Mittuniversitetet
MID SWEDEN UNIVERSITY

Evolving Trends in the Adoption and Effectiveness of DEPENDABOT Security Pull Requests

Jacob Jerneståł

Final Project
Main field of study: Computer Engineering BA (C)
Credits: 15
Semester/year: Spring 2024
Supervisor: Rodi Jolak
Examiner: Felix Dobslaw
Course code/registration number: DT133G
Programme: Software Engineering

Evolving Trends in the Adoption and Effectiveness of DEPENDABOT Security Pull Requests

Jacob Jernestål
Department of Communication, Quality Management and Information Systems
Mid Sweden University
Östersund, Sweden
{jaje1300}@student.miun.se

***Abstract*—In the rapidly evolving software industry, bots have become integral to automating tasks and enhancing developer productivity and are revolutionizing the way security patches are implemented in software projects. Our study investigates the impact of DEPENDABOT on the speed and efficacy of security patching in GitHub Open Source Software projects, by studying merge times and factors that contribute to DEPENDABOT's resolution of security issues in JavaScript projects. We use a dataset containing DEPENDABOT Security Pull Requests. Our study validates previous findings by collecting data from the GitHub API and publishing a dataset collected between 2021 and 2024. We face challenges with collecting features impacting merge times, but overcome them by prioritizing the top 3 features and 2 additional ones. We also investigate the factors behind not merging Pull Requests to identify the obstacles in adopting DEPENDABOT's recommendations, by analysing Pull Request comments. We start performing sentiment analysis and topic modeling but switch to GitHub Copilot instead and continue investigating presence of factors impacting rapid merge times. Our results present a lower adoption rate of DEPENDABOT Security Pull Requests in JavaScript Open Source Software projects, specifically 13%, compared to those of the original study. 76% of Pull Requests are merged within 4 days, with a median decision time of 0,3 days. The main reason for not merging a DEPENDABOT Security Pull Requests is that another DEPENDABOT Security Pull Request supersedes it. Factors associated with faster merge are related to smaller changes and, controversially, disabling auto merge.**

***Index Terms*—GitHub, Dependabot, Security, Rapid Merge Time, Copilot**

ACKNOWLEDGEMENT

Some individuals go out of their way to support others. During the project period Academic mentor Sonya Edström clarifies academic writing rules and shares time-management strategies anchored in neurodivergent studies on keeping a sustainable pace. Course examiner Felix Dobslaw finds an additional research tutor and gives statistics-based recommendations for the baseline focus factor required to succeed. Course supervisor Rodi Jolak provides weekly assistance via video and textual reviews. Research tutor César Soto Valero provides motivational concepts, practical examples and fresh perspectives on the subject of research by sharing experience within related fields. The artificial intelligence extension Github Copilot provides pair-programming company. My fiancée Louise Andersson maintains our physical health and keeps ourselves grounded by introducing fun exercise breaks into our routine but also telling me to get back to studying when I lose focus. Our adorable cats Ciri & Tiger provide adorable distractions while fighting for supreme control of our computer keyboard, until eventually receiving a keyboard of their own. And last but not least, we will always remember our dear friend Petronella Björk who burned with a "righteous anger" that fuelled those around her until the very end.

I. INTRODUCTION

Security vulnerabilities in software systems pose a significant threat to societal infrastructure, with the potential to compromise sensitive data and disrupt critical services. The prevalence of such vulnerabilities has escalated with the complexity of modern software, making it imperative to address these security gaps promptly. In 2021 alone, over 18,000 new vulnerabilities were reported, underscoring the urgency for effective security measures [1]. Open Source Software (OSS) projects often rely on third-party dependencies to expedite the release of new features. However, managing these dependencies to keep them up-to-date is challenging, and any lapse exposes software projects to security risks [2]. These dependencies also have dependencies with themselves, creating a web of nested dependencies of which some are unstable, deprecated, or even unnecessary [3]. Various automated tools offer limited support to handle dependency updates, yet automated tools still require developers to perform manual updates frequently. A Continuous Integration and Delivery (CI/CD)-tool is able to perform these updates during deployment. However, dynamically altering dependencies between build and deployment goes against best practices, specifically "Enforcing the lockfile" as recommended by Wall *et al.* [4]. "Locking" the dependency versions ensures consistency, meaning every developer on the team and every environment the code runs on has an identical set of dependencies. Without a lockfile, new minor or patch updates of dependencies would be built whenever the install command is run, meaning that if a newly released update has security issues these are inadvertently introduced into the project [5].

Development bots (DevBots) like DEPENDABOT offer an automated response to vulnerable dependencies in a project's repository, suggesting patches to be manually reviewed or auto-merged [6]. When integrated with a CI Pull Request (PR), DEPENDABOT proactively responds to alerts raised for

vulnerable dependencies in a repository's dependency graph. It alerts on dependencies or malware, and suggests security updates or version updates to resolve these issues, providing a swift response to potential security vulnerabilities [7]. Optionally, the time to apply a fix is reduced further by automating the decision process and enabling automatic merging of PRs, meaning any suggestion from DEPENDABOT results in a 'swift auto-merge'. However, this process faces challenges in practical application due to varying levels of trust developers place in automated suggestions, and the trade-offs between swift auto-merging and thorough manual reviews. Swift auto-merging comes with a risk of introducing new errors, while manual reviews potentially slow down the patching process and expose the software to vulnerabilities for a prolonged period.

Alfadel *et al.* [8] previously shed light on these challenges. Their study provides empirical evidence for understanding developer issues when adopting DEPENDABOT by publishing a dataset [9] created between June 2017 and April 2020. They also developed a logistic regression model that identify factors explaining merge times. The authors investigated reducing attack-surface via automated features minimising the risk of applications being affected by external vulnerable dependencies. They wanted to understand tool usage and adoption-rate, to measure the effect on vulnerability mitigation. Their study investigated GitHub OSS JavaScript projects and started a month after the trial launch of DEPENDABOT. However, it is based on a previous version of DEPENDABOT, called DEPENDABOT-Preview. The preview has since been deprecated in favour of a GitHub-native version (differences detailed in Section II), and our understanding of DEPENDABOT's adoption, performance and the factors contributing to faster patch times remains limited, as mentioned by Nasrabi *et al.* [10]

In this study, we investigate GitHub-native DEPENDABOT's usage in OSS JavaScript projects, a context chosen for its accessibility and representation of practical software development. We investigate DEPENDABOT usage because of the potential impact software integration and automation has on maintenance and security. JavaScript has been one of the most used programming languages for the last decade according to Stack Overflow [11], and DEPENDABOT is the most visible bot on GitHub according to He *et al.* [12]. The current DEPENDABOT version is a native feature of GitHub and is fully integrated with other GitHub features, such as GitHub Actions. Other similar bots (e.g. "Renovate") are not native to GitHub, meaning there is more friction involved (e.g. additional costs and configuration needed) when installing the tool. More primitive tooling (e.g. "CodeCov") uses DEPENDABOT internally to focus on providing code coverage features instead [13].

While insights from this our study does not generalize to proprietary or closed-source software projects, or OSS projects using other programming languages, ecosystems or DevBots, our findings provide valuable insights about the utilization and effectiveness of DEPENDABOT in real-world scenarios, with potential implications for future research and practice. Areas to consider for future work are to expand the study to include other DevBots (like Snyk), programming languages and ecosystems, conduct a longitudinal study to track changes in PR handling over an extended period, or investigate the impact of DEPENDABOT use on software quality, as measured by metrics such as code coverage or code complexity. Investigating the use of Artifical Intelligence to evaluate changelogs and provide a recommended decision to merge or not for PRs that contain breaking changes is another path for future contribution.

II. PURPOSE AND CONTRIBUTIONS

According to Nasrabi *et al.* [10] the findings of Alfadel *et al.* [8] might not be accurate in reflecting developer reception and usage of the DEPENDABOT service provided by GitHub. There is a three-year gap between the time of the study by Alfadel *et al.* in 2021 and the time of our study in 2024, during which time DEPENDABOT migrated out of the "DEPENDABOT-preview" onto the "GitHub-native DEPENDABOT" [14]. There are key differences between the two services. For instance, the preview version is not directly integrated with GitHub's vulnerability alerting system, meaning it is unable to present all available security patches [15]. Furthermore DEPENDABOT-preview always suggests upgrading to the most recent non-vulnerable version, whereas GitHub-native DEPENDABOT suggests the minimum required version instead [16].

Our study aims to partially reproduce the research by Alfadel *et al.* [8], focusing on JavaScript GitHub projects that use the GitHub-native DEPENDABOT between April 2021 and March 2024. The goal is to validate the previous findings that are dependent on DEPENDABOT-preview, and extending the knowledge in the field by identifying trends and changes in DEPENDABOT's effectiveness over time. Additionally, the research explores developers' reasons behind non-merged PRs to uncover the challenges and barriers in adopting DEPENDABOT Security Pull Requests (DSPRs). Increased build times due to frequent pull requests, heightened notification noise, dependency conflicts, and potential incompatibility issues with the project environment are some of the pitfalls to consider. We aim to identify potential improvements to DEPENDABOT's approach leading to a higher adoption rate of DSPRs.

Our study investigates how the use of DEPENDABOT affects the speed and efficacy of security patching in JavaScript projects. We aim to understand how developers adopt DSPRs and identify factors that contribute to the tool's successful utilization for swift and effective resolution of security issues. There are trade-offs and risks associated with immediate auto-merging and prolonged manual review; the former inadvertently introduces unstable dependencies and breaking changes, while the latter leaves software exposed to vulnerabilities for a longer duration. Given the assumption that DEPENDABOT patches are correct and are to be trusted, we aim to identify the factors that contribute to faster patch times and thus fewer vulnerabilities in JavaScript projects using DSPRs.

To achieve these objectives, we conduct an empirical study to answer the following research questions:

- **RQ1: How often and how quickly are DSPRs merged?** This question will help us understand the general receptiveness to DSPRs and their life-cycle. Receptivity in this case refers to the adoption rate of accepted DSPRs.
- **RQ2: What are the reasons for DSPRs not being merged?** This question aims to uncover the challenges and barriers in adopting DSPRs.
- **RQ3: What factors are associated with rapid merge times?** This question will help us identify strategies for maximizing the effectiveness of DSPRs.
- **RQ4: How do our results compare to the original study?** This question will reveal any trends or changes in the use of DSPRs over time since the sunset of DEPENDABOT-Preview and the launch of GitHub-native DEPENDABOT, and over a different set of projects, as old projects have changed and new projects have been introduced.

Finally, we publish a GitHub repository containing the tool developed to pull down, transform and analyse data to produce our datasets (detailed in Section IV-E). The tool is available to use and open for contribution, and is designed to be easily modified for future studies. Our ambition is that the tool is used for continuous research and to shed light on potential uncertainties in our result.

III. BACKGROUND AND RELATED WORK

As the software development landscape continues to evolve more organizations are shifting from "DevOps" where security testing is a separate process at the end of software development [17], to "DevSecOps" where security testing is integrated into the software development process [18]. This transition requires a comprehensive understanding of how developers and security specialists are able to effectively collaborate with automated tools or "bots" to maintain security.

The adoption of DEPENDABOT and similar DevBots represents a shift in development practices towards more automated workflows. Understanding this transition yields insights into the evolving nature of software development and the role of automation tools in it, thereby contributing to the larger discourse in the software engineering field. Developers set the adoption rate by using or not using these tools, and their reasons are subjective as they are based on direct experience of the developer, their personal history, preference and prejudice. We are unable to rely on interviews to identify these reasons without subjecting ourselves to interviewer bias. However, developers leave a trail of data during software development in the form of commit history which we are able to extract and analyse without influencing their opinions.

Given this context, our study evaluates DEPENDABOT's performance in patching JavaScript OSS projects. By modeling merge times and analyzing the factors influencing them, this research contributes to a better understanding of how automated tools like DEPENDABOT is used more effectively to address security vulnerabilities in software development. This investigation is particularly timely given the ever-increasing reliance on OSS and the consequent risks posed by potential security vulnerabilities.

A. Related Work

Prior to the original study research in the domain of DSPRs was primarily focused on understanding the integration and management of security updates. Gousios *et al.* [19] and Rigby *et al.* [20] had laid the groundwork by examining the factors influencing merge times, providing a foundation for Alfadel *et al.* [8] to build upon. Their work highlighted the importance of rapid integration for security fixes and identified key factors that contribute to swift merging practices.

The findings of our comparative study serve as a bridge between the foundational research on DSPR integration and management and the broader literature recognizing the challenges in this domain. While earlier studies laid the groundwork for understanding the factors influencing merge times and the importance of rapid integration, our study indicates a shift in the landscape with a significant decrease in the adoption rate of DSPRs and changes in merging practices. This suggests a need for a reevaluation of current strategies and highlights the dynamic nature of DSPR management.

In the broader literature, there is a recognition of the challenges faced in managing DSPRs. Studies by Kula *et al.* [21] and Zerouali *et al.* [22] point out the slow response to security vulnerabilities, while Nasrabadi *et al.* [10] highlight cases where manual intervention is necessary, leading to extending merge times up to months. Furthermore, He *et al.* [12] discuss the ambivalent attitudes towards automated tools like DEPENDABOT, which, despite reducing technical lag, are met with skepticism by some developers who prefer manual oversight or prefer using DEPENDABOT solely for notifications. This ongoing discourse underscores the complexity of DSPR management and the need for continued research to optimize the process.

IV. RESEARCH METHODOLOGY

Our methodology is divided into collection of project and PR data, analysis of non-merged and merged PRs, and a comparative study. Our first three research questions aim to reproduce empirical evidence from the original study, while the fourth compares our results with those of the original study. While our project selection criteria is the same as that of the original study, the resulting sets of projects is different as newer projects have accumulated since the time of the original study. Our study is based on a dataset divided into three phases and split into three files: the initial set, a refined set, and a set with metadata.

The dataset is obtained and processed according to the following flow divided into six steps (as can be seen in Figure 3): 1) Collect DSPRs. 2) Select high-quality projects, filtering on refinement criteria. 3) Exclude open PRs. 4) Calculate time difference between creation and closure of PRs. 5) Perform in-depth manual analysis of closed PRs to identify reasons for

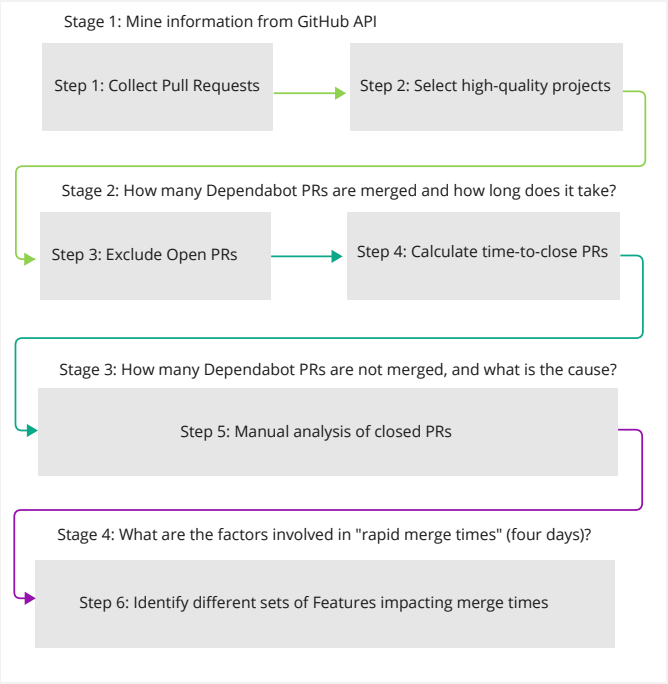

Fig. 1: Visualization of the methodology flow, representing using the GitHub API to answer three research questions by performing six steps divided into four stages.

not merging. 6) Identify different sets of features impacting merge times.

A. Collection of Project and PR data:

We use the GitHub API [1] to collect data on DSPRs in JavaScript projects that address security-related dependency vulnerabilities. The focus is on high-quality, non-forked projects that has more than 20 commits and have been starred by users, as recommended by Mirhosseini and Parnin [23], and Kalliamvakou *et al.* [24]. We exclude open PRs because they have no decision taken and are thus not valuable to analyse.

Some of the limitations of the GitHub API relate to rate limits and available REST resources, for instance: The GitHub API does not support searching for repositories based on the number of commits. This limitation does not directly restrict the ability to select repositories based on activity, as we are able to first collect a list of repositories and then fetch their respective commits. However, the list of repositories is capped at 1000. This limitation does not impact the size of our data set, because the amount of active repositories that match our query during the period between April 2021 to April 2024 is 728. Furthermore, the API does not directly flag PRs that patch a security-related dependency vulnerability. Some ways we mitigate these issues are: (i) filter repositories that have more than 500 stars and fork status set to false, (ii) structure queries and result pagination to accurately retrieve the data without breaching any limits, (iii) examine PR bodies for indications of a security patch. Our data collection period covers the GitHub-native DEPENDABOT's entire lifespan.

[1]https://docs.github.com/en/rest?apiVersion=2022-11-28

B. Analysis of Non-Merged PRs:

We extract discussion and review comments from each closed PR and qualitatively examine non-merged PRs based on the reasons provided by the maintainers. We manually categorize the reasons for not merging into the categories identified by Alfadel *et al.* (as shown in Table I). PRs that lack discussion or comment data entirely are excluded.

To mitigate the risk of author bias, we use natural language processing (NLP) techniques to analyze PR comments. Sentiment analysis helps gain an understanding of the attitudes, opinions, and emotions of the maintainers. Topic modeling helps identify key themes, which indicate common reasons for not merging PRs. We begin to preprocess the text data to remove noise and irrelevant information, such as the presence of industry jargon, abbreviations, or project-specific terminology. However, we find our approach to be overly time-consuming and ultimately not providing the value we anticipated. The preprocessing required and the inherent bias in selecting NLP techniques leads us to conclude that a less convoluted approach is more effective for our purpose. Instead, we opt for a more straightforward method, programmatically filtering comments by reason for non-merging based on examples from our own dataset. These are combined with additional examples provided by GitHub Copilot. By leveraging the vast amount of publicly available code it has learned from, including DEPENDABOT-related contexts, GitHub Copilot uses NLP to provide context-aware examples which remove the need for further preprocessing and modelling of our text data. This approach not only saves time but also minimizes the bias that may arise from selecting specific NLP techniques and any lapse in judgement or experience in applying them. Our strategy allow us to efficiently process and categorize DSPRs, and our solution remains easy to extend with additional categorization capabilities in case our time constraint should change. Additionally, we validate our results with manual reviews, using a sample of PRs. This approach provides a objective and systematic way to identify reasons for deciding not to merge.

C. Analysis of Merged PRs

We calculate the time difference in days between PR creation and closure, and examine the presence of specific features that impact merge time of DSPRs (as shown in Table II) and contribute to 'rapid merge time' which is classified as below four days by Alfadel *et al.* [8] [9]. Identifying and analysing all (13) features that could potentially impact this metric is a complex and time-consuming task. Due to time constraint we focus our approach by prioritizing the top 3 features that have the most significant influence on merge times according to Alfadel *et al.* [8]. Once the primary features are addressed, our strategy is further enhanced by selecting an additional 2 features that do not belong to the top performers of the original study, so that we also have a chance

TABLE I: Categorized reasons for not merging DSPRs, extracted from PR data.

Id	Reason	Description
R1	Superseded	The PR was superseded by another newer PR
R2	Up to date	The update was applied manually on the dependency file
R3	No longer a dependency	The affected dependency is removed and no longer exists in the project
R4	No longer updateable	Incompatible peer dependency requirement
R5	Tests	Test failures
R6	Errors	Error in DEPENDABOT due to incorrect implementation
R7	Quality Requirement	Uncompliant project standards for handling PRs
R8	Unknown	The PR cannot be classified due to insufficient information

TABLE II: Features impacting merge time, grouped into sets. Descriptions relate to snapshots taken at the time of PR creation. Features in bold are included in our study, and the remaining ones are excluded due to time constraint.

JSON key	Description	Feature set
"severity"	**Severity (Critical, High, Moderate, Low) of the vulnerability**	**Vulnerability patch features**
"patch_level"	**Patch level (Major, Minor, Patch) of the dependency update**	**Vulnerability patch features**
"changed_lines"	**Number of modified lines of code**	**PR features**
"auto_merge"	**Status (True or False) of auto-merge method**	**PR features**
"sloc"	Number of lines of code in the project	Project features
"team_size"	Number of active team members in the project	Project features
"num_submitted_PRs"	Number of submitted DSPRs to the project	Project features
"perc_accepted_PRs"	**Percentage of merged DSPRs in the project**	**Project features**
"num_dependencies"	Number of total project dependencies	Project features
"num_recent_commits"	Number of commits in the project during the previous month	Project features
"age"	Project age (in days)	Project features
"num_issues"	Number of total project issues	roject features
"num_PRs"	Number of total project PRs	Project features

to catch unexpected changes to previously under-performing features. This methodical approach ensures a comprehensive understanding while maintaining efficiency in the data analysis process. It allows for a more manageable and targeted analysis, leading to actionable insights that can effectively reduce merge times.

D. Comparative Study:

We contrast our findings with the results of the original research conducted by Alfadel *et al.* [8], to identify trends and shifts, and explore potential explanations for these findings. We compare Changes in PR acceptance rates, as they serve as indicators of the perceived effectiveness and trustworthiness of DEPENDABOT over time. For instance: An increase in PR acceptance rates could suggest improvements in DEPENDABOT's patch suggestion quality, heightened trust in automated security fixes, or more rigorous security practices over time. Conversely, a decrease in PR acceptance rates could signal potential issues with DEPENDABOT's recommendations, increasing complexity of security updates, or shifts in the project maintenance practices. If PR acceptance rates remain similar to those reported in the original study, it might indicate the persistence of the same factors.

E. Artifacts

Our study produces a dataset including spreadsheets divided into two packages; a reproduction package and an extended reproduction package. The first package contains DSPR data from our study, presented in the same format as the artifact produced by the original study to simplify comparisons between the two. The second package contains additional data from our study. The code to produce artifact dataset of our study can be found on Github [2]. The reproduction package and the extended reproduction package of **our study** can be found on GitHub too[3].

The **reproduction package** for artifact dataset of our study include:

- **Initial:** Total identified DSPRs, with columns "url", "title" and "body"
- **Refined:** Remaining DSPRs after project filtration, with additional columns "repository_url_short" and "pull_request_url".
- **Meta:** Refined DSPRs with metadata and additional columns "comments_url", "status" and "closed_by"

The **extended reproduction package** for artifact dataset of our study include:

- **Comments:** Discussions & reviews related to deciding not to merge DSPRs
- **Statistics:** Describing factors related to merge speed

The comments are stored as csv files and the statistics as json files (as can be seen in Appendix A). The reproduction package (e.g. initial, refined & meta) of the **original study** can be found on Zenodo [4].

V. RESULTS

We focus on the frequency and speed of merges, reasons for not merging, and factors influencing rapid merge times.

[2]https://github.com/jaje1300/dependabot-security-pull-requests-reader

[3]https://github.com/jaje1300/dependabot-security-pull-requests-reader/tree/main/src/data/manual-cache

[4]https://zenodo.org/records/4437290

Finally we compare the results with results from the original study.

RQ1: *How often and how quickly are DSPRs merged?*

- ***RQ1.1:*** *How long does it take to merge a security PR since it was first created?*
- ***RQ1.2:*** *How long does it take to close a security PR since it was first created?*

TABLE III: Amount of DSPRs merged or not.

Total	Merged	Not Merged
1185 (100%)	455 (38%)	730 (62%)

The results indicate that 38% of DSPRs are merged, while a significant majority of 62% are not (as shown in Table III). Median decision time to merge is 0,6 days and median decision time to not merge is 7 days.

RQ2: *What are the reasons for DSPRs not being merged?*

- ***RQ2.1:*** *What reason is most popular when not merging?*

TABLE IV: Reasons for DSPRs not being merged.

Id	Reason	Comment amount
R1	Superseded	1006 (49.41)%
R2	Up to date	208 (10.21)%
R3	No longer a dependency	0 (0.00)%
R4	No longer updateable	0 (0.00)%
R5	Tests	6 (0.29)%
R6	Errors	10 (0.49)%
R7	Quality Requirement	18 (0.88)%
R8	Unknown	788 (38.70)%

The predominant reason for not merging is that the requests are superseded, accounting for 49% of the cases (as shown in Table IV). Other reasons include being up to date or encountering errors, though these are less common. Note that the reason superseded refers to a decision taken by DEPENDABOT itself. Additionally, the average amount of comment per DSPR is 2.

A. ***RQ3:*** *What factors are associated with rapid merge times?*

- ***RQ3.1:*** *How many DSPRs were merged rapidly?*
- ***RQ3.2:*** *What factors had the biggest impact?*

TABLE V: Amount of DSPRs merged rapidly.

Total	Rapidly	Not Rapidly
455 (100%)	348 (76.48%)	107 (23.52%)

A majority of 76% of DSPRs are merged rapidly (as shown in Table V), and in 96% of those instances auto-merge is disabled. Furthermore, smaller changes have a higher likelihood of quick acceptance, such as those with fewer than 100 changed lines of code, peaking at 20 or 30. At the project level, the percentage of accepted pull requests before initiating a DSPR also plays a role, with higher acceptance rates correlating with rapid merges. Finally, the severity vulnerability eludes our search but the patch level addressed by the DSPR is a factor, with minor vulnerabilities being addressed more swiftly than major ones.

B. ***RQ4:*** *How do our results compare to the original study?*

- **RQ4.1:** Difference in adoption rate of DSPRs?
- **RQ4.2:** Difference in time to merge DSPRs?
- **RQ4.3:** Difference in time to not merge DSPRs?
- **RQ2.1:** Different reasons for not merging DSPRs?
- **RQ3.1:** Difference in time to rapidly merge DSPRs?
- **RQ3.2:** Different factors impact merge time of DSPRs?

TABLE VI: Our results compared to the original study. Rows that begin with "+" are related to merging DSPRs, while rows that begin with "-" are reasons for not merging a DSPR.

Result	Our study	Original study
Adoption rate	38 %	65 %
Rapidly merged	76 %	75 %
Time to merge	0.6 days	1 days
+Auto-merge enabled	3.1 %	65.7 %
Time to not merge	7 days	8 days
-Superseded	49.4 %	50.8 %
-Up to date	10.2 %	30.1 %
-No longer a dependency	0.0 %	6.6 %
-No longer updateable	0.0 %	6.4 %
-Tests	0.2 %	3.2 %
-Errors	0.4 %	1.4 %
-Quality Requirement	0.8 %	1.1 %
-Unknown	38.7 %	0.4 %

Our comparison reveals a significant decrease in the adoption rate of DSPRs, with the rate appearing to drop by 27% (as shown in Table VI). The median time for DSPRs to merge has decreased by half a day, while the time for DSPRs not to merge has decreased by an entire day. A higher percentage of DSPRs are not merged due to being "Superseded" by DEPENDABOT, and by taking that into account we get an actual adoption rate of 69%. We apply the same logic to the original study result and get an actual adoption rate of 82%, meaning that the **actual drop in adoption rate of DSPRs is 13%**. Furthermore, there is a significant increase in "Unknown" reasons for not merging, with complete reduction in reasons "No longer a dependency" and "No longer updateable", indicating that the latter two are included in one of the other categories due to a shift in behaviour of DEPENDABOT. Additionally, the trend of rapid merging is slightly increased and remains consistent despite a marked trend away from automatic merging of DSPRs, as indicated by "auto-merge" being set to true in 3% of cases, compared to 65% in the original study.

VI. DISCUSSION

The comparison between the results of our work and the original study reveals several intriguing trends and changes in the behavior of DSPRs. Building upon the foundational work of Gousios *et al.* [19], Rigby *et al.* [20], and Alfadel *et al.* [8], our study echoes the criticality of swift integration for

security patches and brings to light the evolving nature of DSPR adoption and management practices.

The integration of GitHub-native DEPENDABOT with GitHub's security features is anticipated to increase its adoption rate by offering more valuable suggestions that are easier for developers to trust and accept, compared to its predecessor DEPENDABOT Preview. However, our study indicates an actual decrease of 13% in the adoption rate of DSPRs. This can be partially explained by considering the use of the "auto-merge" feature. Automatic merging of PRs is expected to be enabled as it means the PR will be merged immediately after the build succeeds, without any human interaction. However, in 96% of cases "auto-merge" is disabled. Based on the assumption that a manual review process takes longer compared to an automatic review process, and the fact that we only fetch closed PRs that are merged or not, we believe that there are currently more PRs lingering in an open state pending decision. Future studies should take this angle into consideration when investigating if a drop in adoption rate impacts the speed of patching security vulnerabilities, as it suggests a shift in how these pull requests are being handled and indicates a need to reassess current methodologies. This aligns with the broader literature that acknowledges the multifaceted challenges in DSPR management, such as the slow response to security vulnerabilities and the necessity for manual intervention in certain cases, as highlighted by Kula *et al.* [21], Zerouali *et al.* [22], and Nasrabadi *et al.* [10].

Interestingly, the median time for DSPRs to merge has been reduced by half a day, suggesting an improvement in processing efficiency. Conversely, the time for DSPRs not to merge has also decreased by a full day. This could be attributed to a higher incidence of DSPRs being superseded, accounting for half of the unmerged DSPRs. Another reason for not merging DSPRs show notable differences; the 'Unknown' category has increased dramatically from 0.4% to 38%, indicating a potential area for further investigation to understand what is contributing to this uncertainty. The mixed reactions to automated tools like DEPENDABOT reflect the ongoing debate and the complexity inherent in DSPR management, as discussed by He *et al.* [12]. Our study contributes to this discourse by providing fresh insights and reinforcing the necessity for ongoing research to refine and enhance DSPR management strategies. The original study published their dataset but not the tooling used to produce it, making it challenging to reproduce their study. Therefore, we publish both of our datasets and the tool developed to produce them, in an effort to enable simple reproducibility and extension of our study.

Overall, these comparisons highlight that while some aspects remain consistent, such as the high percentage of rapidly merged DSPRs, there are areas where the management of DSPRs has evolved. This could reflect changes in the practices of software maintenance or the evolution of DEPENDABOT algorithms and policies. Our research provides a foundation for further exploration into the dynamics of automated security updates and their integration into software development workflows. It's essential to continue monitoring these trends to improve the efficiency and security of software maintenance practices.

In terms of rapid merge times, our study shows that the percentage of rapidly merged DSPRs remains consistent, but there is a significant change in the factors influencing merge time, highlighting a decrease in auto-merging practices. The 'patch' and 'minor' updates also seem to play a significant role in our findings. This comparative analysis suggests that while the speed of merging DSPRs has not changed markedly, the decision-making process and the factors influencing rapid merges have shifted. We recommend that future research investigate and compare different time periods, to narrow down when different trends start and stop.

A. Ethical Considerations & Limitations

The ethical implications of our findings are multifaceted and warrant careful consideration. The decrease in the merge rate of DSPRs raises questions about the fairness and transparency of the review process, potentially indicating a need for a more inclusive process. To address concerns regarding fairness, broader review participation can be encouraged but carries with it the risk of increased decision time. Furthermore, the dramatic increase in the 'Unknown' category for unmerged DSPRs suggests a lack of clarity and communication regarding the decision-making process. This opacity could potentially undermine trust in the system, indicating a need for clear guidelines and accountability in the management of DSPRs to uphold the integrity and security of software maintenance practices. Additionally, the shift away from auto-merging practices reflects a move towards more human-centric decision-making, which, while potentially beneficial, must be balanced against the efficiency and objectivity that automated systems can provide. It is imperative that the criteria for merging or not merging DSPRs are applied consistently and justly across all contributions without slowing down the time to merge, in order to be vulnerable for as short a time as possible.

In addressing the threats to validity in the analysis of factors influencing merge times, we strategically prioritize the examination of the top 3 features identified by Alfadel *et al.* [8] as having the most significant impact. This focused approach not only streamlines the research process in the face of time constraints but also ensures a thorough understanding of the critical elements affecting DSPR merge times. By subsequently incorporating an additional 2 features into the analysis, we enhance our study's methodological rigor, enabling a targeted and efficient investigation that yields practical insights to expedite merge times. Our layered analysis serves to mitigate potential biases and confers a structured framework for assessing the multitude of variables at play.

Our study, while comprehensive, is not without other limitations. The dataset used does not fully capture the nuances of DSPR behaviors across different projects and ecosystems. Additionally, the observational nature of our research limits our ability to establish causality between observed trends and underlying factors. However, our methodology benefits from a robust comparative approach, allowing for a detailed analysis

of trends over time. Furthermore, the significant decrease in merge rates and the shift in factors influencing DSPR management provide valuable insights into the evolving practices of software maintenance. These findings underscore the dynamic nature of DSPR handling and highlight the need for continuous adaptation in automated systems like DEPENDABOT. Future studies could expand on this work by incorporating a broader data set and employing methods that can better ascertain causal relationships.

VII. CONCLUSIONS

Our findings highlighted a pivotal shift in the management of DSPRs, signaling a departure from previous patterns. The decline in DSPR merge rates, coupled with a rise in unexplained non-merges, pointed to emerging complexities within the process that warrant closer examination. This trend suggests that the criteria or the decision-making framework for merging DSPRs has undergone a change, possibly reflecting a broader evolution in software maintenance protocols or the operational algorithms of tools like DEPENDABOT.

The stability of rapid merges amidst these changes indicated that while certain aspects of the process remain unchanged, the factors influencing the decision to merge have become more nuanced. The move towards increased manual scrutiny over automated merges, especially for minor updates and patches, could be indicative of a growing emphasis on quality control and risk management in software development.

This transformation could have been driven by several factors, including heightened awareness of security vulnerabilities, the need for more rigorous testing procedures, or a strategic shift in organizational priorities towards more sustainable development practices. The increased 'Unknown' category in non-merges presents an area ripe for further research, as understanding these unclassified cases could reveal insights into the evolving landscape of software maintenance.

Our research underscored the dynamic nature of software development, where automated tools like DEPENDABOT need to be adapted to the changing demands of security, efficiency, and reliability. Continuous monitoring of these trends is essential to refine these tools and practices, ensuring they align with the industry's progression and maintain the integrity of software systems. For this purpose we publish a tool capable of mining and analysing data from GitHub by inputting parameters for time period, author and programming language, enabling future studies to drill into remaining uncertainties surrounding the behaviour of DEPENDABOT and developers.

Future research endeavors are recommended to focus on dissecting the "Unknown" reasons for non-merges, examining the temporal shifts in DSPR management, and exploring the interplay between automated tools and human oversight. Such investigations will be crucial in fortifying the resilience of software systems against an ever-evolving technological environment. Our comparative study serves as a call to action for the industry to remain vigilant and proactive in adapting to these shifts, to safeguard the cornerstone of modern digital infrastructure; reliable and secure Open Source Software.

A. Main findings & contributions

- Differences in the behaviour and use of DEPENDABOT combined with increase in Unknown factors present areas ripe for future research of the domain
- Tool to use for the collection and analysis of GitHub PRs
- Recommendations for practitioners and organizations to optimize their vulnerability patching process

B. Practical takeaways

- Disabling auto-merge does not impact the speed to merge, thus practitioners do not need to be afraid of incorporating breaking changes by mistake
- Smaller code changes are merged faster, below 100 lines
- Minor patches are merged faster than major patches

REFERENCES

[1] Redscan, "Nist nvd analysis - record vulnerabilities in 2021," *The Redscan*, Dec 2021. [Online]. Available: https://www.redscan.com/news/nist-nvd-analysis-2021-record-vulnerabilities/

[2] R. G. Kula, D. M. German, A. Ouni, T. Ishio, and K. Inoue, "Do developers update their library dependencies? an empirical study on the impact of security advisories on library migration," *Empirical Software Engineering*, vol. 23, pp. 384–417, 2018. [Online]. Available: https://doi.org/10.1007/s10664-017-9521-5

[3] C. Soto-Valero, N. Harrand, M. Monperrus, and B. Baudry, "A comprehensive study of bloated dependencies in the maven ecosystem," *Empirical Software Engineering*, vol. 26, no. 3, p. 45, 2021. [Online]. Available: https://doi.org/10.1007/s10664-020-09914-8

[4] K. W. CheatSheets Series, S. Heigh, J. Manico, and J. Maćkowski, "Npm security best practices¶," *NPM Security - OWASP Cheat Sheet Series*, 2021. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/NPM_Security_Cheat_Sheet.html#2-enforce-the-lockfile

[5] M. M. A. Kabir, Y. Wang, D. Yao, and N. Meng, "How do developers follow security-relevant best practices when using npm packages?" in *2022 IEEE Secure Development Conference (SecDev)*. IEEE, 2022, pp. 77–83. [Online]. Available: https://doi.org/10.1109/SecDev53368.2022.00027

[6] GitHub, "Managing pull requests for dependency updates," *GitHub Docs*, 2024. [Online]. Available: https://docs.github.com/en/code-security/dependabot/working-with-dependabot/managing-pull-requests-for-dependency-updates#about-dependabot-pull-requests

[7] GitHub, "About dependabot security updates," *GitHub Docs*, 2024. [Online]. Available: https://docs.github.com/en/code-security/dependabot/dependabot-security-updates/about-dependabot-security-updates

[8] M. Alfadel, D. E. Costa, E. Shihab, and M. Mkhallalati, "On the use of dependabot security pull requests," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, 2021, pp. 254–265. [Online]. Available: https://doi.org/10.1109/MSR52588.2021.00037

[9] M. Alfadel, "On the use of dependabot security pull requests," *Zenodo*, Jan. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.4437290

[10] H. Mohayeji Nasrabadi, A. Agaronian, E. Constantinou, N. Zannone, and A. Serebrenik, "Investigating the resolution of vulnerable dependencies with dependabot security updates," in *Mining Software Repositories conference*, Mar. 2023. [Online]. Available: https://doi.org/10.1109/MSR59073.2023.00042

[11] Stack Overflow, "Stack overflow developer survey 2022," *Stack Overflow*, 2022. [Online]. Available: https://survey.stackoverflow.co/2022/#overview

[12] R. He, H. He, Y. Zhang, and M. Zhou, "Automating dependency updates in practice: An exploratory study on github dependabot," 2022. [Online]. Available: https://doi.org/10.1109/TSE.2023.3278129

[13] J. Holland, "From vulnerability alerts to mitigation: Our code security strategy with codecov and dependabot," *Codecov*, Mar 2023. [Online]. Available: https://about.codecov.io/blog/from-vulnerability-alerts-to-mitigation-our-code-security-strategy-with-codecov-and-d

[14] M. McDonald, "Goodbye dependabot preview, hello dependabot!" *The GitHub Blog*, Apr 2021. [Online]. Available: https://github.blog/2021-04-29-goodbye-dependabot-preview-hello-dependabot/

[15] L. Dohm, "Dependabot vs. dependabot-preview - detecting security vulnerabilities · community · discussion 22520," *GitHub Community*, Oct 2019. [Online]. Available: https://github.com/orgs/community/discussions/22520

[16] C. Soto-Valero, T. Durieux, and B. Baudry, "A longitudinal analysis of bloated java dependencies," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 1021–1031. [Online]. Available: https://doi.org/10.1145/3468264.3468589

[17] Amazon Web Services, "What is devops? - devops models explained," *Amazon Web Services*, 2024. [Online]. Available: https://aws.amazon.com/devops/what-is-devops/

[18] Amazon, "What is devsecops? - developer security operations explained," *Amazon Web Services*, 2024. [Online]. Available: https://aws.amazon.com/what-is/devsecops/

[19] G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 345–355. [Online]. Available: https://doi.org/10.1145/2568225.2568260

[20] P. C. Rigby and C. Bird, "Convergent contemporary software peer review practices," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013. New York, NY, USA: Association for Computing Machinery, 2013, p. 202–212. [Online]. Available: https://doi.org/10.1145/2491411.2491444

[21] R. Kula, D. German, A. Ouni, T. Ishio, and K. Inoue, "Do developers update their library dependencies?" *Empirical Software Engineering*, vol. 23, pp. 1–34, 02 2018. [Online]. Available: https://doi.org/10.1007/s10664-017-9521-5

[22] A. Zerouali, E. Constantinou, T. Mens, G. Robles, and J. Gonzalez-Barahona, "An empirical analysis of technical lag in npm package dependencies," 04 2018. [Online]. Available: https://doi.org/10.1007/978-3-319-90421-4_6

[23] S. Mirhosseini and C. Parnin, "Can automated pull requests encourage software developers to upgrade out-of-date dependencies?" in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2017, pp. 84–94. [Online]. Available: https://doi.org/10.1109/ASE.2017.8115621

[24] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining github," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 92–101. [Online]. Available: https://doi.org/10.1145/2597073.2597074

APPENDIX

APPENDIX 1: TIME PLAN

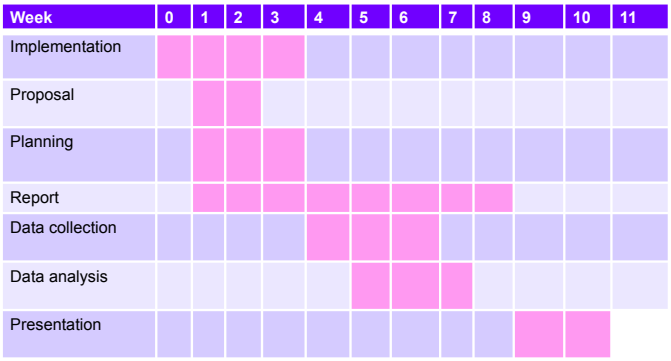

Fig. 2: Initial time plan for our study.

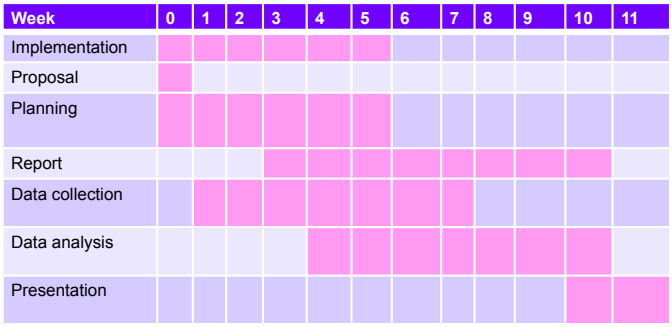

Fig. 3: Actual time plan of our study.

APPENDIX 2: ARTIFACT EXAMPLES

A. Extended reproduction package JSON snippets

RQ1-time-to-decide.json:

```
[
  {
    ...
    "percentages": {
      "merged": 38.39,
      "notMerged": 61.60
    },
    "timeInDays": {
      "merged": {
        "median": 0.64,
      },
      "notMerged": {
        "median": 7.00,
      }
    }
  }
]
```

RQ2-reasons-for-not-merging.json:

```
[
  {
    ...
    "percentages": {
      "R1_Superseded": 49.41,
      "R2_Up_to_date": 10.21,
      "R3_No_longer_a_dependency": 0.00,
      "R4_No_longer_updatable": 0.00,
      "R5_Tests": 0.29,
      "R6_Errors": 0.49,
      "R7_Quality_Requirement": 0.88,
      "R8_Unknown": 38.70
    }
  }
]
```

RQ3-rapid-merge-times.json:

```
[
  {
    ...
    "percentages": {
      "rapid": 76.48,
      "notRapid": 23.51
    }
  }
]
```

RQ3-rapid-merge-factors-pull-request.json:

```
[
  {
    ...
    "percentages": {
      "autoMergeTrue": 3.16,
      "autoMergeFalse": 96.83,
      "below": {
        "changedLines_2": 11.78,
        "changedLines_50": 67.81,
        "changedLines_100": 79.02
      }
    }
  }
]
```

RQ3-rapid-merge-factors-vulnerability.json:

```
[
  {
    ...
    "percentages": {
      "unknown": 9.77,
      "patch": 36.78,
      "minor": 42.24,
      "major": 11.20
    }
  }
]
```

RQ3-rapid-merge-factors-project.json:

```
[
  {
    ...
    "percentages": {
      "percAccepted": 87.35,
      "percAcceptedPrs_0": 12.64,
      "below": {
        "percAccepted_10": 12.64,
        "percAccepted_50": 29.88,
        "percAccepted_90": 68.96
      }
    }
  }
]
```