instruction, other
C_keyPoint	Is the comment a key point in the discussion?	Boolean	true, false

Table 4.5: Manually extracted features

issue dimension are provided.

**I.clear:** Based on our personal impression and the reaction of the community in the follow-up discussion, we judged if the author of an issue clearly explained the issue in his or her report. Clear issue reports are reports that contain enough detail and are written in such a way that the problem or unexpected behaviour can be understood. They can, but do not have to, be structured or contain source code snippets. Figure 4.1 shows an example of a clearly described issue that even includes reproducibility information. Figure 4.2, however, shows an example of an unclear issue. The description only contains an error message and does not provide any further context.

**I.info:** We categorised resources that were provided either directly in the issue or via a hyperlink into the categories “fix”, “documentation”, “cve” and “reproducibility”. A “fix” is generally a concrete suggested change to the source code that is likely to resolve the issue. “Documentation” refers to further information about the functionality of software or a third-party library. “CVE” included all kinds of entries in vulnerability databases. We used the value “reproducibility” if the report included any kind of resources that described how to reproduce an issue. Finally, we assigned the value “none” if none of the information mentioned before was provided.

**I.pullRequest:** We checked if a pull request was directly linked to an issue. If no pull request could be found we assigned the value “no”. If a pull request existed, we checked whether or not (and how many times) the review functionality on GitHub was used on the pull request corresponding to the issue. If no review had been performed, we assigned the value “no review”. If one review had been conducted, we assigned the value “single review”, and if more than one review had been done, we assigned the value “multiple reviews”.

**I.roleOfAuthorInPR:** We differentiated if the author of the issue also created the pull request, was assigned as a reviewer of the pull request or was not involved at all in the pull request. These roles are exclusive, as the creator of a pull request should not review his or her own pull request.

**I.numberOfParticipantsInPR:** This feature should not be confused with the number of participants in a pull request that is displayed on every GitHub pull request. Our feature only includes distinct people who left a comment on the pull request, while GitHub’s number also includes people who merely performed an action on a pull request (e.g. adding a label or merging the pull request), even when they did not leave a comment.

**I.prDiscussionChallenge:** This is a personal impression on how challenging the discussion on a pull request was. We did take into consideration how long it took to reach the final merge state or the decision to merge or abandon the pull request and how strong the participants disagreed or argued. We assigned a value “low” if there was basically no disagreement between the participants of the discussion, a value “medium” if there was a moderate discussion going on and a value “high” if it was very challenging to come to a conclusion or the pull request was highly controversial.

### Distinguished Name Predicate throws NPE for null user DN #41305 New issue

**Closed** jkakavas opened this issue on 17 Apr 2019 · 1 comment

**jkakavas** commented on 17 Apr 2019 · edited ▾ Contributor 🗨️ ⋮

The implementation of `DistinguishedNamePredicate` (<https://github.com/elastic/elasticsearch/blob/master/x-pack/plugin/security/src/main/java/org/elasticsearch/xpack/security/authc/support/UserRoleMapper.java#L161>) doesn't take into consideration that the user might have a `null` DN (user property) when a role mapping that uses a wildcard or regexp is in use and would possibly apply on that user. When evaluating the rule, this causes an NPE in `org.apache.lucene.util.automaton.CharacterRunAutomaton`.

This can be reproduced with a role mapping such as

```
{
  "enabled" : true,
  "roles" : [
    "kibana_user"
  ],
  "rules" : {
    "field" : {
      "dn" : "*,ou=finance,dc=somethign,dc=com"
    }
  },
  "metadata" : { }
}
```

and a i.e. SAML user in a realm that has no `attributes.dn` mapping rule in the configuration. The result is an uncaught NPE that causes the authentication to fail

**Assignees**

albertzaharovits

---

**Labels**

Security/Authentication

bug

---

**Projects**

None yet

---

**Milestone**

No milestone

---

**Linked pull requests**

Successfully merging a pull request may close this issue.

[Fix role mapping DN field wildcards...](#)

---

**Notifications** Customize

You're not receiving notifications from this thread.

Figure 4.1: Clear issue report

### The instruction for running integration tests from integration-tests/README.md doesn't work #7842 New issue

**Closed** leventov opened this issue on 6 Jun 2019 · 7 comments

**leventov** commented on 6 Jun 2019 Member 🗨️ ⋮

```
[ERROR] 1) Error in custom provider, java.lang.RuntimeException: java.io.FileNotFoundException:
```

FYI @jon-wei

**Assignees**

No one assigned

---

**Labels**

Area - Testing

Development Blocker

Security

Figure 4.2: Unclear issue report

During the manual analysis we also took notes on special or interesting findings on security issues that we made during the course of the study. Some interesting results are described in section 5.11.

## 4.5.2 Features in comment dimension

In addition to the features already present from the large-scale analysis we answered the questions for the features listed in table 4.5 and prefixed by the letter *C* for the 1 335 comments that were posted in the discussions of the sampled 333 security issues. In the following paragraphs additional explanations to selected features in the comment dimension are provided.

**C\_mention:** We checked if any developer was mentioned in the comment using the @-mention functionality of GitHub as described in the background chapter and categorised the mentions into “author of issue”, “member”, “collaborator”, “contributor” or “other” according to their association with the issue and the repository the issue belongs to.

**C\_confident:** Based on the style, formulation and use of words, we tried to judge if the person who commented was confident about his or her opinion and the issue itself. In some cases, this could not be judged or was irrelevant in which case we assigned the value “not applicable”. Figure 4.3 shows an example of all three values. The first comment is clearly confident, and the author knows what he is talking about. The middle comment is rather unconfident. The use of words like “seems” or “maybe” indicates the uncertainty of the author of this comment. The final comment at the bottom of figure 4.3 is simply a notification that a pull request was created and the determination is irrelevant.

**C\_info:** We categorised resources that were provided either directly in the comment or via a hyperlink into the categories “fix”, “documentation”, “cve” and “reproducibility”. A “fix” is generally a concrete suggested change to the source code that is likely to resolve the issue. “Documentation” refers to further information about the functionality of software or a third-party library. “CVE” included all kinds of entries in vulnerability databases. We used the value “reproducibility” if the comment included any kind of resources that described how to reproduce an issue. Finally, we assigned the value “none” if none of the information mentioned before was provided.

**C\_nature:** This is one of the most important features of our manual analysis. We assigned each comment one of the categories “elaboration”, “ask/request”, “reminder”, “notification”, “instruction” or “other” based on the nature of the comment. The value “elaboration” was used whenever there was an addition of more information or an explanation of the issue or any facts related to it. We assigned “ask/request” if someone asked for further information about the issue or requested clarification by the author of the issue. If someone asked if there are any updates on an issue or mentioned certain developers to remind them of an issue, we used the value “reminder”. For notifications

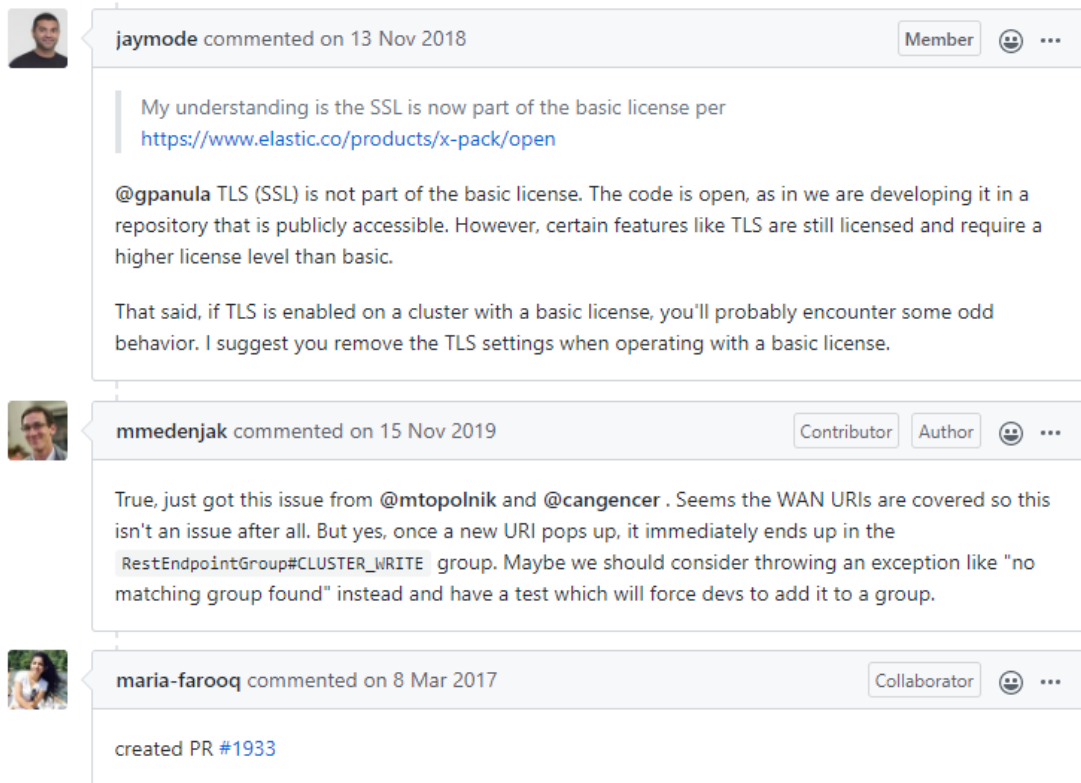



Figure 4.3: Confident vs. unconfident vs. not applicable comments (from top to bottom)


of any kind, e.g. the notification that a pull request was created or that the issue will be closed or reopened, we used the value “notification”. If a user instructed another user or a group of developers to perform certain actions, we assigned the value “instruction”. Finally, if none of the mentioned characteristics applied to a comment, we used the value “other”, e.g. when someone simply thanked a contributor for fixing an issue. A concrete example for each comment nature can be found in figure 4.4.

**C\_keyPoint:** In some issues a single or even multiple comments can be identified as a key point (or turning point or decision point) during the discussion. Where we were able to identify such a comment in an issue discussion, we used the value “yes” for this feature of the comment; otherwise we assigned the value “no”. Figure 4.5 shows a short issue discussion, where the second comment marked a key point in the discussion as the author of the comment notified the community that she will now close the issue because a new method has been added to the code. Other common key points include the identification of the root cause of a problem, the decision that a pull request can be created or the decision that an issue should be closed.

**Elaboration**



norbert-ka commented on 5 Jan 2015 😊 ...

In my opinion validation should only be done at the boundary of components. It is the sole concern of the facade layer (named in the architecture guidelines, which is probably what is called service layer in OASP). At the boundary of a component we cannot guarantee that external code is complying to the component contract and therefore have to do parameter validation. Inside of the component only component code itself is calling the lower layers. We can guarantee the correctness of parameter usage with Unit-Tests there and do not need another layer of parameter validation.

**Ask / Request**


jaymode commented on 26 Oct 2018 Member 😊 ...


where are you configuring ssl in your settings? What other settings have you configured?

**Reminder**


woshifudayun commented on 12 Oct 2019 😊 ...


@leventov Any updates? I'm getting the same error.

Thanks.

**Notification**



kristip17 commented on 3 Mar Contributor Author 😊 ...

This is resolved, going in under #10799

**Instruction**


davivel commented on 24 Oct 2016 Member 😊 ...

@croisez , your issues are different than what's described here. Please, open two new separate issues for them.

**Other**


javanna commented on 28 Dec 2018 Member Author 😊 ...

thanks @jkakavas !!

Figure 4.4: Examples of different comment natures



## Enhance LibertyServer class with additional FAT infrastructure

#2173 New issue

Closed ayoho opened this issue on 13 Feb 2018 · 1 comment

**ayoho** commented on 13 Feb 2018 Member  ...

Tracks the addition or enhancement of baseline code that can be brought in from common security FAT infrastructure.

This is intended to include, at first, very basic enhancements such as:

- Server identifying information (e.g. host name, IP address, ports)
- File management (e.g. config file manipulation)
- Server runtime actions (e.g. stop, start, restart, reconfigure the server)

 **ayoho** added Epic team:Security SSO labels on 13 Feb 2018

**ayoho** commented on 7 May 2018 Member Author  ...

A `reconfigureServer()` method has been added which is sufficient for now. I'm therefore closing this epic and additional enhancements can be added on a case-by-case basis.

 **ayoho** closed this on 7 May 2018

**Assignees**  
No one assigned

**Labels**  
Epic  
team:Security SSO

**Projects**  
None yet

**Milestone**  
No milestone

**Linked pull requests**  
Successfully merging a pull request may close this issue.  
None yet

**Notifications** Customize  
Subscribe

Figure 4.5: Key point in an issue discussion

# 5

## Results

In this chapter we present the results of our analysis that we did, using the methodology described in chapter 4 in order to answer the questions outlined in the introduction.

We thematically grouped our results into sections 5.1 to 5.8, where we look at various characteristics of security issues and also compare them to non-security issues. Furthermore, we make comparisons between the different issue statuses. In section 5.9 we look at how certain characteristics of security issues evolve over the lifespan of an issue and if certain factors have an impact on the resolution time of security issues. In the next section 5.10, we present our findings concerning the evolution of security issues over the last six years. In the final section 5.11, we present other general observations we made over the course of the study during any stage of data collection or analysis.

### **5.1 Prevalence and emerging of security issues**

Our full dataset consisted of 182 GitHub projects which included a total of 249 043 issues, of which 3 493 (1.42%) are security issues and 245 550 (98.60%) are non-security issues. When analysing the percentages of security issues in each individual project, we found the distribution displayed in table 5.1. We can see that, even though the data is quite spread out, security issues generally make up for a very small percentage of all the issues of a project. We find that 75% of all projects have a percentage that is lower than 2.23%.

Furthermore, we were interested when security issues emerge in a project. We analysed the time interval between the creation of a project and the creation of the first

<b>Metric</b>	<b>Value</b>
n	182
Mean (Std. Error)	3.92% (0.94%)
Std. Deviation	12.67%
Minimum	0.02%
25th percentile	0.29%
Median	0.91%
75th percentile	2.23%
Maximum	92.31%

Table 5.1: Percentage of security issues in individual projects

security issue and also the mean time interval between subsequent security issues in a project. We found that the median is 655.50 days (interquartile range (IQR) 1 202.25 days) until the creation of the first security issue. Once the first security issue is created, we found that the median time between subsequent security issues is 54.37 days (IQR 107.06 days). To put these results into perspective, we also calculated the numbers for non-security issues and found that the median is 28.21 days (IQR 140.03 days) until the creation of the first non-security issue. Once the first non-security issue is created, we found that the median time between subsequent non-security issues is 2.53 days (IQR 4.68 days). This shows that security issues are created later in a project and emerge less frequently than non-security issues.

We were particularly interested if the activity of a project is related to the number of security issues it contained. When assessing the activity, we used three different metrics. The number of commits is an indicator for the activity in terms of code changes, while the number of forks and the number of stars give an idea of how many developers are involved in a project. The results of our statistical correlation analysis, using the Pearson correlation coefficient to measure the relationship between the number of security issues and all three metrics separately, can be found in table 5.2. We found that there is a statistically significant positive correlation between the number of security issues and all three metrics with the 2-tailed p-value being smaller than 0.001. According to Cohen [4], the correlation coefficient indicates that the effect is medium for the number of commits ( $r=0.374$ ) and strong for the number of forks ( $r=0.552$ ) and stars ( $r=0.593$ ). We can say that the more active a project is (in terms of code contributions and involved persons), the more security issues it contains or vice versa.

The relationship between the number of commits and the number of security issues in a project is displayed in figure 5.1. A quick background check using linear regression analysis showed that the number of non-security issues increases (also in relation to the number of commits) at a higher pace with a standardised regression coefficient of  $\beta = 0.374$  for security issues and  $\beta = 0.503$  for non-security issues.

	<b>r</b>	<b>p</b>	<b>n</b>
<b>commits</b>	0.374*	0.000	182
<b>forks</b>	0.552*	0.000	182
<b>stars</b>	0.593*	0.000	182

Table 5.2: Correlation analysis of activity of a project with number of security issues in project

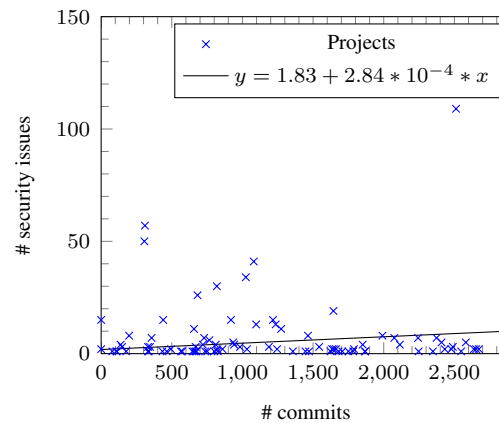


Figure 5.1: Number of commits vs. number of security issues in projects (outliers removed)

## 5.2 Reporters of security issues

We analysed how the reporters of security and non-security issues are associated with the project the issue belongs to and came up with the results in figure 5.2. A noticeable difference exists between security issues and non-security issues. For security issues only 733 of 3 493 (21.0%) issues are reported by outside users who are not associated with the project at all, 1 257 of 3 493 (37.5%) come from core members (members and collaborators) and 1 448 of 3 493 (41.5%) from established contributors to the project. For non-security issues a lot more issues are reported by unassociated users, namely 86 325 of 245 550 (35.2%). Here core members account for 70 626 of 245 550 (31.1%) and contributors for 82 684 of 245 550 (33.7%) of non-security issues. The Chi-Square p-value for these findings is 0.000, so they are significant.

Our comparison between different issue statuses for this feature can be seen in figure 5.3. It reveals that accepted security issues are more often reported by core members of a project than rejected security issues and that the latter are more often reported by users who are unassociated with the project. The p-value for these findings is also 0.000.

In this context it is also interesting to see if the issue reporters are the same per project.

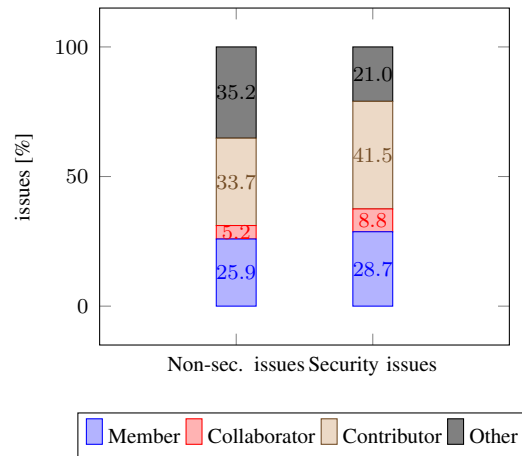


Figure 5.2: Reporters of non-security issues and security issues

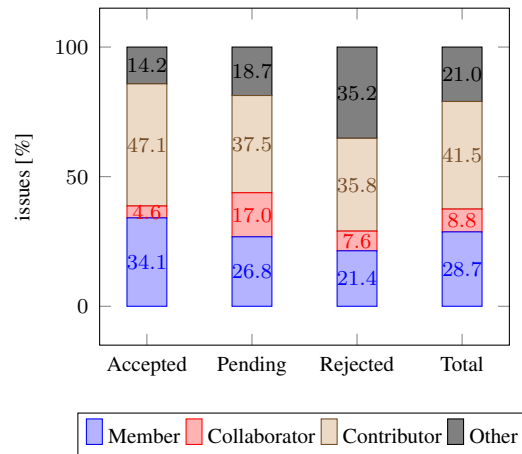


Figure 5.3: Reporters of security issues

We counted how many security issues and non-security issues reporters posted in the analysed projects. We found that for security issues, the mean is 2.37 (SE 0.40) reported issues per reporter, and for non-security issues, the mean is 2.61 (SE 0.72) reported issues per reporter. The median is one issue in both cases. This is not a significant difference. Also, we found that 15 reporters are cross-project security issue reporters, i.e. they created security issues in more than one of the 182 projects in our dataset. Thirteen of those reporters reported security issues in two projects, while two reported in five projects.

Furthermore, we analysed how far reporters of issues in general and security issues in particular overlap. In our dataset we found 30 490 distinct reporters of issues. Only 611 of 30 490 (2.0%) developers reported both security and non-security issues. 29 749

(97.6%) reported only non-security issues and 130 (0.4%) only security issue. This leads us to the conclusion that only 741 of 30 490 developers (2.4%) are reporting security issues. Other than that no notable impact on other characteristics by the type could be measured.

We also did a gender analysis of the reporter of issues in our dataset. We were able to identify the gender of a reporter in 66.8% of the issues in our global dataset of issues, in 62.3% of the security issues and in 66.8% of the non-security issues. Our analysis revealed 155 of 2 175 (7.1%) security issues were reported by females, which is higher than 6 300 of 157 850 (3.8%) for non-security issues. Furthermore, we found that females have an acceptance ratio of 70.4 % for reported security issues vs. 72.0 % for male security issue reporters. This difference is not significant as the p-value is 0.724, so we cannot reject the independence of gender and acceptance rate. Splitting this up even further into core members (members and collaborators) and outsiders (contributors and other) we find that acceptance ratios differ even less for core members (male vs. female: 75.6% vs. 72.0%) and slightly more for outsiders (male vs. female: 57.6% vs. 60.0%). However, none of these findings are significant neither for outsiders nor for core members with p-values of 0.853 and 0.447, respectively. We tried to identify if there were any other observations in characteristics between issues reported by female and male developers and the only significant difference we found was that female-reported security issues have a lower number of comments with 1.41 (SE 0.18) comments vs. 3.08 (SE 0.12) for security issues reported by males.

### 5.3 Reports of security issues

During our manual analysis we investigated two aspects of security issue reports, i.e. the text that the issue reporter submits to GitHub. As described in the methodology section, we analysed the clarity of the issue report and if an issue reporter provides additional information.

As can be seen from figure 5.4, security issues are mostly clearly explained with 276 of 333 (82.9 %) issues considered clear. There is a significant difference (p-value = 0.013) however, between accepted and rejected security issues with 13.0% of accepted issues and 25.3% of rejected issues considered as unclearly explained.

We found that 200 out of 333 security issue reports (60.1%) include further information. Figure 5.5 shows that those issues include different types of information. Most notably accepted security issues more often refer to documentation (44.2%) or reproducibility information (16.9%) than rejected security issues (34.3% resp. 14.1%). Furthermore, there was no occurrence of a pending security issue containing a reference to a CVE entry. This cross-comparison between further information and issue statuses is significant, with a p-value of 0.027.

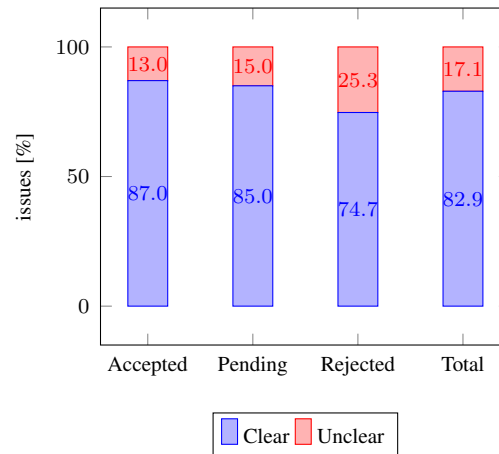


Figure 5.4: Security issue report clarity

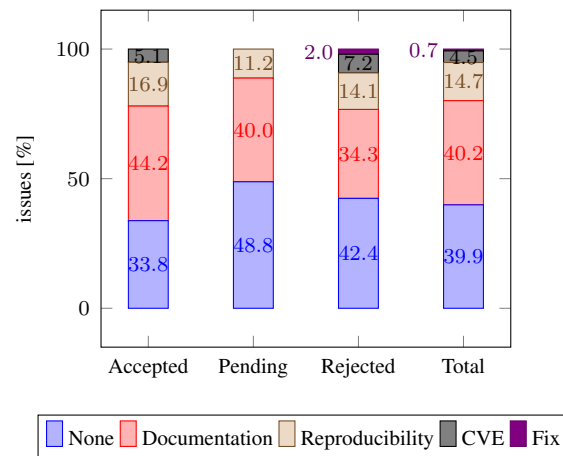


Figure 5.5: Further information in security issue reports

## 5.4 Reaction to security issues

We wanted to know how fast developers react to security issues. In order to answer this question, we analysed two metrics. We looked at the time it takes for the first comment to emerge on a reported security issue (see table 5.3) and the mean time it takes between every comment that follows after the first comment in the discussion (see table 5.4). Naturally, issues without any comments were excluded from this analysis.

Judging by the mean non-security issues get their first comment approximately six days faster than security issues (47.52 days vs. 53.17 days). However, the median reveals a different picture. 50% of all security issues have their first comment within 264.95 minutes vs. 321.28 minutes for non-security issues, which is a difference of nearly one

Metric	Accepted	Pending	Rejected	Sec. issues	Non-sec. issues	Total
n	1 090	563	701	2 354	192 391	194 745
Mean (Std. Error)	35.89 (4.48)	57.21 (7.07)	76.80 (7.77)	53.17 (3.55)	47.52 (0.43)	47.59 (0.43)
Std. Deviation	147.89	167.77	205.60	172.42	188.06	187.88
Minimum	0	0	0	0	0.00	0.00
25th percentile	0.15 [min]	0.03 [min]	8.76 [min]	0.18 [min]	5.73 [min]	5.65 [min]
Median	172.22 [min]	106.56 [min]	1.17	264.95 [min]	321.28 [min]	320.78 [min]
75th percentile	6.82	6.97	32.53	12.52	5.89	5.93
Maximum	2 215.97	1 232.21	1 724.11	2 215.97	2 848.49	2 848.49

Table 5.3: Time until first comment [days]

Metric	Accepted	Pending	Rejected	Sec. issues	Non-sec. issues	Total
n	769	401	434	1 604	134 380	134 984
Mean (Std. Error)	38.36 (3.84)	83.54 (9.22)	79.70 (8.28)	60.84 (3.74)	74.22 (0.47)	74.06 (0.46)
Std. Deviation	106.48	184.62	172.42	149.79	171.64	171.40
Minimum	0.00	0.00	0.00	0.00	0.00	0.00
25th percentile	9.30 [h]	23.06 [h]	11.98 [h]	12.11 [h]	5.26 [h]	5.31 [h]
Median	4.23	14.94	11.06	6.87	4.71	4.74
75th percentile	24.43	92.52	73.16	50.47	56.32	56.21
Maximum	1 567.72	2 017.01	1 707.25	2 017.01	3 517.41	3 517.41

Table 5.4: Mean time between comments [days]

hour. Standard deviation is very high with this metric indicating that the data is very spread out and outliers (e.g. issues taking years until their first comment) are influencing the mean. This explains the big difference between mean and median values.

With our second metric, the mean time between subsequent comments in the issue discussion, we find a similar picture with outliers influencing the distribution. Here the quartile and median values suggest that the discussions proceed slower in security issues than in non-security issues (median 6.87 days vs. 4.71 days).

When analysing data for this question, we need to keep in mind that comparisons between security and non-security issues should be made with caution. Due to class imbalance, the data for non-security issues is suspected to be stronger influenced by the extensive use of bots. In our manual analysis 73 out of 333 security issues (21.9%) were first reacted to by a bot.

We also compared these two metrics between accepted, pending and rejected security issues. As can be seen from table 5.3 rejected security issues have a much slower reaction time than accepted security issues, which is less than half the time with 35.89 (SE 4.48) days vs. 76.80 (SE 7.77) days. Also, the mean time between discussion comments is significantly longer in pending and rejected security issues than it is in accepted security issues, as can be seen from table 5.4.



## 5.5 Resolution of security issues

The resolution time of security issues is a very important metric. When observing the gathered data for the resolution time of issues in table 5.5, we can see that the data is very spread out with a standard deviation of 283.51 days. 25% of the security issues are resolved within 1.90 days or less, and 25% take 94.88 days (approximately three months) or more to be resolved. When judged by the mean security issues are resolved slightly faster than non-security issues (110.08 days vs. 116.97 days), when judging by the median; however, non-security issues are resolved in half the time of security-issues (8.06 days vs. 15.32 days). When analysing the data per project, we found that in 94 out of 182 available projects (51.6%) security issues were on average resolved slower than non-security issues.

The comparison between accepted and rejected security issues is also shown in table 5.5. We can see that accepted security issues are resolved significantly faster than rejected security issues with a mean of 84.95 (SE 5.03) vs. 153.32 (SE 9.61) days. As a comparison we can also add the mean age of pending security issues here, which is 625.94 (SE 17.84) days.

We also analysed how long issues without a single comment have been pending and how long the resolution time for accepted and rejected issues with zero comments was. We found that in such cases non-security issues have been pending only insignificantly longer than security issues with means of 614.65 (SE 5.18) days vs. 602.29 (SE 28.28) days. 50% of such non-security issues have been pending for more than 420 days, and 50% of such security issues have been pending for more than 417 days. For closed issues without any comments the resolution times were on average 34.58 (SE 0.67) days and 33.49 (SE 4.17) days for accepted non-security and security issues respectively while 132.02 (SE 3.50) days and 48.27 (SE 11.92) days for rejected non-security and security issues respectively. This shows us that issues without any comment get rejected significantly faster in the case of security issues than for non-security issues.

Further results concerning the resolution time of security issues are presented in section 5.9.

Metric	Accepted	Rejected	Sec. issues	Non-sec. issues	Total
n	1 598	929	2 527	209 939	212 466
Mean (Std. Error)	84.95 (5.03)	153.32 (9.61)	110.08 (4.80)	116.97 (0.62)	116.89 (0.62)
Std. Deviation	201.25	293.10	241.36	283.98	283.51
Minimum	0.00	0.00	0.00	0.00	0.00
25th percentile	2.14	0.99	1.90	0.53	0.54
Median	13.12	24.23	15.32	8.06	8.11
75th percentile	70.39	138.87	94.88	74.26	74.59
Maximum	2 215.98	1 954.81	2 215.98	3 311.65	3 311.65

Table 5.5: Resolution time of closed issues [days]

## 5.6 Discussion of security issues

### 5.6.1 Participants in discussion

We analysed how many developers participate in discussions of security issues. In order to answer this question, we excluded pending issues and only included closed issues into our analysis because a lot of very recently created issues do not yet have any participants in their discussions. Looking at the number of distinct participants in issue discussions in table 5.6 we can see that there is no real difference between security issues and non-security issues, except for a few high outliers with non-security issues that are expected due to the much higher number of cases. In total, issue discussions have a mean of 1.55 (SE 0.00) participants with a median of 1. When analysing the data per project, we found that in 91 out of 182 available projects (50.0%), security issues have more participants in discussions than non-security issues.

The distribution of the number of participants in closed security issue discussion can be seen in figure 5.6. This figure reveals a minor difference between accepted and rejected security issues. In accepted security issues there are more cases with no participants, i.e. no comments at all, than one participant, whereas with rejected security issues, it is vice versa.

We also analysed how many developers participate in the security issue discussions of a whole project. We found that on average only 7.78 (SE 1.47) developers are involved in all security issues of a project combined. The median is even lower with only three developers. This is very few especially when we compare it to the number of participants in non-security discussions of a whole project where we find that on average 4 628.79 (SE 725.32) developers are involved in all issues of a project combined, with the median being 1 550 participants.

In this context it is interesting to know how far participants in issue discussions in general and in security issue discussions in particular overlap. In our dataset we found 37 093 distinct participants of issue discussions. Only 941 of 37 093 (2.5%)

<b>Metric</b>	<b>Sec. issues</b>	<b>Non-sec. issues</b>	<b>Total</b>
n	2 527	209 939	212 466
Mean (Std. Error)	1.51 (0.03)	1.55 (0.00)	1.55 (0.00)
Std. Deviation	1.66	1.47	1.47
Minimum	0	0	0
25th percentile	0	1	1
Median	1	1	1
75th percentile	2	2	2
Maximum	17	43	43

Table 5.6: Number of participants in closed issue discussions

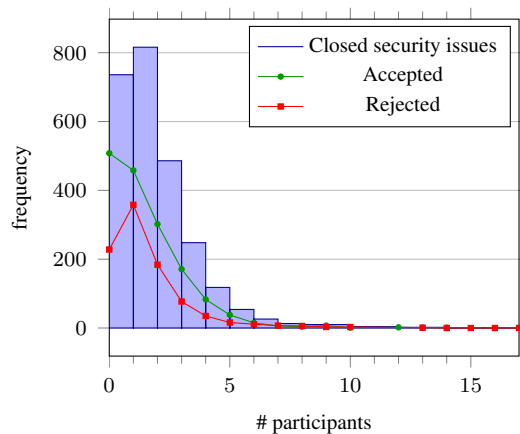


Figure 5.6: Number of participants in closed security issue discussions

developers participated in both security and non-security issue discussions. 35 904 (96.8%) participated only in non-security issue discussions and 248 (0.7%) only in security issue discussions. This leads us to the conclusion that only 1 189 of 37 093 developers (3.2%) are involved in security issue discussions.

Furthermore, we checked how frequently developers participate in issue discussions. For each participant in security issue discussions we calculated the ratio between the total number of security issues in a project and the number of security issues in a project that he or she participated in. We found that the developers participate in more than one-fourth of the security issue discussions of a project with the mean ratio being 0.25 (SE 0.01). Of course, this value is magnitudes lower for non-security issues where the mean participation ratio is only 0.0083 (SE 0.00029).

## 5.6.2 Comments in discussion

First of all, we counted how many comments there were in issue discussions. We found that security issue discussions have a mean of 2.87 (SE 0.09) comments. This is less than non-security issue discussions that have a mean of 3.47 (SE 0.01) comments. The detailed statistics are visible in table 5.7. 75% of all security issue discussions have three or fewer comments; the median is one single comment. When analysing the data per project, we also found that in 81 out of 182 analysed projects (44.5%) security issue discussions contained on average fewer comments than non-security issue discussions in the same project. The distribution of the comment count in security issue discussion can be seen as a histogram in figure 5.7.

We also compared this feature among different issue statuses for security issues. The results can be seen in the left part of table 5.7. The only significant difference is between the pending and all other security issues. Pending issues have fewer comments

Metric	Accepted	Pending	Rejected	Sec. issues	Non-sec. issues	Total
n	1 598	966	929	3 493	245 550	249 043
Mean (Std. Error)	3.00 (0.13)	2.38 (0.15)	3.15 (0.21)	2.87 (0.09)	3.47 (0.01)	3.46 (0.01)
Std. Deviation	5.14	4.55	6.29	5.33	5.52	5.52
Minimum	0	0	0	0	0	0
25th percentile	0	0	1	0	1	1
Median	1	1	1	1	2	2
75th percentile	4	3	3	3	4	4
Maximum	67	60	116	116	385	385
Total comments	4 797	2 301	2 923	10 021	850 927	860 948

Table 5.7: Number of comments in issue discussions

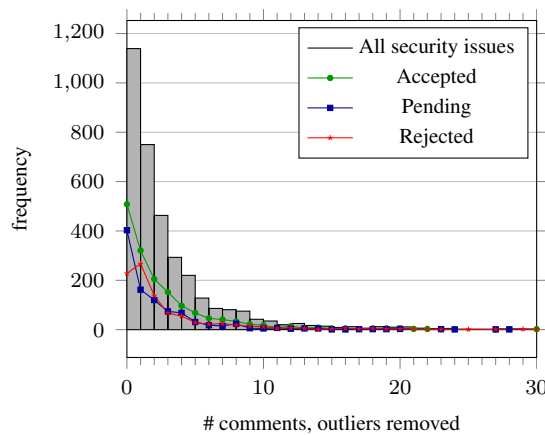


Figure 5.7: Number of comments in security issue discussions

than accepted and rejected issues. It is also worth mentioning that 508 of 1 598 (31.8%) accepted security issues had zero comments, 403 of 966 (41.7%) pending security issues had zero comments and 228 of 929 (24.5%) rejected security issues also had zero comments. As with the participants in the previous section, we can observe here that there are fewer rejected security issues with zero comments than with one comment.

We further analysed the number of comments that the issue reporter of a security issue made in the discussion (after the initial issue report). We found that in 144 out of 333 security issues (43.8%), the issue reporter did not leave any more comments. The mean value is 1.39 comments (SE 0.14) with a standard deviation of 2.55. The distribution is visualised in figure 5.8, where we see that no notable differences exist between different issue statuses. On a large scale we also analysed this question for closed security issues vs. closed non-security issues. We found that in security issues the author is posting a mean of 1.13 (SE 0.05) comments and in non-security issues a mean of 1.81 (SE 0.00) comments. This means that, admittedly on a low level, the issue reporter is leaving slightly more comments in the discussion of “his or her” issue in non-security issues than in security issues.

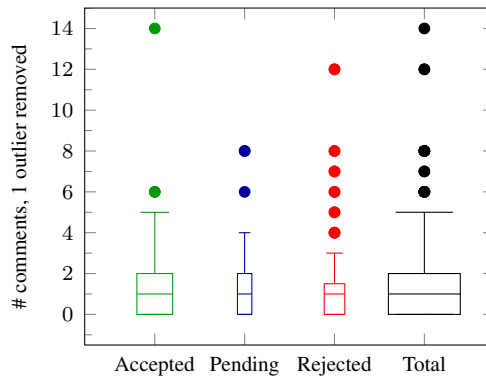


Figure 5.8: Number of comments by the issue reporter in security issue discussions

As described in the methodology chapter, we manually investigated various qualitative features of the comments in security issue discussions. Our results are presented in the following paragraphs.

First and foremost, we categorised discussion comments based on their nature. Both figure 5.9 and figure 5.10 are based on the same data, but they emphasis two different perspectives of comment natures in security issue discussions. Figure 5.9 compares the percentages that issue comments of a certain nature take up in security issue discussion across the three issue statuses accepted, pending and rejected. In figure 5.10, on the other hand, every bar represents the mean composition of a security issue discussion with the corresponding issue status. We find that elaborative comments are much more common in accepted (30.6%) and pending (33.2%) security issues compared to rejected security issues (20.4%). Notably, the share of comments that are of a thematically relevant nature (elaboration, ask/request and instruction) differs significantly between accepted and rejected security issues (39.7% vs. 30.1%). Other things worth noting are that notifications make up for more than half of all comments in rejected security issue discussions (53.4%) and that reminders are a lot more common in still pending security issues (29.6%). Our cross-comparison between comment natures and issue statuses is significant, with a p-value of 0.000.

For the 145 reminders that we found during our manual analysis, we wanted to see if they speed up the resolution process of the security issues. We were unable to find any significant evidence that would support this assumption. We found though, that 98 of 145 (67.6%) reminders were posted by bots.

Looking at our next feature, we found that developers are generally quite confident with 73.1% of all elaborative comments being confident. When comparing different issue statuses in figure 5.11, we see that developers are only slightly more confident in accepted security issues (75.5%) than they are in pending (71.1%) or rejected security issues (70.1%). The cross-comparison between confident comments and issue statuses is

significant, with a p-value of 0.049.

As with the issue reports, we also checked if developers provided any further information in their comments during the discussions of security issues. As shown in figure 5.12, nearly 80% of all comments in security issue discussions do not contain any further information directly or via a hyperlink. Few occurrences of documentation and fixes were found, very few references to CVE entries or reproducibility information. There were no notable differences among the different issue statuses.

Very similar to further information, mentioning other people is not very common in security issue discussions. As shown in figure 5.13, in only 30.9% of all comments was someone mentioned. Core members or established contributors to the project were mentioned in 20.8% and the initial author of the security issue in 6.2% of comments.

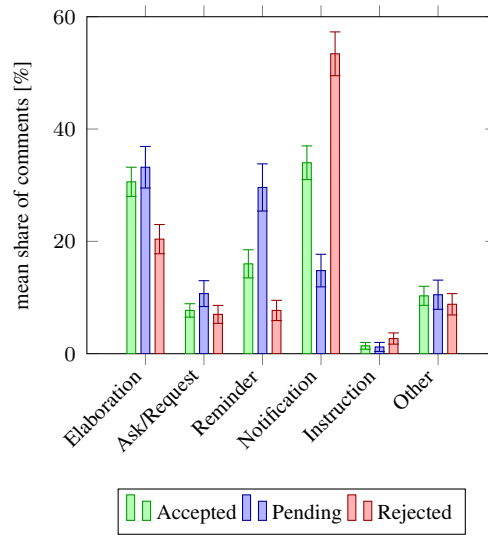


Figure 5.9: Distribution of comment natures in security issue discussions (error bars indicate standard error of mean)

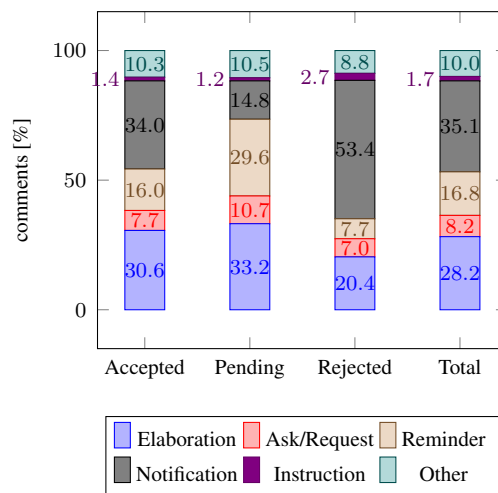


Figure 5.10: Mean composition of security issue discussions

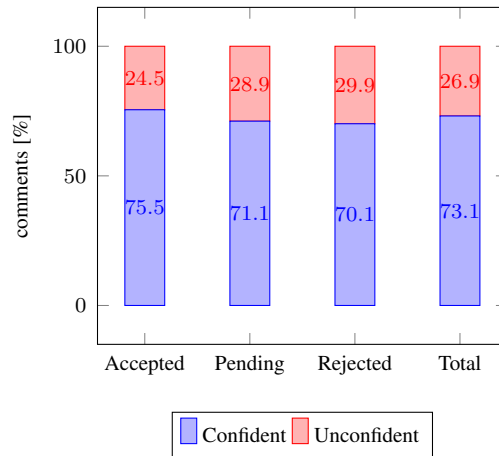


Figure 5.11: Confident vs. unconfident elaboration in security issue discussions

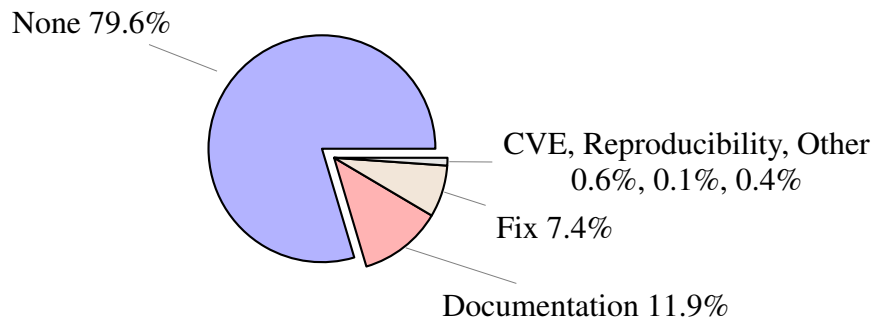


Figure 5.12: Further information in security issue discussions

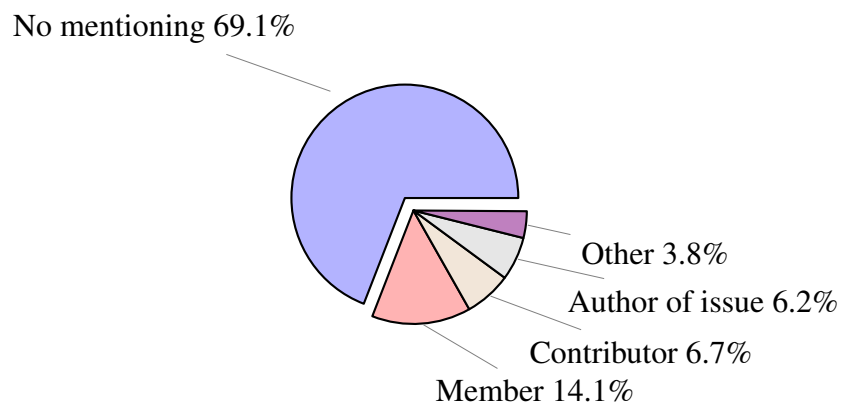


Figure 5.13: Mentioning in security issue discussions



## 5.7 Assignment of security issues

In this section we present our results concerning the assignee feature of GitHub. We found that in 1 698 of 3 493 (48.6%) security issues was someone assigned to the issue, whereas for non-security issues this share was lower with 107 923 of 245 550 (44.0%) having at least one assignment. In figure 5.14 we can see the comparison of the number of assignees between the different statuses of security issues (p-value = 0.000). We find that for accepted issues 1 069 of 1 598 (66.9%) issues have one or multiple assignees, while for pending issues only 316 of 966 (32.7%) and for rejected issues only 313 of 929 (33.7%) security issues have an assignee. We observe that security issues have at most

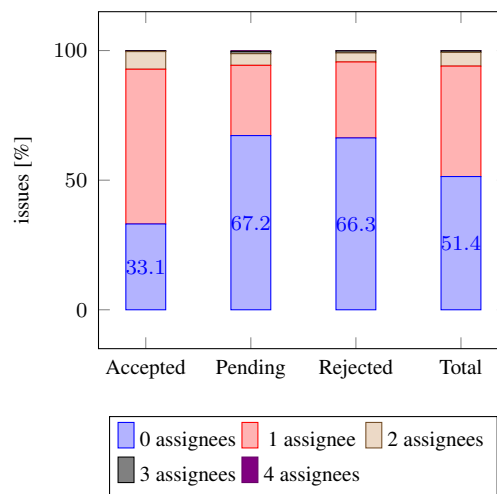


Figure 5.14: Assignees of security issues

four assignees and rarely (3.8%) more than one assignee. We can conclude that accepted issues have most assignees while in pending and rejected issues about two-thirds of the security issues do not have an assignee. This is also confirmed by the mean values with 0.71 (SE 0.01) assignees for accepted, 0.31 (SE 0.02) for pending and 0.33 (SE 0.02) for rejected security issues. In reverse, this also yields a much higher acceptance rate for security issues that have an assignment with 77.4% vs. 46.3% for issues with no assignment.

Further analysis shows that in 1 075 of 1 698 (63.3%) security issues with assignees, actually a self-assignment took place, i.e. a user assigned himself to a security issue. Additionally, in 444 of 1 698 (26.2%) issues, the reporter of the security issue is added as an assignee. A developer who is assigned to a security issue is posting an average of 1.44 (SE 0.06) comments on that issue.

Furthermore, we were interested in the developers who were assigned to security issues, which we will call security assignees for this section. In the 1 698 security issues that had at least one assignee, we identified 328 such security assignees. We found that

those 328 security assignees reported 1 505 of 3 493 (43.1%) of the security issues in our dataset. The mean number of reported security issues by such a security assignee is 7.00 (SE 0.80). This is significantly more than the 2.37 (SE 0.40) security issues for the “normal” developer that we calculated in section 5.2.

Additionally, those 328 security assignees were responsible for 3 942 of 9 901 (39.8%) comments in all the security issues. 1 522 of 3 493 (43.6%) security issues have received at least one comment by a security assignee. The mean number of total comments across all security issues by a security assignee is 15.28 (SE 1.97) comments. The mean number of comments per security issue they are involved in is 2.59 (SE 0.08) comments. In issues they are assigned to security assignees leave a mean of 1.00 (SE 0.03) comments, whereas in security issues of their project that they are not assigned to only they leave a mean of 0.22 (SE 0.03) comments. We can conclude by stating that security assignees were involved in 2 230 of 3 493 (63.8%) security issues by either reporting or commenting.

Further investigations revealed that of the 1 117 developers who were not security assignees, 1 079 (96.6%) have fewer comments in security issues and fewer security issue reports than the average security assignee. Six persons have more reports but fewer comments, 18 have fewer comments but more reports and only seven have more comments and more reports and outperform security assignees in both metrics. We also found that the acceptance rate for security issues reported by security assignees is 77.7%, which is higher than the global acceptance rate for security issues of 63.2%. Also, the mean resolution time for security issues reported by security assignees is 80.86 (SE 5.95) days, which is significantly shorter than the global resolution time for security issues of 110.08 (SE 4.80) days that we calculated in section 5.5.

Finally, we also analysed when the first assignment of a security issue happens. On average, the first assignment takes place 51.00 (SE 4.52) days after the creation of a security issues. It happens earlier for accepted issues with a mean of 39.29 (SE 5.30) days than for pending and rejected issues with means of 87.19 (SE 13.10) and 54.49 (SE 9.69) days respectively. At the time of the first assignment, a mean of 1.34 (SE 0.09) comments have already been left on the security issues, and a mean of 2.81 (SE 0.11) comments follow after the first assignment.

## 5.8 Pull requests related to security issues

Submitting a pull request to a repository is the most common way to fix an issue on GitHub. We found that 106 out of 154 analysed accepted security issues (68.8%) were fixed by a pull request, while with the majority of other instances namely in 44 issues (28.6%) the implementation was made with a direct commit to the repository without a pull request. In four cases it was unknown where or how a fix was implemented. The distribution is visualised in figure 5.15. Furthermore, we found that 12 of 80 pending security issues (15.0%) had a pull request and six of 99 rejected security issues (6.1%)

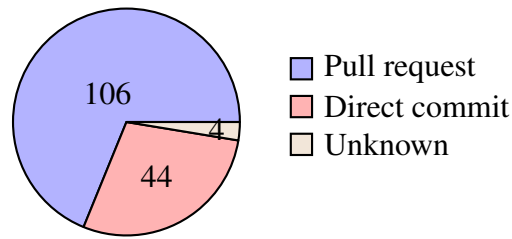


Figure 5.15: Implementation of accepted security issues

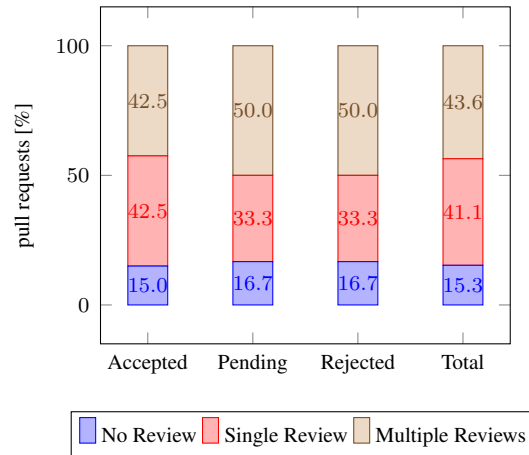


Figure 5.16: Review situation of pull requests

as well. We were interested why those six pull requests were rejected and found that in five cases a fix was implemented and merged in another (unlinked or not properly linked) pull request and in one case the pull request was actually closed unmerged and subsequently the corresponding issue was closed as well, without any further explanation. We already saw that 31.2% of accepted security issues were closed without a pull request, for rejected security issues this number is naturally much higher with 93 of 99 (94.0%).

A key functionality of pull requests on GitHub is the possibility to add users as reviewers so that they take a look and evaluate the proposed code changes. Fortunately, we found that 105 of 124 (84.7%) analysed pull requests were reviewed at least once by another person than the pull request creator. In 54 of 124 (43.6%) pull requests there were even multiple reviews carried out. Across different issue statuses there were no notable differences (figure 5.16). However, we need to keep in mind that data is scarce here with only twelve and six pull requests available for pending and rejected security issues, respectively.

We were interested in the role of a security issue reporter in a pull request that emerged from a security issue. Figure 5.17 shows us that in 50.8% of analysed cases, the issue reporter is not involved in the pull request emerging from a security issue at all.

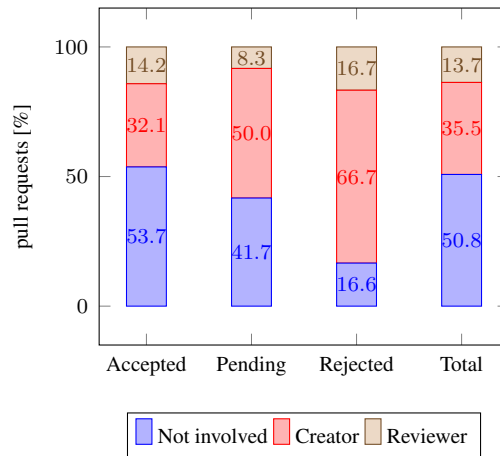


Figure 5.17: Role of the security issue reporter in corresponding pull request

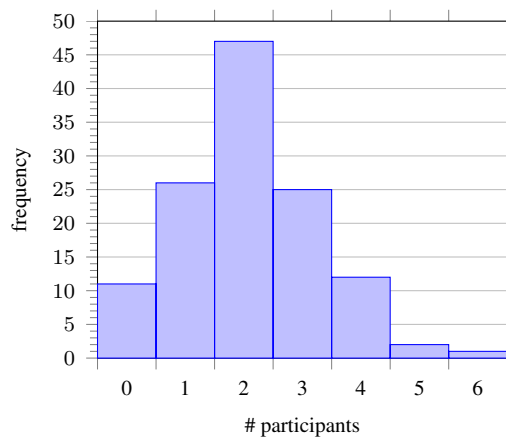


Figure 5.18: Number of participants in pull request discussions

There are some differences between the different issue statuses, but we need to keep in mind that data is scarce here with only twelve and six pull requests available for pending and rejected security issues, respectively. Still, it is interesting to see that in four of six rejected security issues with a pull request, the creator of the issue also created the pull request.

Naturally, we also analysed the discussions of pull requests. The number of participants in pull request discussions emerging from security issues follows a normal distribution with a mean of 2.09 (SE 0.11) and a standard deviation of 1.20, as can be seen in figure 5.18. No pull request discussion with more than six participants was found across all 124 analysed pull requests.

The distribution of the number of comments in the 124 analysed pull request discus-

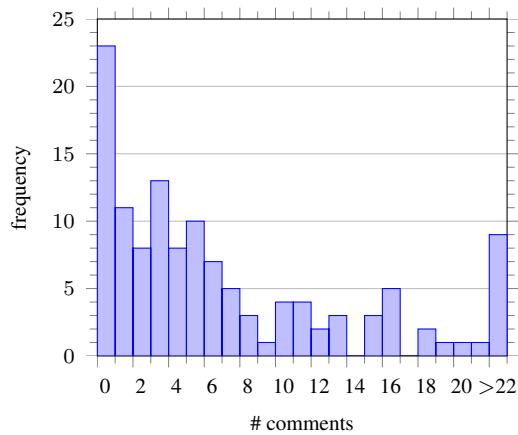


Figure 5.19: Number of comments in pull request discussions

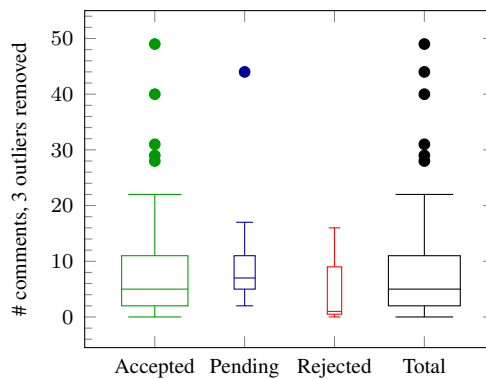


Figure 5.20: Number of comments in pull request discussions

sions emerging from security issues can be seen in figure 5.19 and a comparison between the different issue statuses in figure 5.20. We found that pull request discussions have a mean of 10.06 comments (SE 1.48) with a standard deviation of 16.52. As can be seen from figure 5.19, the data is widely spread out with several outliers (maximum is 110 comments) and nine pull requests having 22 or more comments which influence the mean heavily. Comparisons between different issue statuses in figure 5.20 should be made tentatively because data is scarce with pending and rejected security issues pull requests, but there is a visible tendency that pending pull requests have slightly more comments than accepted and rejected pull requests.

Now that we have analysed the number of comments in issue and pull request discussions, it seems logical to compare them. The relation between the number of comments in the discussion of the security issue itself and the number of comments in the pull request discussion is shown in figure 5.21. Statistical correlation analysis

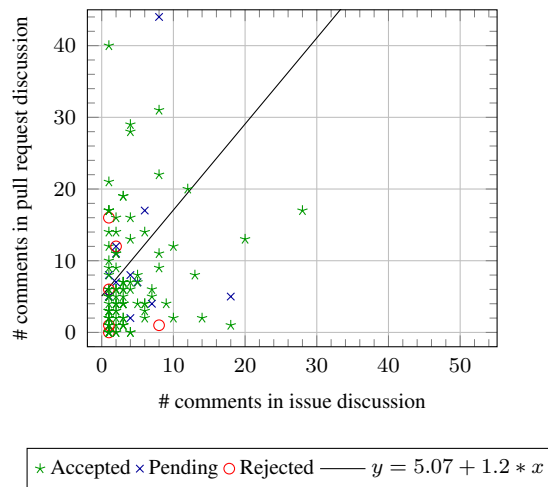


Figure 5.21: Number of comments in security issue discussions vs. number of comments in pull request discussions

using Pearson's correlation coefficient has shown that the number of comments in the discussion of the security issue and the number of comments in the pull request discussion correlate significantly ( $r = 0.427$ ,  $p = 0.000$ ,  $n = 124$ ). The more comments there are in the issue discussion, the more comments there are in the discussion of the pull request or vice versa. According to Cohen [4], the strength of this effect is medium. Recalling the results from table 5.7 we can also state that pull request discussions have a significantly higher number of comments than security issue discussions with a mean of 10.06 (SE 1.48) comments vs. 2.87 (SE 0.09) comments for the issue discussions (for accepted security issues also only 3.00 (SE 0.13) comments). This is also visible in figure 5.21.

Finally, we had 62 instances of security issues available where we identified a key point in the discussion and also a pull request corresponding to this issue was created. We found that in 27 cases the key point was before the pull request, and in 35 cases the key point was after the pull request. Whenever the key point happened before the pull request, the discussion of the issue generally came to a conclusion, and there was consensus that a pull request can now be created to implement a fix. When the key point happened after the pull request, it was generally a notification that informed the community that a pull request that addresses this security issue has been created and merged and that this issue can now be closed.

## 5.9 Trends in security issues over the age

As the resolution time of a security issue is critical in mitigating potential damage, we were interested if we could identify any other characteristics that potentially have an impact on resolution time. This is what we report in the first part of this section. In the second part, where we also take pending security issues into consideration, we look at how certain characteristics of security issues evolve as the issue ages.

The results of the correlation analysis between other numeric features and the resolution time are shown in table 5.8. We can see that there are four statistically significant correlations indicated by an asterisk in the table. We find that there is a positive, yet weak according to Cohen [4], correlation between the resolution time of a security issue and its number of comments, number of participants and number of participants in the pull request. This means that the bigger the discussion of a security issue is, the longer it takes to resolve it or vice versa. Additionally, we have a strong correlation between the resolution time of a security issue and its reaction time, i.e. the time until the first comment. This signifies the importance of the first comment in a security issue. The faster there is a first comment, the faster a security issue is resolved or vice versa.

	<b>r</b>	<b>p</b>	<b>n</b>
<b># comments</b>	0.228*	0.000	2 527
<b># participants</b>	0.230*	0.000	2 527
<b># assignees</b>	-0.019	0.352	2 527
<b>reaction time</b>	0.637*	0.000	1 791
<b># comments PR</b>	0.104	0.276	112
<b># participants PR</b>	0.193*	0.042	112

Table 5.8: Correlation analysis of resolution time with other numerical characteristics

At this point, we should also add that we found a statistically significant weak correlation between the age of a project and the number of participants involved in security issues ( $r = 0.289$ ,  $p = 0.000$ ,  $n = 3\,493$ ) and also between the age of a project and the number of participants involved in pull requests corresponding to security issues ( $r = 0.203$ ,  $p = 0.024$ ,  $n = 124$ ). This means that more participants are involved in security issues as the project gets older or vice versa.

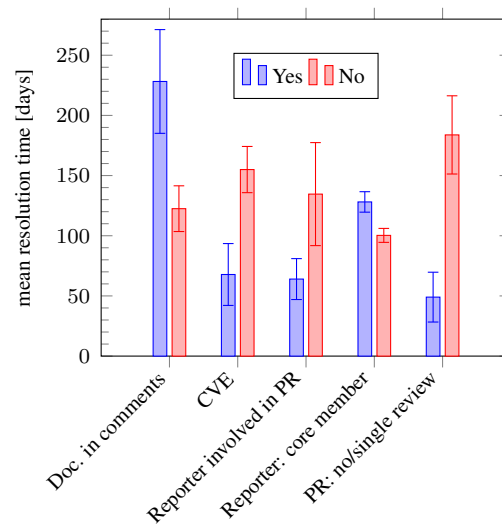


Figure 5.22: Mean resolution time of closed security issues for different values of categorical features

For the categorical features we recorded in the manual analysis, we compared the mean resolution time for the security issues with different values. We were able to make the following observations (the following enumeration corresponds to the x-axis in figure 5.22):

1. Security issues where additional documentation is provided in the comments are resolved significantly slower.
2. Security issues with a CVE reference in the report or the comments are resolved significantly faster.
3. Security issues where the issue reporter is involved in the pull request are resolved significantly faster.
4. Security issues that are reported by core members (members and collaborators) are resolved significantly slower.
5. Security issues with pull requests that were not reviewed or reviewed once are resolved significantly faster than security issues with pull requests that were reviewed more than once.

Furthermore, we checked if the mean resolution time was different when a security issue has an assignee. We found that there is no significant difference between the mean resolution times of security issues with assignee 109.81 (SE 6.36) days vs. 110.42 (SE 7.31) days for security issues without an assignee. Interestingly though, there are



<b>ageBin</b>	<b>Min age [days]</b>	<b>Max age [days]</b>	<b># issues in full dataset</b>	<b># issues in sample</b>
0	0	0.25	348	24
1	0.25	2.27	351	30
2	2.27	7.34	350	36
3	7.34	20.95	351	30
4	20.95	47.30	346	30
5	47.30	103.99	350	37
6	103.99	193.46	349	39
7	193.46	426.71	350	32
8	426.71	898.27	349	37
9	898.27	3042.50	349	38

Table 5.9: Age binning of security issues

differences between issue statuses. Accepted issues have a slightly longer resolution time if they have an assignee with 88.29 (SE 6.44) vs. 78.19 (SE 7.84) days, while rejected issues have a significantly longer resolution time if they have an assignee with 183.29 (SE 16.78) days vs. 138.09 (SE 11.69) days. Finally, pending issues have a significantly “younger” age if they have an assignee, namely 553.29 (SE 29.09) days vs. 661.25 (SE 22.31) days for pending security issues without an assignee. No significant differences could be identified for the other categorical features.

In order to analyse how security issue evolve when they age, we grouped the 3 493 security issues in our dataset into ten “age bins”, according to the 10%-quantiles of the age of the issues. The binning details are shown in table 5.9. We then made figure 5.23 to show the number of accepted, pending and rejected security issues per age bin. We see that in general as security issues age the numbers of accepted and rejected issues decrease (because they are closed) and the number of pending issues increases of course. For very short-lived or very old issues there are more rejected than accepted security issues, while for issues between two days and two months old, there are more than twice as many accepted issues than rejected issues. The cross-comparison between the age bin and the issue statuses is significant with a p-value of 0.000.

In figure 5.24 and table 5.10 we see the evolution of the number of comments in security issue discussions over the age of the issue. Overall we can say that the more issues age, the more comments they receive. Pending issues generally have fewer than three comments, while accepted and rejected issues are similarly discussed.

Finally, in figure 5.25 and table 5.11 we see the evolution of the number of participants in security issues discussions over the age of the issue. We observe that accepted and rejected issues are generally discussed by fewer than two participants; issues older than half a year involve more participants though. In pending issues the number of participants increases, but it seems that if an issue is not resolved after half a year, fewer participants

get interested in joining the discussion. We also analysed if the occurrences of further information (documentation, reproducibility information, cve or fix) varied over the age of issues, but we did not find any notable changes.

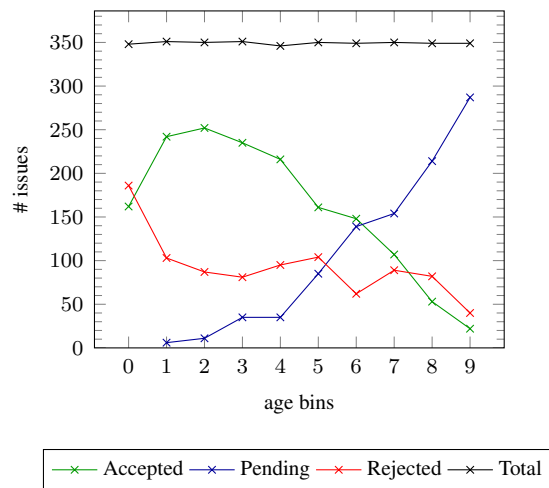


Figure 5.23: Number of security issues per age group

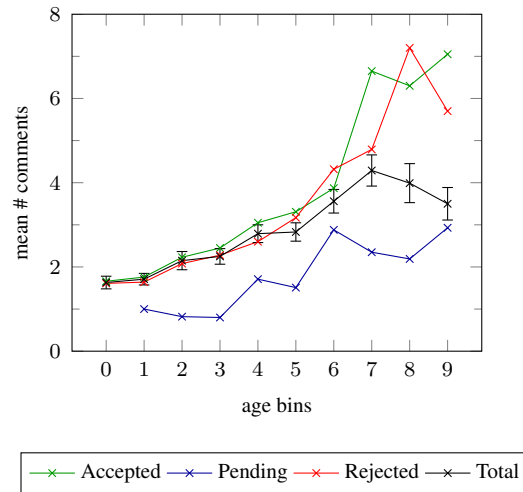


Figure 5.24: Mean number of comments of security issues per age group

ageBin	Accepted Mean (SE)	Pending Mean (SE)	Rejected Mean (SE)
0	1.66 (0.24)	-	1.61 (0.19)
1	1.76 (0.17)	1.00 (0.26)	1.64 (0.23)
2	2.23 (0.28)	0.82 (0.40)	2.08 (0.29)
3	2.45 (0.25)	0.80 (0.27)	2.28 (0.28)
4	3.05 (0.29)	1.71 (0.48)	2.60 (0.35)
5	3.31 (0.31)	1.51 (0.28)	3.17 (0.49)
6	3.87 (0.41)	2.88 (0.45)	4.32 (0.69)
7	6.65 (1.00)	2.35 (0.25)	4.79 (0.60)
8	6.30 (1.25)	2.19 (0.24)	7.20 (1.62)
9	7.05 (1.75)	2.93 (0.36)	5.70 (1.85)

Table 5.10: Mean number of comments of security issues per age group

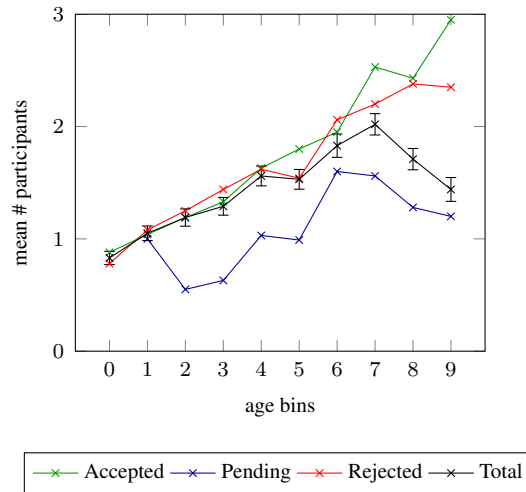


Figure 5.25: Mean number of participants in security issues per age group

<b>ageBin</b>	<b>Accepted Mean (SE)</b>	<b>Pending Mean (SE)</b>	<b>Rejected Mean (SE)</b>
0	0.88 (0.10)	-	0.78 (0.07)
1	1.04 (0.08)	1.00 (0.26)	1.08 (0.13)
2	1.19 (0.10)	0.55 (0.28)	1.25 (0.11)
3	1.33 (0.10)	0.63 (0.18)	1.44 (0.13)
4	1.63 (0.11)	1.03 (0.25)	1.62 (0.18)
5	1.80 (0.13)	0.99 (0.15)	1.54 (0.18)
6	1.95 (0.16)	1.60 (0.17)	2.06 (0.24)
7	2.53 (0.19)	1.56 (0.13)	2.20 (0.19)
8	2.43 (0.30)	1.28 (0.10)	2.38 (0.20)
9	2.95 (0.66)	1.20 (0.10)	2.35 (0.40)

Table 5.11: Mean number of participants in security issues per age group

## 5.10 Trends in security issues over the years

In this section we report our observations about security issues over the course of the last few years. Our dataset contains security issues that were created between June 2010 and April 2020. Unfortunately, the number of issues before 2014 in our dataset is extremely low and would not allow for accurate analysis. This is why we focus on issues created from 2014 until 2020 in this section. In figure 5.26 we see the number of security issues over the last six years. The cross-comparison of the creation year and the issue statuses is significant with a p-value of 0.000. What is evident is a big increase in the number of security issues after 2016. We have to keep in mind though, that our selection only includes projects that were pushed to at least once after 01.01.2019 (see section 4.1). The decrease in 2020 is explained by the fact that at the time of writing we could only include issues until April 2020.

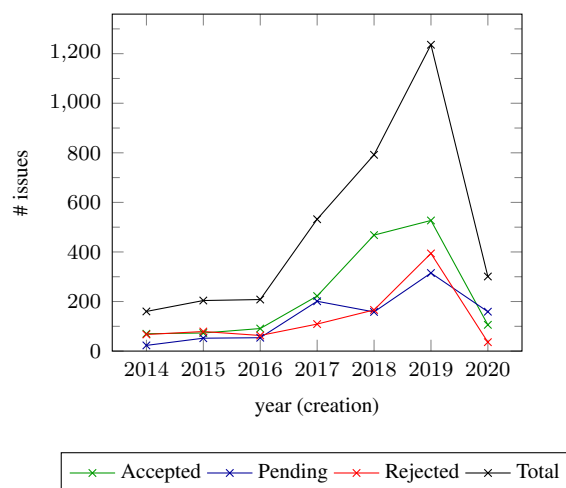


Figure 5.26: Number of security issues per year

Figure 5.28 and table 5.12 show the development of the mean number of comments in security issues discussions from 2014 to 2020. This graph suggests that in recent years fewer comments have been needed to resolve issues. We need to keep in mind however, that this analysis is biased, as the data point for every year can only include issues up to the age of the difference between that year and today. So data points in earlier years include older issues, while data points in more recent years do not. 75% of security issues are resolved in less than 95 days though, which should mitigate this problem.

Figure 5.29 and table 5.13 show the development of the number of participants in security issues discussions from 2014 to 2020. It is obvious to see that the number of participants involved has not really changed over the years.

Finally, we analysed if there are any trends in the resolution time of security issues over the years in figure 5.27. But as with the number of comments, we need to keep

in mind that earlier years include longer-lasting security issues. One interesting thing though, is that rejected security issues are resolved slower than accepted security issues.

As with the trends over the age, we also analysed if the occurrences of further information (documentation, reproducibility information, cve or fix) varied over the last years, but we did not find any notable changes to report at this point.

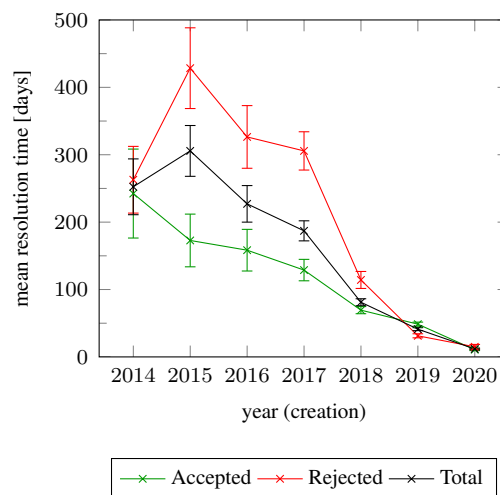


Figure 5.27: Mean resolution time of closed security issues per year

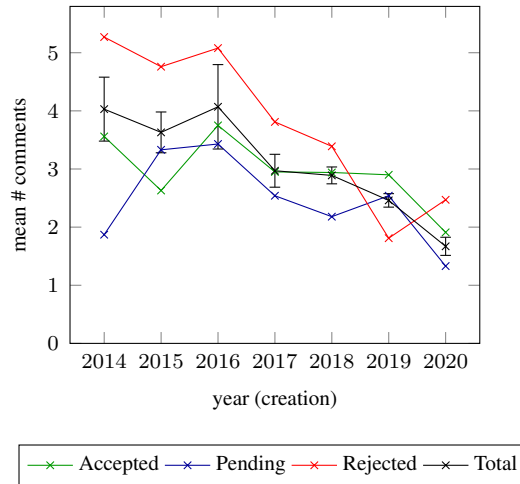


Figure 5.28: Mean number of comments of security issues per year

<b>year</b>	<b>Accepted Mean (SE)</b>	<b>Pending Mean (SE)</b>	<b>Rejected Mean (SE)</b>
2014	3.56 (0.57)	1.87 (0.49)	5.27 (1.15)
2015	2.63 (0.42)	3.33 (0.71)	4.76 (0.66)
2016	3.75 (0.63)	3.43 (1.13)	5.08 (2.01)
2017	2.95 (0.52)	2.54 (0.38)	3.81 (0.53)
2018	2.94 (0.21)	2.18 (0.25)	3.39 (0.28)
2019	2.90 (0.19)	2.54 (0.24)	1.81 (0.18)
2020	1.91 (0.30)	1.33 (0.19)	2.47 (0.49)

Table 5.12: Mean number of comments of security issues per year

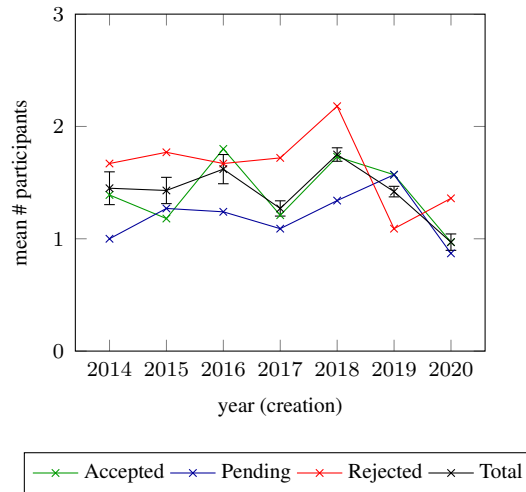


Figure 5.29: Mean number of participants in security issues per year

<b>year</b>	<b>Accepted Mean (SE)</b>	<b>Pending Mean (SE)</b>	<b>Rejected Mean (SE)</b>
2014	1.39 (0.21)	1.00 (0.20)	1.67 (0.26)
2015	1.18 (0.15)	1.27 (0.30)	1.77 (0.18)
2016	1.80 (0.24)	1.24 (0.20)	1.67 (0.19)
2017	1.21 (0.11)	1.09 (0.11)	1.72 (0.13)
2018	1.73 (0.08)	1.34 (0.12)	2.18 (0.12)
2019	1.57 (0.07)	1.57 (0.10)	1.09 (0.08)
2020	0.97 (0.13)	0.87 (0.10)	1.36 (0.17)

Table 5.13: Mean number of participants in security issues per year



## 5.11 Other general findings

The goal of this section is to describe a few general findings that were made mainly during the manual analysis for this study and are unrelated to any other specific topic.

### 5.11.1 Role of bots

Bots are a common tool used in collaborative software engineering. The GitHub Marketplace<sup>1</sup> currently lists 35 bot applications that can be directly installed into any GitHub project. During the manual analysis of 333 security issues we were able to identify three main areas where bots were used. In four projects a “stale bot” was used to label issues as stale when there was no activity on them for a certain number of days and automatically close them after a further waiting period. In two projects a bot was used to import older issue threads from a different previously used issue tracking system into GitHub in order to consolidate all the history about the project in one place. Finally, in two projects a bot was used to automatically “ping” a certain user group based on a label that was assigned to the issue in order to attract their attention (e.g. a group of senior projects members specialised in security questions were automatically notified when the label “security” was assigned to an issue).

### 5.11.2 Tools to detect security issues

During our manual analysis we watched out for any tools that developers mention in their security issue reports that have helped them to identify a security issue. We were not able to identify any specific tools, but we found that in some cases, projects used GitHub’s Security Advisory feature<sup>2</sup>. GitHub Security Advisories builds upon the foundation of the Common Vulnerabilities and Exposures (CVE) list. Project developers can be notified if their project uses a dependency that has a known vulnerability. A new feature is currently in a closed beta phase that allows periodic scanning of the source code to detect known vulnerability patterns. Furthermore, GitHub is a CVE Numbering Authority (CNA) and is authorised to assign CVE identification numbers if users create a GitHub Security Advisory for a vulnerability in their own project, that does not yet have a CVE identifier.

### 5.11.3 Rejected security issues

Even though we did not have a feature in our manual analysis for the reason a security issue gets rejected, we did notice some recurring patterns. The most common reason for the rejection of security issues was that an issue had been resolved when another related

---

<sup>1</sup><https://github.com/marketplace?query=bot>

<sup>2</sup><https://github.com/advisories>

or unrelated issue was fixed or because an external dependency was updated. We also found cases where an issue was declared as a duplicate issue. Finally, there were a few cases where the issue was not reproducible, or it was found that issue did not concern this project but rather a third-party product or dependency.

#### 5.11.4 CVE entries

During our manual analysis of 333 security issues we found four security issues that directly led to the creation of a CVE entry. The first one, CVE-2019-11405, was created because the OpenAPI Tools OpenAPI Generator<sup>3</sup> used `http://` instead of `https://` in various Gradle build files, which may have caused insecurely resolved dependencies. The other three security issues all occurred within the FasterXML jackson-databind<sup>4</sup> project. They led to a series of CVE entries being created. Two vulnerabilities would allow an attacker to perform remote code execution (RCE) attacks (CVE-2018-14718 and CVE-2018-14719), one would allow external XML entity (XXE) attacks (CVE-2018-14720), another one server-side request forgery (SSRF) attacks (CVE-2018-14721) and the final one would allow exfiltration of content (CVE-2018-11307).

#### 5.11.5 Unique issues

We came across a few security issues that are quite special in some aspect. In two cases a security vulnerability was not disclosed publicly (“responsible disclosure”), but an issue was simply created to get a contact email address where the details could be submitted to. In one other case, a financial reward of unknown amount was paid directly to the reporter of a security vulnerability by a core member of the project. Furthermore, a group of researchers from the University of Nebraska reported a security issue to a project in one instance. Finally, we saw two occasions where a used dependency was upgraded more than one year after it became public, that the used version of the dependency contained a vulnerability, which even received a CVE identifier.

---

<sup>3</sup><https://github.com/OpenAPITools/openapi-generator>

<sup>4</sup><https://github.com/FasterXML/jackson-databind>

# 6

## Discussion

In this chapter we discuss our results from the previous chapter and connect different findings in order to create a coherent picture of security issues in Open Source Software.

As expected, we were able to measure in section 5.1 that security issues only make up a very small part of all issues in Open Source Software projects (1.42%). Only 2.4% of all developers in our analysed dataset are reporting security issues, and only 3.2% are involved in the discussion of security issues. Nevertheless, we were able to identify significant differences and similarities between security issues and non-security issues:

- Security issues are more often reported by core members or established contributors to the project and less often by outsiders than non-security issues.
- Initial reaction time for security issues is faster than for non-security issues.
- Discussions proceed slower in security issues than in non-security issues.
- Generally, discussions of security and non-security issues involve a similar number of participants.
- Discussions of security issues are shorter (in terms of the number of comments) than discussions of non-security issues.
- Discussions of security issues include fewer comments by the issue reporter than discussions of non-security issues.
- Security issues are resolved significantly slower than non-security issues.

- Long pending issues without comments are rejected faster in security issues than in non-security issues.

A possible explanation for the differences might be the complexity and urgency of security issues. Due to the severe consequences they can have, security issues are in general rather urgent compared to non-security issues and addressing or reporting them requires specialised knowledge that only a limited number of developers possess. This might explain why security issues are mostly reported by developers associated with the project. Additionally, if a matter is very complex, then the discussion might be progressing more slowly, there might be fewer comments and the issue can take longer to be resolved. Short initial reaction time could be explained by the urgency that comes with security issues.

Moreover, we found that in 50% of all analysed projects, it takes 656 days or more until the first security issue is created. This nearly two-year time span may be explained by the setup and initial development of a project that is required until a security issue can even emerge and by the fact that some projects do not have issue tracking enabled right from the beginning or do not enforce strict labelling right from the beginning. Once the first security issue of a project is created, a new one appears on average every 54 days.

Furthermore, we were able to measure a medium correlation between the activity of a project (in terms of the number of commits) and the number of security issues in a project and a strong correlation between the popularity of a project (in terms of the number of forks and stars) and the number of security issues in a project. This finding would lead to the hypothesis that the more developers are involved in a project, the more security issues (of all existing security problems) are discovered.

Our findings further support the hypothesis of the scarcity of security knowledge. In section 5.6 we measured that security issues have a low mean of only 2.87 comments and 1.51 involved distinct participants per issue in their discussions. Furthermore, all security issues are discussed and, as mentioned in the introductory paragraph to this chapter, also reported by a very small circle of developers.

In section 5.2 we also analysed the gender of issue reporters. We found that the percentage of female issue reporters is slightly higher for security issues than for non-security issues, but there is no significant difference in the acceptance rates between male and female security issue reporters.

During our manual analysis we compared many features between security issues with different statuses, and we were able to identify significant differences between accepted and rejected security issues.

- Accepted security issues are more often clearly explained than rejected security issues.
- Accepted security issues have a faster initial reaction and faster proceeding discussions than rejected security issues.

- Reporters of accepted security issues more often provide further information than reporters of rejected security issues. Notably accepted security issues more often contain or refer to documentation and reproducibility information than rejected security issues.
- The discussions of accepted security issues have a significantly higher proportion of thematically relevant comments (elaboration, ask/request or instructions) than discussions of rejected security issues, where generic comments (notifications, reminders and other) are more common.
- Accepted security issues more often have an assignee than rejected security issues.
- Accepted security issues are resolved faster than rejected security issues.

These findings constitute significant differences between accepted and rejected security issues. It remains to be analysed by future studies if those factors have a causal influence on the acceptance rate of security issues. At this point we can also mention a few more findings. In section 5.6 we found that pending security issues have the slowest reaction time among all issue statuses. Unsurprisingly, reminders are most common in pending issues. Also, notifications nearly make up for more than 50% of the comments in rejected security issues. Furthermore, elaboration in discussions is only slightly more confident in accepted security issues than it is in rejected security issues.

We found that among all issue statuses, comments in the discussion of security issues do not contain a hyperlink or further information in eight of ten comments. There are no notable differences between the issue statuses when it comes to the type of further information either. Mostly, further documentation is provided, followed by proposed fixes. No differences were also observed with the mentioning feature of GitHub. Developers do not mention another user in seven of ten comments. There are no notable differences between the issue statuses when it comes to the association with the project of the person who is mentioned. Members are most commonly mentioned, followed by contributors and the author of the issue himself. Finally, the issue reporter does not participate in the follow-up discussion at all in four of ten security issues.

In section 5.8 we also analysed pull requests that are associated with security issues. We found that two-thirds of all accepted security issues use the pull request feature of GitHub (and other functionality that comes with it like code review, build pipelines, etc.), while one-third of all accepted security issues had their code changes directly committed to the repository. On a positive note, we found that nearly 85% of the analysed pull requests were reviewed by another developer.

Interestingly, and this was quite unexpected, the issue reporter of a security issue is not involved at all in the pull request in 50% of the cases, neither as creator nor as a reviewer or discussion participant of the pull request. This could be partially explained if the security issue is reported by an external person who is not otherwise involved

in the project because pull requests and therefore code changes are mostly handled by established contributors or core members of a project. This assumption was confirmed by a quick background check. The number of comments in the issue discussion and the difficulty of the pull request discussion suggest that those issues were presumably relatively easy to fix and did not even require the reporter of the issue to be involved in the pull request.

When analysing the number of comments and distinct participants in pull request discussions, we found that the numbers are significantly higher than the equivalent numbers for the security issue discussions. A potential explanation that can be supported by observations made during the analysis is that the code changes that are discussed and reviewed in pull requests, often cause a lot of “going back and forth” if the matter is complex or hard to fix. This causes a high number of comments in the pull request discussion.

Furthermore, we also did correlation analysis during our manual analysis. We found that the number of comments in the discussion of a security issue has a statistically significant correlation of medium effect with the number of comments in the pull request discussion. This is consistent with the intuition that a hard or complex security issue that requires more discussion also leads to more discussion during the implementation phase (pull request) because the changes are likely more complex as well.

We made several investigations concerning the assignment feature of GitHub and reported our results in section 5.7. Most importantly, we could observe that the acceptance rate of security issues with an assignee is more than 30% higher than for security issues without an assignee. Rather interestingly two-thirds of all assignments in security issues are self-assignments, where a developer assigns himself to a security issue. Also, in one-fourth of the cases the assigned developer was actually the reporter of the security issue itself. We found that the group of developers who is assigned to security issues plays an important role in resolving security issues. Those security assignees were involved in roughly 64% of all security issues, either by commenting and reporting. When they reported security issues, they had an acceptance rate of 77.7%, which is higher than the average for security issues. Their resolution time was significantly shorter than the global mean as well. Furthermore, we found that security assignees outperform nearly 97% of the other developers involved in security issues, based on the number of comments and reported issues.

In section 5.9 we analysed the relationship between the resolution time and other characteristics of security issues. We were able to make the following significant observations:

- There is a strong positive correlation between the reaction time and the resolution time of security issues.
- There is a weak positive correlation between the number of comments and the resolution time of security issues.

- There is a weak positive correlation between the number of participants and the resolution time of security issues.
- There is a weak positive correlation between the number of participants in the pull request and the resolution time of security issues.
- The resolution time of security issues is longer if further documentation is provided in the comments.
- The resolution time of security issues is shorter if a CVE report is referenced in the issue report or in the comments.
- The resolution time of security issues is shorter if the issue reporter is a core member of the project.
- The resolution time of security issues is shorter if the issue reporter is involved in the pull request.
- The resolution time of security issues is shorter if a pull request is not reviewed or reviewed only once, versus multiple reviews.

These findings constitute significant differences in resolution time for categorical features or significant correlation of resolution time with other numerical features. It remains to be analysed by future studies if those factors have a causal influence on the resolution time of security issues. Interestingly, there was no significant difference between the mean resolution time for security issues with and without an assignee.

When comparing security issues of different age or resolution time in section 5.9 we found that for very short-lived and very old issues, there are more rejected than accepted security issues. For issues that are between two days and two months old, on the other hand, there are more than twice as many accepted security issues than pending security issues. Generally, we can say that the number of comments and participants increases as security issues age, but the numbers also show that if a security issue is not resolved after half a year, fewer participants are joining the discussion.

In section 5.10 we provided an overview of security issues that were reported from 2014 until 2020. Most importantly, we found a big increase in security issues in the years 2018 and 2019. We were also able to show that the number of participants in security issues over the years is rather constant, while there is a visible decrease in the number of comments and the resolution time of security issues over time; however, a certain grouping bias certainly applies for those analyses.

Finally, we were able to report some general or special findings in our last results section 5.11. Not only did we observe frequent usage of bots in security issues, but we did also observe a few unique issues that involved responsible disclosure, financial reward for reporting security issues and researchers pointing out security issues to the open

source community. Furthermore, we observed that security issues are mostly rejected because they are duplicate issues or they have been resolved elsewhere. Finally, we were lucky enough to find security issues in our manual analysis that led to the creation of new CVE reports.



# 7

## Threats to validity

Even though all the research that was carried out for this thesis was done to the best of our knowledge and with extreme caution, there are still possible threats to the validity of our results that we would like to report transparently in this chapter.

### **7.1 External validity**

Threats to external validity are related to the generalizability of our findings. A major threat to validity is the sampling bias of our project selection, which is described in section 4.1. Especially the restrictions to Java projects and projects with English issue tracking, which have been imposed for logistical reasons, might bias the results. Further bias might have been introduced during the sampling of security issues for the manual analysis. Even though we tried to pick a sample as representative as possible, while still getting insights into the wide variety of our population using the stratified-random approach, the results might have looked different with another sample. Furthermore, we have to keep in mind that software engineering projects are of highly dynamic nature, and the data for our study represents a snapshot of a very specific point in time. The projects will likely evolve in the future, and the results could look different if the study were to be carried out in the future, especially with the open source community rapidly growing at the moment.

## 7.2 Internal validity

Threats to internal validity relate to experimenter bias and errors. There is a threat to validity concerning the classification of issues into security issues and non-security issues. As described in the methodology section 4.2, we used a simple label-based approach to do the classification. It was clear that we will not be able to capture all security issues with this approach and multiple studies have proposed new ways using machine learning that provide more accuracy at higher costs [10, 22, 26]. It should be noted positively though, that we did not find any false-positive security issues among the 333 security issues we analysed during our manual analysis.

Another threat to internal validity could be the assignment of the issue statuses (accepted, pending and rejected). It is possible that despite our efforts to automatically detect all code changes we missed some, which would result in misclassification of actually accepted issues. Sometimes such code changes are poorly linked to the issue or only mentioned in a comment. To mitigate this threat, we manually checked all rejected issues in the manual analysis for missed code changes.

Finally, personal bias is also a threat to validity. We tried to describe our methodology, principles and criteria as transparently as possible and adhere to them in order to minimise the effects of our personal opinions or expectation of results. Furthermore, we tried to mitigate subjective influence during the manual analysis by conducting a pilot study previous to the real study, where results were discussed and agreed upon between the experimenters.

# 8

## Conclusion and future work

### 8.1 Conclusion

In this thesis we present our results of an explorative study on security issues on GitHub in order to answer the two research questions posed in the introduction.

We were able to show that the prevalence of security issues is low. We described how security issues evolve in the areas of reporters, reports, reaction, resolution, discussion, assignment and associated pull requests. Furthermore, we presented trends in security issues over the course of their lifespan and in recent years. We were able to identify multiple characteristics of security issues and their discussions that differentiate them from non-security issues, but also similarities between those two classes of issues. Additionally, we compared security issues that were accepted, pending and rejected security issues and found possible factors that may lead to their respective outcomes. Finally, we compared the resolution time of security issues for different characteristics and were able to report significant differences.

We believe our work is a substantial contribution to the open source community and will be useful for both researchers and practitioners in understanding and further improving the security culture in Open Source Software engineering.

### 8.2 Future work

This project is mainly of explorative nature. We were able to describe certain characteristics of security issues and identify similarities and differences among them, but

a lot of future work is still required to fully understand the reasons, explanations and causal connections behind these findings. The differences between security issues and non-security issues we listed in the discussion need further investigation to be fully explained. It would also be desirable if future research can confirm the possible factors of security issue acceptance and security issue resolution time we propose and use them to devise further recommendations to the open source community. Furthermore, it could be interesting to see if researches can leverage our findings to build prediction models that predict if a security issue gets accepted or rejected and how long it will take to resolve it.

Future studies on security issues in Open Source Software could widen the scope by looking at issues from other programming languages than just Java or even including other popular source code hosting and issue tracking systems than GitHub. Furthermore, future studies could make use of advanced methods to classify security issues using machine learning. One example would be the FARSEC approach proposed by Peters et al. in 2017 [22], which was proven to outperform labelling based classification used in this study, which is simply because not all projects and developers are consequent with labelling security issues strictly and errors in manual labelling also happen. Even more recently a new framework called LTRWES was proposed that even outperforms FARSEC [10]. Those are just two examples of how future studies could improve the classification of security issues and therefore use a dataset of greater quantity and quality for analysis.

In terms of the content of the analysis, it would be very interesting to see future works that leverage more or other metadata features of security issues than we did. For example, one could analyse the number of words in security issue reports or the use of source code snippets. Furthermore, it would be interesting to analyse if there are characteristics of different types of security issues, e.g. code injection vs. data breach vulnerabilities. Additionally, the content of the comments in the discussion or the security issue report itself could be taken into consideration by using natural language processing. That way one could possibly make statements about the informativeness of security issue reports, create a list of keywords that identify security issues or describe a relationship between the issue description and commit messages. Another dimension that could prove worthwhile to explore would be the difficulty of a fix for a security issue, especially when combined with existing metrics like the time it takes to resolve a security issue.

### **8.3 Acknowledgement**

I want to thank Dr. Mohammad Ghafari and Prof. Dr. Oscar Nierstrasz for supervising this thesis. Additionally, I would like to thank Lars Galliker for proofreading and his valuable inputs. Finally, I am really grateful to my family for the outstanding support they provided me with over the course of this project.

# Bibliography

- [1] Sebastian Baltes and Paul Ralph. Sampling in software engineering research: A critical review and guidelines, February 2020.
- [2] Marco Carvalho, Jared DeMott, Richard Ford, and David A. Wheeler. Heartbleed 101. *IEEE Security & Privacy*, 12(4):63–67, July 2014.
- [3] Indu Chawla and Sandeep K. Singh. Automatic bug labeling using semantic information from LSI. In *2014 Seventh International Conference on Contemporary Computing (IC3)*, pages 376–381, Los Alamitos, CA, USA, August 2014. IEEE.
- [4] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Routledge, Cambridge, MA, USA, May 2013.
- [5] Valerio Cosentino, Javier L. Canovas Izquierdo, and Jordi Cabot. A systematic mapping study of software development with GitHub. *IEEE Access*, 5:7173–7192, 2017.
- [6] Michael Gegick, Pete Rotella, and Tao Xie. Identifying security bug reports via text mining: An industrial case study. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pages 11–20, Los Alamitos, CA, USA, May 2010. IEEE.
- [7] Katerina Goseva-Popstojanova and Jacob Tyo. Identification of security related bug reports via text mining using supervised and unsupervised classification. In *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 344–355, Los Alamitos, CA, USA, July 2018. IEEE, IEEE.
- [8] Georgios Gousios and Diomidis Spinellis. GHTorrent: Github's data from a firehose. In Michael W. Godfrey and Jim Whitehead, editors, *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 12–21, New York, NY, USA, June 2012. IEEE.
- [9] Sameera Horawalavithana, Abhishek Bhattacharjee, Renhao Liu, Nazim Choudhury, Lawrence O. Hall, and Adriana Iamnitchi. Mentions of security vulnerabilities

- on reddit, twitter and GitHub. In *IEEE/WIC/ACM International Conference on Web Intelligence on - WI '19*, pages 200–207, Los Alamitos, CA, USA, October 2019. ACM Press.
- [10] Yuan Jiang, Pengcheng Lu, Xiaohong Su, and Tiantian Wang. LTRWES: A new framework for security bug report detection. *Information and Software Technology*, 124:106314, August 2020.
- [11] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. The promises and perils of mining GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014*, pages 92–101, New York, NY, USA, 2014. ACM Press.
- [12] David Kavalier, Sasha Sirovica, Vincent Hellendoorn, Raul Aranovich, and Vladimir Filkov. Perceived language complexity in GitHub issue discussions and their effect on issue resolution. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 72–83, Los Alamitos, CA, USA, October 2017. IEEE.
- [13] Pavneet Singh Kochhar, Eirini Kalliamvakou, Nachiappan Nagappan, Thomas Zimmermann, and Christian Bird. Moving from closed to open source: Observations from six transitioned projects to GitHub. *IEEE Transactions on Software Engineering*, 46:1–1, 2019.
- [14] Robert V. Krejcie and Daryle W. Morgan. Determining sample size for research activities. *Educational and Psychological Measurement*, 30(3):607–610, September 1970.
- [15] Jeff Luszcz. Apache struts 2: how technical and development gaps caused the equifax breach. *Network Security*, 2018(1):5–8, January 2018.
- [16] Benjamin S. Meyers, Nuthan Munaiah, Andrew Meneely, and Emily Prud'hommeaux. Pragmatic characteristics of security conversations: An exploratory linguistic analysis. In *2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 79–82, Los Alamitos, CA, USA, May 2019. IEEE.
- [17] Patrick Morrison, Tosin Daniel Oyetoyan, and Laurie Williams. Identifying security issues in software development: Are keywords enough? In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings, ICSE '18*, pages 426–427, New York, NY, USA, 2018. Association for Computing Machinery.

- [18] Patrick J. Morrison, Rahul Pandita, Xusheng Xiao, Ram Chillarege, and Laurie Williams. Are vulnerabilities discovered and resolved like other defects? *Empirical Softw. Engg.*, 23(3):1383–1421, June 2018.
- [19] Dongliang Mu, Alejandro Cuevas, Limin Yang, Hang Hu, Xinyu Xing, Bing Mao, and Gang Wang. Understanding the reproducibility of crowd-reported security vulnerabilities. In *Proceedings of the 27th USENIX Conference on Security Symposium, SEC'18*, pages 919–936, USA, 2018. USENIX Association.
- [20] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. Curating GitHub for engineered software projects. *Empirical Software Engineering*, 22(6):3219–3253, April 2017.
- [21] Naresh Kumar Nagwani and Ashok Bhansali. A data mining model to predict software bug complexity using bug estimation and clustering. In *2010 International Conference on Recent Trends in Information, Telecommunication and Computing*, volume 1, pages 13–17, Los Alamitos, CA, USA, March 2010. IEEE.
- [22] Fayola Peters, Thein Than Tun, Yijun Yu, and Bashar Nuseibeh. Text filtering and ranking for security bug report prediction. *IEEE Transactions on Software Engineering*, 45(6):615–631, June 2019.
- [23] Daniel Pletea, Bogdan Vasilescu, and Alexander Serebrenik. Security and emotion: Sentiment analysis of security discussions on github. In *Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014*, pages 348–351, New York, NY, USA, 2014. Association for Computing Machinery.
- [24] Mohammad Masudur Rahman and Chanchal K. Roy. An insight into the pull requests of GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014*, pages 364–367, New York, NY, USA, 2014. ACM Press.
- [25] Lucía Santamaría and Helena Mihaljević. Comparison and benchmark of name-to-gender inference services. *PeerJ Computer Science*, 4:e156, July 2018.
- [26] Rui Shu, Tianpei Xia, Laurie Williams, and Tim Menzies. Better security bug report classification via hyperparameter optimization, May 2019.
- [27] Marcel Steinbeck. Mining version control systems and issue trackers with LibVCS4j. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 647–651, Los Alamitos, CA, USA, February 2020. IEEE.

- [28] Jason Tsay, Laura Dabbish, and James Herbsleb. Influence of social and technical factors for evaluating contribution in GitHub. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 356–366, New York, NY, USA, 2014. ACM Press.
- [29] Jason Tsay, Laura Dabbish, and James Herbsleb. Let’s talk about it: Evaluating contributions through discussion in github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 144–154, New York, NY, USA, 2014. ACM Press.
- [30] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. Quality and productivity outcomes relating to continuous integration in github. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 805–816, New York, NY, USA, 2015. ACM Press.
- [31] Xinda Wang, Kun Sun, Archer Batcheller, and Sushil Jajodia. Detecting ”0-day” vulnerability: An empirical study of secret security patch in OSS. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 485–492, Los Alamitos, CA, USA, June 2019. IEEE.
- [32] Xiaoxue Wu, Wei Zheng, Xiang Chen, Fang Wang, and Dejun Mu. CVE-assisted large-scale security bug report dataset construction method. *Journal of Systems and Software*, 160:110456, February 2020.
- [33] Xin-Li Yang, David Lo, Xin Xia, Zhi-Yuan Wan, and Jian-Ling Sun. What security questions do developers ask? a large-scale study of stack overflow posts. *Journal of Computer Science and Technology*, 31(5):910–924, September 2016.
- [34] Awad Younis, Yashwant Malaiya, Charles Anderson, and Indrajit Ray. To fear or not to fear that is the question: Code characteristics of a vulnerable function with an existing exploit. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, CODASPY ’16, pages 97–104, New York, NY, USA, 2016. Association for Computing Machinery.
- [35] Yue Yu, Huaimin Wang, Vladimir Filkov, Premkumar Devanbu, and Bogdan Vasilescu. Wait for it: Determinants of pull request evaluation latency on GitHub. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, MSR ’15, pages 367–371, New York, NY, USA, May 2015. IEEE.
- [36] Mansooreh Zahedi, Muhammad Ali Babar, and Christoph Treude. An empirical study of security issues posted in open source projects. In Tung Bui, editor, *51st Hawaii International Conference on System Sciences, HICSS 2018, Hilton*



*Waikoloa Village, Hawaii, USA, January 3-6, 2018*, pages 1–10, Red Hook, NY, 2018. ScholarSpace / AIS Electronic Library (AISeL).

- [37] Feng Zhang, Foutse Khomh, Ying Zou, and Ahmed E. Hassan. An empirical study on factors impacting bug fixing time. In *2012 19th Working Conference on Reverse Engineering, WCRE '12*, pages 225–234, USA, October 2012. IEEE.
- [38] Mengyuan Zhang, Xavier de Carne de Carnavalet, Lingyu Wang, and Ahmed Ragab. Large-scale empirical study of important features indicative of discovered vulnerabilities to assess application security. *IEEE Transactions on Information Forensics and Security*, 14(9):2315–2330, September 2019.



# Anleitung zum wissenschaftlichen Arbeiten

In this appendix we describe the process we followed in order to obtain the data for our study from GitHub. Future projects can use this as a reference in order to facilitate the data collection process. The provided source code and described principles should be easily adaptable for other APIs or data sources.

## A.1 GitHub GraphQL API

The main source for the data of our study was the GitHub GraphQL API v4<sup>1</sup>. GraphQL is an Open Source data query and manipulation language for APIs. It was originally developed by Facebook but now hosted by the non-profit Linux Foundation.

The GitHub GraphQL API is free for use by any registered GitHub user after one has obtained an API key. Like any public API, GitHub enforces a rate limit to prevent excessive or abusive calls to their servers. This rate limit is based on a scoring system. All API calls by the same user must not exceed 5000 cumulative points in a one hour period. The details of the rate limitations and very detailed documentation can be found on the website in the footnote. Furthermore, GitHub provides an “explorer”, where queries can be tested directly in a web browser.

---

<sup>1</sup><https://developer.github.com/v4/>

## A.2 Environment

As we did not perform any serious computation tasks, no special hardware was required. A fast and stable Internet connection is highly recommended though because some downloading tasks can span over multiple hours.

We performed our downloading tasks on a Lenovo T470s with an Intel Core i7-7600U CPU at 2.80 GHz and 20 GB of RAM. Our operating system was Windows 10, but macOS or a Linux distribution would work equally well.

Our downloading scripts were written in Python. We chose Python because of personal preference and because there are many useful libraries available for Python that can be used for data analysis and scientific computations like *NumPy*<sup>2</sup>, *matplotlib*<sup>3</sup> and *pandas*<sup>4</sup>. We did not use a dedicated IDE but a browser-based, interactive computational environment called *Jupyter Notebooks*<sup>5</sup>. A Jupyter Notebook consists of “cells” that contain Markdown, executable Python source code or Python output and therefore allow you to document your process very well. A typical screenshot of the Jupyter interface can be seen in figure A.1.

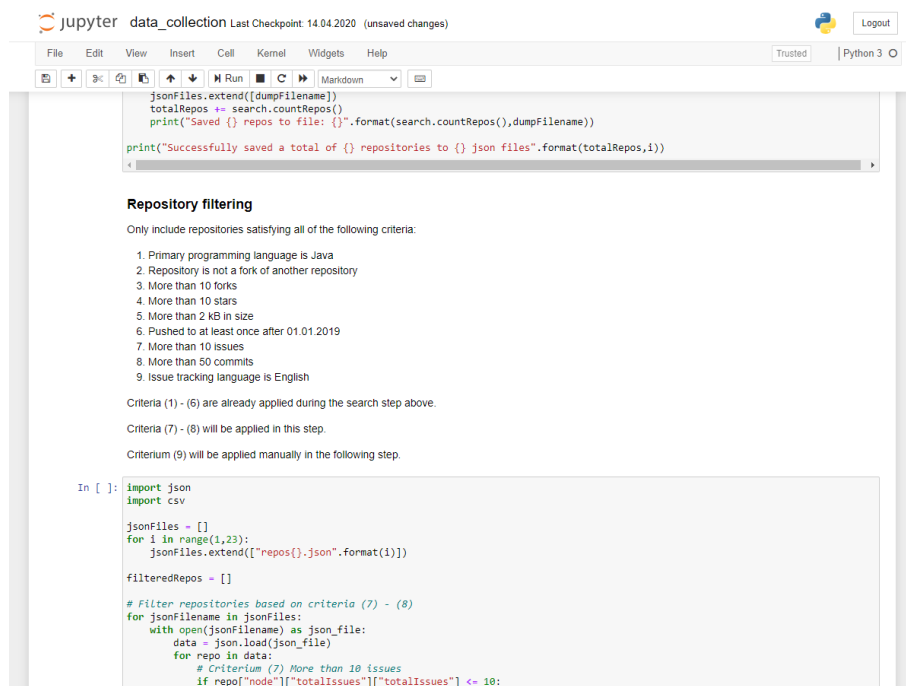


Figure A.1: Typical Jupyter Notebook

<sup>2</sup><https://numpy.org/>

<sup>3</sup><https://matplotlib.org/>

<sup>4</sup><https://pandas.pydata.org/>

<sup>5</sup><https://jupyter.org/>

## A.3 Basics

For sending basic queries to the GitHub API that do not require any pagination, we used the function shown in listing 1. As a parameter it takes a GraphQL query string, performs the API call and returns the result in JSON. We saved all responses from the API into .json files that we could later parse into CSV files and use with any data analysis tool.

```

1 import requests
  import json

  API_KEY = ""
5  BASE_URL = "https://api.github.com/graphql"

  # Simple function to perform basic query on the GitHub GraphQL API
  def run_query(query):
    errorMsg = "Query failed to run by returning code of {}. {}".format(API_KEY, query)
10    headers = {"Authorization": "Bearer {}".format(API_KEY)}
    tries = 10
    for i in range(tries):
        request = requests.post(BASE_URL, json={'query': query},
        ↪ headers=headers)
        if request.status_code == 200:
15            return request.json()
        else:
            if i < tries:
                continue
            raise Exception(errorMsg.format(request.status_code,
            ↪ query))

```

Listing 1: Simple query function

For performing more sophisticated queries that may require pagination because they result in long lists of objects or subobjects (e.g. a list of more than 100 issues), we used the class shown in listing 2 and created a subclass from it for every use case separately. The request itself is performed in the *generator* function, while the *iterator* function is implemented in the subclasses to loop through multiple pages of results when pagination was used.

For simplicity reasons we will not show the implementation of the subclasses in this appendix, but only the GraphQL queries that were used for the different use cases.

```

1 import requests
  import json

  # Base class to perform complex queries on the GitHub GraphQL API
5  class GitHubGraphQLQuery:
    BASE_URL = "https://api.github.com/graphql"
    def __init__(
        self,

```

```
10     github_token=None,
        query=None,
        query_params=None,
        additional_headers=None
    ):
15         self.github_token = github_token
        self.query = query
        self.query_params = query_params
        self.additional_headers = additional_headers or dict()

    @property
20     def headers(self):
        default_headers = dict(
            Authorization=f"token {self.github_token}",
        )
        return {
25             **default_headers,
            **self.additional_headers
        }

    def generator(self):
30         while True:
            try:
                yield requests.request(
                    'post',
                    GitHubGraphQLQuery.BASE_URL,
                    headers=self.headers,
                    json=dict(query=self.query.format_map(self.
↪ query_params))
                ).json()
            except requests.exceptions.HTTPError as http_err:
                raise http_err
            except Exception as err:
40                 raise err

    def iterator(self):
        pass
```

Listing 2: Base class for complex queries

When performing hundreds of API calls over multiple hours, we had to keep in mind the rate limits of the API. As the number of remaining “points” and the time of the next reset was included in every response, we implemented an easy logic that was called after every API call. The source code is shown in listing 3. When the remaining “points” fall under a certain threshold, the program simply pauses and resumes at the reset time provided in the response of the API.

```
1 # Check rate limits
  remaining_rate_limit = result["data"]["rateLimit"]["remaining"]
  print("Remaining rate limit - {}".format(remaining_rate_limit))

5 resetAtUTC = datetime.datetime.strptime(result["data"]["rateLimit"]["
  ↪ resetAt"], "%Y-%m-%dT%H:%M:%SZ")
  nowUTC = datetime.datetime.utcnow()
  if(int(remaining_rate_limit) < 15):
    print("Rate limit nearly exceeded, waiting for {} seconds until
    ↪ reset".format((resetAtUTC-nowUTC).seconds))
    time.sleep((resetAtUTC-nowUTC).seconds)
```

Listing 3: Rate limit check

## A.4 Searching repositories

For searching the repositories on GitHub, we used the GraphQL query shown in listing 4. The variables *\$searchString* and *\$pointer* are populated by the surrounding python code.

```
1 {
  search(query: "$queryString", type: REPOSITORY, first: 50, after:
  ↪ $pointer) {
    pageInfo {
      hasNextPage,
5      endCursor
    }
    repositoryCount
    edges {
      node {
10        ... on Repository {
          id
          name
          url
          description
          createdAt
          pushedAt
          releases {releases: totalCount}
          forks: forkCount
          pullRequests {pullRequests: totalCount}
          stargazers {stars: totalCount}
          totalIssues: issues {totalIssues: totalCount}
          master: object(expression:"master") {
            ... on Commit {
              history {
25                commits: totalCount
              }
            }
          }
        }
      }
    }
  }
}
```

```
    }
    labels(first: 10, query: "security") { nodes {name} }
30    primaryLanguage {primaryLanguage: name}
    languages(first: 3) { nodes {name} }
  }
}
}
35 }
}
```

Listing 4: Search query

The first line indicates that we want to use the “search” query type with four parameters. We want to have all the repositories that match the search string paginated into responses with 50 repositories. The pointer for the parameter *after* is provided with the response of the API and can be used to get the next 50 repositories with the next API call. In accordance with our criteria, we used the following search string:

```
stars:>10 forks:>10 size:>2048 created:$timeFrame
fork:false pushed:>2019-01-01 language:java
```

The filtering for more than ten issues and 50 commits per repository had to be done locally later because the “search” query type does not support those criteria. We had to separately query the repositories for every half-year-interval since the creation of GitHub in 2009 because even with pagination, the search would not return the full results for search queries that produce more than 1000 results. We did so by populating the *\$timeFrame* variable with the desired time frame (e.g. “2016-01-01..2016-06-30”).

As can be seen in the lower parts of listing 4 from line 10 to line 31, the APIs response will include all necessary details we need about repositories, especially labels that contain the word “security”, if any of those exist.

## A.5 Downloading issues and comments

After we locally filtered and selected 182 repositories, we had to download all the issues and corresponding comments from those repositories. From the previous step we had a list with the ids of all repositories. Such an id uniquely identifies a node in the GitHub GraphQL data model (in this case a repository object). We could now use our list of ids with the “node” query type, as shown in listing 5. The *\$repoIds* variable is populated with the array containing all 182 repository ids. This query uses nested pagination because we need all the issues of every repository and all the comments on every issue. We are navigating between the pages using the variables *\$issuePointer* and *\$commentPointer* the same way as described in the previous section.

In the first part of listing 5 between line 6 and 34, the details requested about every issue are specified (title, author, state, etc.). In the middle part between line 35 and

52, we specify the details of the request for the comments of every issue (creation date, author, author association, etc.). In the final part from line 53 on, a list of timeline items for every issue is requested. We wanted to have all timeline items with type *REFERENCED\_EVENT*, *CROSS\_REFERENCED\_EVENT* and *ASSIGNED\_EVENT* with the details that belong to them. We did this, because we later wanted figure out, at what point in time an issue was assigned to whom, and if an issue was referenced anywhere else on GitHub, e.g. in commit messages, in other issues or in pull requests.

```

1 {
  node(id: $repoIds) {
    ... on Repository {
      id
      name
5     issues(first: 10, after: $issuePointer) {
      pageInfo {
        hasNextPage
        endCursor
10    }
      totalCount
      nodes {
        id
        title
        number
15       author {
          ... on User {
            id
            name
20          }
        }
        bodyText
        authorAssociation
        createdAt
25       closedAt
        url
        state
        assignees(first: 30) {
          totalCount
30          nodes {
            id
            name
          }
        }
      }
35     comments(first: 50, after: $commentPointer) {
      pageInfo {
        hasNextPage
        endCursor
40    }
      totalCount

```



```
nodes {
  createdAt
  author {
    ... on User {
45       id
        name
      }
    }
  authorAssociation
50  bodyText
}
}
timelineItems(first: 50, itemTypes:
  [REFERENCED_EVENT,
55  CROSS_REFERENCED_EVENT,
  ASSIGNED_EVENT]) {
  totalCount
  nodes {
60    __typename
    ... on ReferencedEvent {
      id
      createdAt
      commit {
65        id
          oid
          url
        }
      isCrossRepository
    }
70    ... on CrossReferencedEvent {
      id
      createdAt
      willCloseTarget
      source {
75        __typename
        ... on Issue {
          id
          url
        }
        ... on PullRequest {
80          id
          url
        }
      }
    }
85  }
  ... on AssignedEvent {
    id
    createdAt
    actor {
```

```
90         ... on User {
          id
          name
        }
      }
95     assignee {
      ... on User {
        id
        name
      }
100   }
    }
105  }
  }
}
```

Listing 5: Issue and comment query