"I wasn't sure if this is indeed a security risk": Data-driven Understanding of Security Issue Reporting in GitHub Repositories of Open Source npm Packages

Rajdeep Ghosh
IIT Kharagpur
ghoshrajdeep2000@gmail.com

Shiladitya De
IIT Kharagpur
shiladityade.bwn2001@gmail.com

Mainack Mondal
IIT Kharagpur
mainack@cse.iitkgp.ac.in

Abstract

The npm (Node Package Manager) ecosystem is the most important package manager for JavaScript development with millions of users. Consequently, a plethora of earlier work investigated how vulnerability reporting, patch propagation, and in general detection as well as resolution of security issues in such ecosystems can be facilitated. However, understanding the ground reality of security-related issue reporting by users (and bots) in npm–along with the associated challenges–has been relatively less explored at scale.

In this work, we bridge this gap by collecting 10,907,467 issues reported across GitHub repositories of 45,466 diverse npm packages. We found that the tags associated with these issues indicate the existence of only 0.13% security-related issues. However, our approach of manual analysis followed by developing high-accuracy machine learning models identify 1,617,738 security-related issues which are not tagged as security-related (14.8% of all issues) as well as 4,461,934 comments made on these issues. We found that the bots which are in wide use today might not be sufficient for either detecting or offering assistance with these issues. Furthermore, our analysis of user-developer interaction data hints that many user-reported security issues might not be addressed by developers—they are not tagged as security-related issues and might be closed without valid justification. Consequently, a correlation analysis hints that the developers quickly handle security issues with known solutions (e.g., corresponding to CVE, or with a suggested solution). However, security issues without such known solutions (even with reproducible code) might not be resolved, hinting at a need for better-automated assistance for npm developers to address security issues. Our findings offer actionable insights for improving security management in open-source ecosystems, highlighting the need for smarter tools and better collaboration. The data and code for this work is available at https://doi.org/10.5281/zenodo.15614029

1 Introduction

Node Package Manager or *npm* [79] is a widely used ecosystem to manage and distribute JavaScript software packages (i.e., libraries). Serving as a central hub for open-source development, it powers diverse applications and services. The npm ecosystem is defined by its dense interdependencies, where packages build upon others, forming a highly interconnected network. However, with increasing interdependency, the risk of propagation of security vulnerabilities increases. Nevertheless, as an open-source platform, npm benefits from a collaborative community, which plays a crucial role in identifying, addressing, and mitigating these vulnerabilities.

Previous research has extensively investigated security vulnerability management systems in prominent package ecosystems such as npm, PyPI, and RubyGems. They focused on vulnerability reporting, propagation, and resolution [45, 46, 82, 100, 102]. Upstream vulnerabilities, patch delivery delays, and the significance of regular dependency updates all represent considerable risks, according to research [65, 100]. Furthermore, the prior work investigated the function of bots in automating processes, increasing productivity, and supporting open-source software (OSS) workflows, with varied results regarding their usefulness and limitations in security management [32, 78, 96].

However, majority of these previous works focus on proposing the *automated* security vulnerability management tools and techniques. They often overlook what techniques are *actually* used in real world. Prior work also did not shed light on whether public (i.e., open source) aspect of ecosystems like npm actually helps in detecting security flaws and mitigating them. Specifically, it is not clear whether users or bots even report/suggest mitigation of security issues in these ecosystems and if those reports have any impact on improving the security of the npm packages. Answering these quite important unanswered questions will help the npm community (both developers and users) evaluate the effectiveness of currently deployed mechanisms for reporting and addressing security issues and identify areas for improvement.

In this work, for the first time, we answer these questions via large-scale user-generated data collection and analysis. We focus on publicly available npm packages and their GitHub repositories (since GitHub provides security issues and comments for these npm packages). By default, issues (both security and non-security), on public GitHub repositories are publicly visible [88, 95]. While features like Private Vulnerability Reporting (PVR) enable private disclosure, they are relatively recent and are not enabled by default [59]. Thus, it is not yet widely adopted.

We systematically chose 45,466 npm packages with different numbers of *dependents* (i.e., how many other packages they are used by —a measure of popularity and impact). For these packages, we collected 37,278 GitHub repositories, and from these repositories, we collected 10,907,467 issues raised in GitHub. Interestingly, only 0.13% of these issues were tagged by the GitHub repository owners as security-related. However, by leveraging machine learning, we found a significant 14.8% of user-reported security issues which are not tagged with any security-related tags. Overall, we collected and analyzed 1,617,738 security-related issues (as well as 4,461,934 comments on those issues) for these repositories. Furthermore, we leverage this data to analyze (i) how each security-related issue is handled during the creation, discussion, and resolution phase (ii) what are the functionalities of bots which are used today in GitHub repositories of npm packages. Overall, we took a deep dive into this large-scale data of security-related issues in npm packages—our in-depth mixed-method analysis (complemented by creating novel machine learning-based detection models) of these issues uncovered a hierarchy of themes for interaction between users/developers while handling these issues as well as factors that affected the resolution of such issues. Specifically, in this work, we answer four key research questions.

RQ1 *How prevalent is reporting security-related issues in GitHub repositories of npm packages?*

RQ2 *How effective are bots in detecting and addressing security-related issues?*

RQ3 *How do the users interact with the GitHub repository maintainers for npm packages while reporting security-related issues?*

RQ4 *What factors correlate with the resolution of security-related issues?*

While investigating these questions, we found that across npm packages with varying numbers of dependents, around 10%–15% of issues are security-related. In fact, more than 23% of the issues did not receive any comments (Section 6). Interestingly, although bots in these repositories help to report security issues, their effectiveness is rather limited, as hinted by the fact that less than 0.1% security issues are tagged by bots in our data. In fact, our in-depth systematic analysis reveals that there is a general lack of security-focused bot usage which leverage techniques like static analysis or machine learning (Section 7). In fact, the number of such bots are also quite limited in our dataset. Moreover, this concerning situation is also present within user-developer interactions. The user-reported security-related issues are often ignored by developers and can be closed without a valid justification (since it becomes stale) (Section 8). Our correlation analysis identifies a potential reason—only when the user reported issues contain a potential fix (via pull request) or include a CVE —publicly accepted set of already reported security vulnerabilities (CVE) and weaknesses (CWE), the npm developers resolve this issue (unless otherwise specified, we treat CVE and CWE references equivalently throughout the paper). Otherwise, even when reproducible code for a security-issue is included in a reported issue it is more likely to become stale without any resolution. Our findings hint at the need for improving bots as well as resolution techniques for reported security issues in repositories of npm packages.

Limitations: Due to computational constraints and API rate limits, collecting GitHub links for millions of npm packages was challenging. As a trade-off, we followed a practical and coverage-driven approach using dependent-based stratified sampling. Since the number of dependents indicates the potential impact of security issues, we included all high-impact packages (> 100 dependents) and large samples of low-impact packages (Table- 1). Thus, although we might have missed some packages, security implications for those missing packages might be limited. Our approach has the potential to miss security issues which are not reported to GitHub. However, we already identified a significant number of reported security issues (which are not tagged with security-related tags), and this shortcoming can only increase the number of such issues. Thus, our results are potentially lower bound on actual security-related issues. One possible limitation is that some context-specific tags may not be captured by our Word2Vec based methodology. However, our final machine learning model to detect security issues was trained (and used) on the full issue descriptions (both title and body). Thus, we ensure that security-related issues are robustly classified, even when tags are missing or applied inconsistently. Furthermore, some inaccuracy is associated with machine learning models used to identify potential security-related issues and related themes. We attempted to reduce this concern by hyperparameter adjustment, rigorous model selection, and manual validation in line with best practices. Furthermore, our thematic saturation attained during thematic analysis suggests that there will be a very small number of such unidentified themes after the application of our machine learning models over millions of issues and comments, if any, which may have a minor impact on the validity of the presented results. In summary, we uncovered interesting and potentially generalizable factors affecting the resolution of security-related issues using our ecologically valid dataset despite these limitations.

2 Related Work

We review the related work along four broad dimensions: security vulnerability management in package ecosystems, analysis of bots in developer workflows, interaction with developers/maintainers and detection of malicious packages in repositories.

Security vulnerability management in package ecosystems: Previous work studied popular package ecosystems, e.g., npm, PyPI, RubyGems. Many of these studies have focused on how security vulnerabilities in one package make its dependent packages vulnerable [45, 46, 82, 100, 102]. Alfadel et al. [37] revealed that many Node.js applications rely on packages with undisclosed vulnerabilities, emphasizing the need for faster remediation. Similarly, findings from other works on npm, RubyGems and Golang ecosystems stress getting timely updates and patches about vulnerabilities in the dependencies of a package [65, 100, 102]. The study by Bühlmann et al. [40] examines developer responses to security issues in Java repositories, while our study, in contrast, collected and analyzed security concerns raised by the users themselves via issues reported in GitHub repositories of open source npm packages often involving high-accuracy ML models . These community-reported security concerns from the real world, unlike previous work, are often explicitly not connected to the known vulnerabilities detected in the dependencies (e.g., via CVE).

Analyzing bots in developer workflows: Bots (programs to automate tasks) have been extensively studied for their ability to automate tasks, enhance productivity, and support decision-making in various domains, including open-source software (OSS) [71, 89]. On platforms like GitHub, bots are frequently employed for tasks such as continuous integration, dependency management, and collaborative modeling [32, 33, 78]. Prior studies have investigated their impact on developer workflows, revealing alterations in commit activity and pull request closure times following bot implementation. Wessel et al. found that out of 351 GitHub projects, 26% utilised bots and a recent follow-up study even introduced bots to address negative impacts on contributions [96, 98]. However, none of these works considered either the diverse functions that bots play in a real-world ecosystem like GitHub repositories of npm packages or the roles bots played in addressing security issues within OSS (specifically, npm). For the first time, this work bridges this gap and investigates the effectiveness of these bots in identifying and addressing security vulnerabilities. To do so, we built on another line of research—bot detection. Methods like BotHunter and BIMAN leveraged machine learning to identify bots in GitHub, whereas Golzadeh et al. developed a ground-truth dataset of GitHub issues and PR comments, to detect bot accounts [34, 50, 60]. We used these techniques to detect bots within issues and comments posted in the npm package repositories on GitHub and uncover that the functionalities of existing bots are not enough to address security issues within these repositories.

Interaction with Developer/maintainer in package repositories: A significant amount of prior works investigated interaction between developers to understand collaboration patterns [81], on boarding mechanisms [90], sentiment difference [48] and community dynamics [73]. However, the focus of these works typically has been on general developer collaboration and project management practices. Rather, our results are more specific to security, e.g., impact of bots and CVEs.

Detection of malicious packages in package repositories : A plethora of previous work aimed to detect malicious packages in popular package repositories such as npm and PyPI. They often use machine learning models, graph analysis, or static and dynamic analysis techniques [66, 70, 74, 77, 87, 101, 103]. Out of them AMALFI by Sejfia and Schafer is specifically tailored to detect malicious packages in JavaScript and TypeScript ecosystems [87]. Furthermore, VulNet [77], PatchFinder [72], Holmes [99], and Ranger [104] have introduced innovative solutions to enhance vulnerability prioritization, patch tracing, and secure version restoration for ecosystem-wide security management. Going one step further, Ferreira et al. [55] introduced a lightweight, permission-based system for Node.js applications, making it significantly more challenging to exploit malicious packages. These prior works generally focused on finding (and sometimes mitigating) security vulnerabilities by analyzing codebases. On the contrary, our work focuses on the security concern-related community feedback received on these packages, presumably about the security issues identified by users, making our work complementary to this prior body of work on automated malicious package detection.

Next we will present our approach of collecting community-reported security issues at scale from a stratified sample of tens of thousands of npm packages.

3 Collecting Data on Issues Reported in GitHub Repositories of npm packages

For our study, we needed to gather large-scale data about security issues reported for real-world GitHub repositories of npm packages. In this section, we detail our approach.

3.1 Selecting npm packages

Collecting package data from npm: npm (Node Package Manager) [79] is the largest and most popular repository for JavaScript libraries (called packages) today. We downloaded the list of the entire 4.3 million public JavaScript packages hosted on npm during May 2024. Furthermore, for each package, we also downloaded the *dependents*—packages which depend on this package using the API [23]. However, collecting and analyzing issues from all 4.3 million packages was computationally difficult. To that end, we decided to create a

# dependent packages	# Packages available	# Packages sampled	# Packages with GitHub link
0	3,457,606	20,000	7023
1-10	462,379	20,000	13,766
10-100	41,982	20,000	15,897
100-500	7,899	7,899	6,557
500-1,000	1,132	1,132	1,117
> 1,000	1,112	1,112	1,106
	Total	70,143	45,466

Table 1: We bucketed the npm packages by number of dependents. We present the number of total as well as sampled packages from each bucket alongwith number of packages where GitHub link was available.

# dependent packages	# Sampled Packages with GitHub link	# Unique GitHub repositories	# Total issues
0	7,023	6,755	1,323,970
1-10	13,766	12,186	2,908,173
10-100	15,897	11,687	3,687,408
100-500	6,557	5,015	1,684,658
500-1000	1,117	832	344,343
> 1000	1,106	803	958,915
Total	**45,466**	**37,278**	**10,907,467**

Table 2: Number of issues collected from GitHub pages of our sample set of npm packages.

stratified sample of these packages based on the number of dependents (i.e., how many other packages used these packages). The reason is simple—intuitively, the importance of reported security related issues for a package is correlated with the number of dependents (used in earlier work [100]).

Stratified sampling of npm packages: We simply divided the 4.3 million npm packages into six buckets based on their number of dependents: 0, 1–10, 10–100, 100–500, 500–1,000, and greater than 1,000. The number of repositories for each bucket is reported in Table 1. We note that buckets with a lower number of dependents contained an overwhelming majority of the packages, making it impractical to include them all in our analysis. Thus, we simply randomly sampled 20,000 packages from each bucket (if a bucket contained less than 20,000 packages, we included all). In the end, in our sample, we ended up considering all the packages with more than 100 dependents and 60,000 packages with less than 100 dependents (Table 1)—in total we curated 70,143 packages. Few examples of packages from each bucket is given in Table 12 of Appendix A.

3.2 Collecting issue data from GitHub for selected npm packages

Next we used a simple insight—we noted that npm registry API for individual packages [23] often provides the GitHub link (where the code is hosted). This corresponding GitHub page contained the issues raised for a particular package by the developer community (both security and non-security related) as well as discussion on how developers tried to respond to and/or resolve these issues—-we collected all of this data.

Tag	# repository using the tag	Tag	# repository using the tag
dependencies	7,526	docs	400
enhancement	4,767	discussion	386
bug	3,796	feature request	343
help wanted	2,439	wontfix	234
question	1,769	github_actions	197
good first issue	1,079	**security**	**196**
documentation	933	performance	182
javascript	568	stale	169
feature	556	bug	141
greenkeeper	458	blocked	136

Table 3: Top 20 most frequently used tags across repositories, along with the number of repositories in which they have been used, highlighting the relative scarcity of security-related tags.

Collecting GitHub links from npm registry: We collected the GitHub links for 70,143 randomly sampled packages using the npm registry API. However, not all npm packages contained GitHub links and the same GitHub links occurred in multiple packages—we collected 37,278 distinct GitHub links for 45,466 packages (Table 1).

Collecting community reported issues data from GitHub: We used the GitHub API [15] to collect *all* the issues data for 37,278 distinct GitHub repositories. In total we collected 10,907,467 issues where an overwhelming majority (10,062,759 or 92.3%) were *closed* issues. For each issue, we collected the issue title, the issue body (i.e., description), and metadata (e.g., the *tags* or usernames which posted issues/comments). We present the summary statistics of our final issues data in Table 2. Out of these more than 10 million issues, next we identify *security* related issues.

3.3 Identifying security-related issues

GitHub provides the option of tagging each issue [52, 76] where a tag is a small phrase signifying the type of issue. One issue can contain multiple tags. However, each repository can create tags on its own (with arbitrary words). Prior work [40, 41, 51, 68] highlights the effective use of issue tags in OSS for tasks like issue resolution, label prediction, and security related issue identification. To that end, we started with the idea that perhaps tags which contain security-related words/phrases will be related to security.

Tags are used with moderate frequency: We found that, out of 10,907,467 issues, around 50% (5,454,149 issues) do not contain any tags. In fact, not all repositories used tags. Out of 37,278 unique repositories, only 13,031 utilized one or more tags. The distribution of tags per repository is shown in Figure 2 of Appendix A. We collected a total of 23,356 unique tags used across 45,466 npm packages. We show the most popular twenty tags in Table 3. We manually reviewed tags across repositories that are potentially related to security such as "vulnerability", "exploit", "cve"; however these tags appeared in fewer than five repositories and did not rank among the top recurring tags. Interestingly, the word "security" appeared as

a tag in only 196 repositories.

Detecting security-related tags with Word2vec: Since tags are developer-defined and can be arbitrary, we focused on identifying tags which contain words semantically similar to security. Building on the work of Bühlmann et al. [40], which identified security-related issues by filtering tags containing the term "security", we created embeddings of each tag in our dataset using Word2Vec and calculated the cosine similarity of those embeddings with the embedding of the term "security". This Word2Vec based tag-detection approach is in line with prior work [51]. We selected tags with similarity of above 0.8 (we tried a number of thresholds and 0.8 gave the most relevant output), resulting in 25 security-related tags (given in Table 14 of Appendix A). We manually inspected them and found that all contained the word *secure*—overall these tags identified 13,835 potentially security-related issues. We manually checked 100 random such issues and confirmed their relevance to security (e.g., injection risks). Thus, the security-related tags we found (which often contained the substring *security*, e.g., "security vulnerability"), when present, indicated security-related issues. However, these security-related issues constitute only 0.13% of more than 10 million issues reported for these repositories.

3.4 Categorizing security-related issues by type of accounts who reported these issues

Identifying bot-reported security issues: Bots have become increasingly common on GitHub, automating repetitive and error-prone tasks to facilitate collaborative development [97]. Bots are, in fact, among the most significant contributors to specific software projects [61]. Among the 13,835 security-related issues, 7,731 were created by accounts with *bot* in their usernames, indicating that these issues were likely created by GitHub bots. We further collected 10,500 comments pertaining to these issues and found that 8,442 comments (80.4 %) are also posted by bot accounts (i.e. with "bot" in their username). In total, we identified 93 unique bot accounts; however, the majority (87.6%) of these issues were created by Dependabot [57]. Given the prevalence of bots in creating security-tagged issues we investigate the effectiveness of these bots in addressing security concerns for npm packages in RQ2 (Section 7).

Identifying user-reported security issues: The remaining 6,104 of the 13,835 issues are created by accounts linked to regular individuals, hereafter referred to as users. Using GitHub API, we further collected a total of 19,324 comments made by accounts while discussing these issues. Out of them only 2,826 (14.6%) were made by bot accounts. We found 70 more unique bots involved in user-reported issues. Thus our dataset contained a total of 163 bots. However, these bots mostly did not address security-related issues. They primarily posted comments signifying inactivity and staleness of the issue rather than contributing to any meaningful discussion about resolution of the security issues. Next, we focus on these user-reported issues and analyze the *process* of resolution.

4 Uncovering Process of Resolving User-Reported Security Issues in npm Using Qualitative Analysis

We next asked: how user-reported security issues are created, discussed and resolved for npm packages.

Extraction and division of quotes: We identified three prominent phases in the life-cycle of each security issue: Creation phase (captured by the user-reported description), discussion phase (interaction between developers of GitHub repositories for npm packages and users), resolution phase (captured by the last comment before the closing event). We first take 6,104 user-reported security issues as well as 16,498 comments and 6,104 closing events from our dataset. We programmatically divided this data into three phases and from each phase randomly selected 500 issue body/comments. Two researchers together extracted a total of 1,747 explanatory quotes from these issue body/comments across three phases. Next, we use open coding and affinity diagramming [85] to develop a hierarchy of themes explaining user/developer action/interaction for security-related issues in each of the three phases.

Open coding: Initially, we open-coded the quotes. For each of the three phases, we randomly selected 100 quotes and then two researchers cooperatively developed three codebooks. The codebook from the creation phase captures the types of information provided by a user, for the discussion phase, it captures how npm package developers responded to security issues, and for the resolution phase, it captures if there is any successful resolution. Additionally, we set aside 25 quotes initially to assess the saturation of themes. Specifically, we started coding 100 ($\sim$5% of) quotes (in line with Raj et al. [83]). Subsequently, the two researchers used the codebooks to independently code all the quotes in each phase. Inter-rater agreement (Cohen's Kappa) at the end of the open coding round was 0.85, indicating almost perfect agreement. At the end of open coding, the two researchers met to discuss and resolve the disagreements. Then one researcher verified that the resultant codes were sufficient to code the 25 quotes, indicating thematic saturation [86]. Finally, we ended up with a total of 13 codes across three phases.

Affinity diagramming to identify the patterns in discussing security-related issues: After the open coding round, the two researchers used affinity diagramming to jointly examine the discovered codes. They did this by looking at the collection of quotes for each code in addition to the code itself [63]. We set aside 10 random codes to check for saturation at the end. Then the coders collaboratively created higher-level themes from the rest of the codes. They kept doing this with the new higher-level themes for two more rounds, or until the coders

Creation	Discussion	Resolution
A. Issue with solution(PR)	A. Acknowledged	A. With valid reason
A.1 Description present	A.1 Spoke against with issue	A.1 Falsely Created
A.1.1 Description of the PR	A.1.1 Duplicate of another issue	A.1.1 Completed - False Positive
A.1.2 Description with testing instructions	A.1.3 Rejected the issue (eg: false positive)	A.1.2 Re-reporting of previously reported issue
A.2 No description	A.1.4. Policy adherence	A.2 Successfully resolved
B. Issue without solution	A.2 Spoke for the issue	A.2.1 Issue resolved in discussion
B.1 Reproducibility	A.2.1 Discussing about problem	A.2.2 Completed by merging PR
B.1.1 Description with code snippet	A.2.1.1 Faced same problem	A.2.3 Refer to other PR/commit
B.1.2 Description with steps of reproduction	A.2.1.2 Accepted the issue	A.3 To be completed
B.1.3 Description with error logs	A.2.1.3 Able to reproduce	A.3.1 Deferred fix (next version)
B.1.4 Description with system info	A.2.1.4. Asking for more clarification	A.3.2 Won't fix / Can't solve now
B.2 Non-reproducibility	A.2.2 Discussing about solution	B. Without valid reason
B.2.1 No Description	A.2.2.1 Solved the issue in the discussion	B.1 Closed without reason
B.2.2 Only Description	A.2.2.2 Suggested changes/solution	B.2 Completed due to staleness
B.2.3 Feature Requests	A.2.2.3 Asked for corrections	
	A.2.2.4 Agree with the solution	
	A.2.2.5 Disagree with the given solution	
	B Ignored	
	B.1 Not interested	
	B.1.1 Doesn't encourage solving the problem	
	B.1.2 No comments	
	B.2 Inconclusive	

Table 4: Our four-level hierarchical themes explaining the action/interaction between users/developers for security related issues.

thought that no more new higher-level themes could come up. In the end, we ended up with a four-level hierarchy of themes, capturing the process of creation-discussion-resolution of user-reported security issues in npm packages. Level 1 themes encompassed abstract, overarching themes generated in the final round of affinity diagramming, and Level 4 comprised the individual codes established during the open coding phase. Finally, we checked that including the 10 random codes did not add any new Level-1 and Level-2 themes, indicating thematic saturation of affinity diagramming. Our hierarchy of themes (first four levels) explaining the patterns in the resolution of user-reported security-related issues is shown in Table 4. However, we note that so far we only leveraged 6,104 user-reported issues, which limits our dataset and analysis.

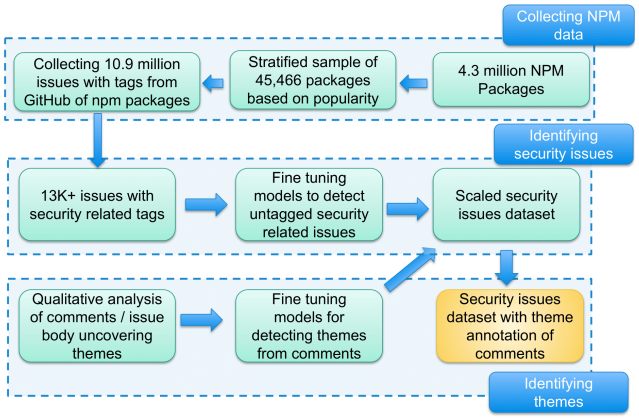

Figure 1: Pipeline for filtering security issues and identifying themes occurring in them.

5 Scaling User-Reported Security Issues Dataset and Theme Annotation

Our dataset of security-related issues contains only 6,104 issues (0.13% of a total of 10,907,467 issues). To that end, we note a potential reason: although bot-reported issues are tagged automatically, for user-reported issues, the tags are assigned by npm package developers. Thus, it might be that there are security-related issues which are not tagged with security-related tags. Thus, to identify such user-reported issues and scale up our dataset, we designed a machine learning-based pipeline presented in Figure 1.

5.1 Leveraging text classification to extend the security-related issue dataset

Creating ground truth labels: We view the problem of detecting security-related issues within the set of issues without any security-related tags as a classification task. Thus, we manually create a gold-standard dataset of 2,000 issues with two classes (security-related and non-security-related) as follows. We randomly sample the issue description of 1000 issues from the set of 6,104 user-reported issues with security-related tags. Furthermore, we randomly sampled 1000 issues which did not have security-related tags. Then two coders code these 1000 issues independently using two labels— security related or non-security related. The Cohen's kappa for the two coders was 0.74, showing substantial agreement. Then they meet, resolved disagreements, and assign final labels. To handle cases where issues had both security-related and non-security-related tags, we applied a simple rule: if an issue had at least one security-related tag, we labeled it as a security issue. In total, combining these two steps, we ended up with 1042 security-related and 1058 non-security-related issues.
We model the problem of discovering potential security-related issues (which are not tagged with security-related

Model	Accuracy	Macro avg F1
CodeBERT	0.9	0.9
BERT	0.93	0.93
RoBERTa	**0.94**	**0.94**
FLAN T5	0.89	0.89
DeBERTa	0.94	0.94

Table 5: Performance of our fine-tuned models on the test dataset for classifying issues into security-related and non-security-related.

		Predicted	
		Security	Non Security
Ground Truth	Security	6	3
	Non Security	1	90

Table 6: Confusion matrix for our RoBERTa text-classifier annotation and ground truth for previously unseen 100 issues. Only four cases were misclassified between the security and non-security categories (marked in gray).

tags) as a two-class classification problem (where the issue title and issue description will be used to classify the text). To that end, we experimented with various models as classifiers, including BERT-based models and large language models (LLMs), the latter in both zero-shot and few-shot in-context learning (ICL) settings. Among these, we selected fine-tuned RoBERTa for further analysis, as it demonstrated the best performance (F1-score of 0.94) (detailed performance metrics for the BERT-based models are provided in Table 5. LLMs performed worse than RoBERTa, achieving an F1 score above 80%—the results are given in Appendix B, Table 15).

Model Architecture: Given the nature of our task and the availability of labelled data, we utilized a range of pretrained models to address the classification challenge. These included BERT [49], RoBERTa [75], CodeBERT [54], FLAN-T5 [42] and DeBERTa [64], all of which were fine-tuned on our dataset for the downstream task of issues classification. In addition, we evaluated LLMs, namely Mistral [67], Qwen [38], Meta-LLaMA [92], and Gemma [91] using zero-shot and few-shot under ICL settings. We leveraged systematic hyperparameter tuning using the Optuna framework [36] to ensure best model configurations.

Given the sequence classification nature of our task, we utilized a range of model architectures, as outlined in Table 27 of Appendix B. We achieved the best accuracy for the downstream task using the fine-tuned RoBERTa model—class-wise performance of the RoBERTa model is shown in Table 8. Although DeBERTa showed similar performance on the test dataset, the manual analysis revealed it produced numerous false positives and negatives, hinting at its unsuitability (misclassification analysis of DeBERTa is in Table 17 of Appendix B).

Model Performance: We randomly split the dataset into 80% training and 20% validation subsets. Post-training, the RoBERTa model achieved an accuracy of 94% and an F1-score of 94% and a ROC-AUC score of 0.94 (details are provided in Table 5). To further investigate the performance and misclassification of our model, we randomly selected an additional 100 issues (excluding the 1,600 issues used for training) and one annotator manually annotated the selected issues to create ground truth. Next, we used our text classification on these 100 reviews and constructed the confusion matrix (Table 6). We note that, in this confusion matrix, only 4 issues were misclassified between security and non-security—three security issues as non-security and one non-security issue as security. Thus, we identify security-related issues correctly in 96% of cases. Our classifiers were trained on issue body and title marked with security-related tags—out of 1,000 classifier-flagged randomly sampled security-related issues, 57% didn't contain the string "security". To further validate the model's performance, we extracted attention scores from the [CLS] token across all heads in the final RoBERTa layer, averaged them, and visualized the most influential tokens using a word cloud shown in Figure- 3 (Appendix B) [35].

Applying the model to scale up user-reported untagged security issues: Finally, we used our validated issue-classifier model on 9,131,800 user-reported issues which were not tagged with any security-related tags. Our model identified 1,617,738 issues as potentially security-related (14.8% of all issues)— representing a significant rise from the originally tagged security issues, which uncovered only 13,835 security-related issues (0.13% of all issues). Thus, our work uncovered 114 times more security-related issues reported by users than actually tagged by npm package developers. A few examples of these issues are provided in Table 19 in Appendix B. This surprising result emphasized the substantial volume of security issues that remain untagged across various GitHub repositories of npm packages.

5.2 Identifying themes mentioned in extended dataset

Next, we aimed to attribute themes to the quotes in our extended dataset of 1,617,738 potentially security-related issues. To that end, we collected a total of 4,461,934 comments and issue bodies and assigned themes to them using a similar idea as before—we modelled this theme identification problem as a multi-label classification task. In particular, we represent the Level-2 (L-2) themes as labels that are allocated to each quote. We opted to conduct our study using L-2 themes, as L-1 themes were deemed excessively wide for detailed examination. Also, L-3 and L-4 themes might be more granular than L-2 themes, but we have a relatively low amount of labelled data per L-3/L-4 theme. Thus, automated theme identification with high accuracy would have been very difficult for them.

Model Architecture: We again used the following pre-trained models, BERT [49], RoBERTa [75], CodeBERT [54], FLAN-

Model	Accuracy	Macro avg F1
CodeBERT	0.78	0.67
BERT	0.77	0.63
RoBERTa	**0.82**	**0.79**
FLAN T5	0.65	0.44
DeBERTa	0.80	0.75

Table 7: Performance of fine-tuned RoBERTa model (for identification of themes explaining the action/interaction between users/developer) on the validation dataset.

	Precison	Recall	F1 score
Security Related Issue	0.95	0.93	0.94
Non Security Related Issue	0.93	0.95	0.94

Table 8: Class-wise performance of our fine-tuned RoBERTa model on the validation dataset.

T5 [42] and DeBERTa [64]. Given the different architectures of these models, we fine-tuned BERT with multi-head classification layers on top of the pre-trained model. FLAN-T5 were fine-tuned on the seq2seq task. We used the Optuna framework [36] for systematic hyperparameter tuning.

Model Performance: We divided the labelled dataset into 80% for training and 20% for validation to assess the performance of the fine-tuned models. We found that the RoBERTa model outperformed other models, identifying more than 80% of themes correctly in the reviews. The results of each model's accuracy on the validation set are displayed in Table 7. In order to further verify the accuracy of the model, we conducted a manual analysis. Specifically, we selected 100 random quotations (excluding the original 1,638 annotated) and manually annotated them with Level-2 themes. Then, we compared the correctness of our predicted themes against this ground truth. The result of the manual analysis is detailed in Table 20 (Appendix C). In both the validation set and the additional ground truth dataset, our classifier was accurate for the majority of the evaluations (almost 80%). Next, we will leverage this extended dataset of theme-annotated issues to explore our research questions.

After validating the model on both the validation set and an additional ground truth dataset—where it achieved nearly 80% accuracy across most evaluations—we applied it to identify themes within the entire dataset of 1,617,738 issues. A detailed summary of the themes identified is provided in Tables 23, 24, and 25 in Appendix C. Additionally, a summary of the number of items of each class is provided in Table 9.

6 How Prevalent is Reporting Security-Related Issues in GitHub Repositories of npm Packages? (RQ1)

Using our machine learning model, we identified more than 1.6 million (14.8%) previously untagged security-related issues within the dataset. We further explore whether these issues are evenly distributed across repositories, the extent of community discussions surrounding them, the time taken to resolve them, and the presence of CVE IDs in these reports.

Type	L2 levels	Count
Creation	Description present	786,818
	No description	39,212
	Reproducibility	418,668
	Non-reproducibility	372,700
Discussion	Spoke against with issue	194,971
	Spoke for the issue	1,786,539
	Not interested	133,035
	Inconclusive	353,046
Resolution	Falsely Created	73,670
	Successfully resolved	727,861
	To be completed	480,391
	Closed without reason	79,188
	Completed due to staleness	37,230

Table 9: Breakdown of activities across creation, discussion, and resolution phases, detailing subcategories (L2 levels) and their respective counts as identified by our fine-tuned RoBERTa model.

# dependent packages	% GitHub repositories that had at least one un-tagged security issue	% detected issues that were un-tagged
0	16.03	15.88
0-10	24.75	16.66
10-100	39.34	14.54
100-500	44.3	14.72
500-1000	55.77	10.94
>1000	66.09	10.98

Table 10: Distribution of security-related issues across different buckets.

How are these security-related issues distributed over the buckets? We first check the fraction of issues reported in our buckets of npm repositories based on their dependent repositories. The result is presented in Table 10. The percentage of repositories with security-related issues increased as we moved to buckets with npm packages with more dependents. However, the percentage of security-related issues remained fairly consistent across all buckets, indicating that the overall occurrence of such issues is relatively uniform over buckets.

How much do the community discuss about security-related issues? We examined user engagement by analyzing comments on security-related issues. Our dataset of 1,617,738 security issues, spread across 15,048 repositories, contains a total of 4,461,934 comments. More than 23% of the issues do not even receive any comments highlighting the lack of interaction in these security-related issues.

Do security-related issues persist over time and take longer to resolve? We observed that security-related issues appeared consistently over the years, highlighting their persistent nature across different periods (Table 13 of Appendix D).To evaluate whether security-related issues require more time to resolve, we computed the resolution time (difference between its opening and closing timestamps) for a random sample of 10,000 non-security-related issues and compared it with

the resolution time of security-related issues. Security-related issues take an average of 56.47 days to close, compared to an average of 48.88 days for non-security issues.

Do these issues contain specific mention of CVE IDs? Among the security-related issues, only 4,783 issues (out of 1.6 million) mention a total of 7,184 CVE IDs, with 2,233 of these being unique. Thus the low inclusion of CVE ID highlights that publicly recognized vulnerabilities might not help the users to identify the security concerns within the npm ecosystem. Thus within GitHub repositories of npm packages, numerous security issues have not been formally identified or documented under CVE IDs. (Note: This analysis considers only CVE mentions.) The five most frequently mentioned CVE IDs are in Table 28 of Appendix D.

7 Effectiveness of Bots in Detecting and Addressing Security Related Issues (RQ2)

Next, we explore how bots function within GitHub repositories of npm packages since out of 13,835 issues with security-related tags, 7,731 (55.9%) were bot-reported.

7.1 Understanding role of bots in bot-reported security-related issues in npm

In Section 3 we noted that Dependabot reported 87.6% bot-reported issues in GitHub repositories of npm packages. It is an automated tool that manages project dependencies by identifying and updating vulnerable packages (among other functions). Dependabot works by analyzing dependency files, including package.json and package-lock.json, to check for outdated or insecure packages. When a vulnerability is detected, Dependabot checks the lock file to identify the affected version and its impact on other dependencies. It provides two main services– 1) version updates, which automatically update dependencies to the latest versions according to the configuration in *dependabot.yml*, and 2) security updates, which scan repositories for known vulnerabilities and alert repository owners.

Despite its utility, Dependabot can overwhelm developers with numerous pull requests, especially in repositories with many dependencies, making management difficult [93, 94]. It often creates excessive pull requests for bloated dependencies, and its configuration can cause issues for developers. Additionally, Dependabot's reliance on predefined vulnerability databases limits its ability to detect undisclosed or emerging threats(e.g.: supply-chain attacks [80]), leaving security gaps.

7.2 Understanding role of bots in user-reported security-related issues in npm

Apart from actively creating more than seven thousand issues tagged as security, bots are also involved in user-reported issues (with account names containing the phrase "bot"). The activities along with their frequencies are given in Table 21, Appendix E. We found that 120 bots were involved in such issues. We focus on them for further understanding their effectiveness in reporting and mitigating security issues in GitHub repositories of npm packages.

Classification of bots We found that many of these bots are hosted on the GitHub Marketplace [58], which may provide free or paid access to the users, while others are independently created by developers. However, hosting a bot on the GitHub Marketplace does not guarantee the availability of its source code publicly; one can keep its source code private. Moreover, some bots are restricted within its own organisation. Based on these observations, we classified the 120 bots into two main categories: (See Table 22 of Appendix E)

Bots deployed on GitHub marketplace: We found 71 bots which are deployed on GitHub marketplace. These can be further classified based on the availability of their source codes. We identified 57 bots which are hosted for public usage on the GitHub marketplace. 40 of these bots have their source codes publicly available, while 17 of them preferred to keep their source codes private. Their transparency allows developers to review their implementation, modify them for specific needs, and integrate them into their workflows effectively. The remaining 14 bots of the 71 are those whose source codes and accessibility are kept private. These are mostly used internally by the organisations that created them.

Bots not deployed on GitHub marketplace: The remaining 49 bots were not deployed in GitHub marketplace. Moreover, they are privately maintained by individuals or organisations. These accounts were recognized because their usernames included the term "bot". Unlike GitHub apps, these are not standardized, making it unclear whether they are real bots or simply user accounts named as such. The next subsection discusses their behaviour and roles in more detail.

7.3 Uncovering functionality of bots

To better understand the role these bots play in security-related issues, we analyzed various activities that bots perform (using publicly available information). We considered the bots from two groups: those with publicly available source codes and those with private repositories but listed in the marketplace, providing descriptions about their working for analysis. Combining both the categories, we analyze 51 bots.

7.3.1 Qualitative analysis of bot functionalities

For each of the 51 bots, we collected their GitHub repositories (readme, comments) and marketplace descriptions. We extracted explanatory quotes about bot functionality from these data (yielding 150 unique quotes) and grouped them based on their capabilities using the affinity diagramming method. Two researchers collected and analyzed data on 51

A. Dependency management	github-actions	bors
dependabot	**D. Security**	github-merge-queue
renovate	socket-security	**E.3 Project Management**
B. PR management	robocop	issue-sh
stale	mend-for-github-com	angular-robot
mergify	github-advanced-security	project-bot
pullapprove	**E. Utility**	dosubot
delete-merged-branch	**E.1 Policy & CLA**	sync-by-unito
coderabbitai	microsoft-github-policy-service	**E.4 Issue Related**
kodiakhq	salesforce-cla	git2gus
trunk-io	google-cla	dotnet-issue-labeler
gitpod-io	dotnet-policy-service	linear
C. CI/CD	linux-foundation-easycla	**E.5 Code Coverage**
cypress	**E.2 PR related**	codecov
codesandbox-ci	changeset-bot	codeclimate
azure-pipelines	jupyterlab-probot	**E.6 Miscellaneous Utility**
shields-deployment	jupyterlab-dev-mode	lock
vercel	pull	gitpoap-bot
nx-cloud	conventional-commit-lint-gcf	welcome
netlify	pull-request-size	gitwave
render	release-clerk	sentry-io
gatsby-cloud	what-the-diff	**F. Miscellaneous**
grafana-delivery-bot	a8c-probot-stale	sonarcloud

Table 11: Hierarchical levels of functionalities of bots(along with the bots).

bots, categorizing them based on their functionalities and publicly available descriptions or marketplace listings.

First, data from a randomly selected subset of 10 out of 51 bots was set aside to test the saturation of the hierarchy. The researchers collaboratively analyzed the functionalities of the remaining bots, grouping them into higher-level themes through an iterative process. This process was repeated across two additional rounds, refining the themes each time. By the third and final round, the researchers concluded that no new high-level themes could emerge, indicating thematic saturation in the affinity diagramming process. The final hierarchy consists of two levels, as shown in Table 11. At the highest level (Level 1), the bots were classified into six overarching themes. Dependency Management, PR Management, CI/CD, Security, Utility, and Miscellaneous. The utility category was further refined in Level 2, where its subclasses elaborated on the themes described in Level 1, offering more precise and specific categorisation of bot functionalities. (Details given in Table 30, Appendix E)

The hierarchy of themes provides a comprehensive view of bot functionalities, ranging from automating dependency updates and managing pull requests to addressing security concerns and ensuring smooth integration and delivery workflows. Interestingly, despite analyzing 51 bots across various functionalities, we observed that only four bots are specifically involved in addressing security-related purposes. This finding is notable considering the widespread presence of user-reported security-related issues, suggesting that bot involvement in this critical domain remains minimal.

To better understand their capabilities, we analyzed the source code of 40 (out of 51) bots with publicly available codebases. We manually downloaded these repositories from GitHub (covering various tech stacks) and examined their implementation. Our analysis involved inspecting the working principle from source code to determine whether the bots relied on static rules, pattern matching, or learning-based decision making. We also reviewed their dependencies to examine which external libraries they relied on, particularly looking for any use of machine learning frameworks. Through this systematic process, we found that 85% (34 bots) relied entirely on pre-defined, rule-based logic, offering limited adaptability to diverse and evolving security issues. Six bots incorporated AI/ML in some form—however, one of them was paid, and two were still in beta stages with restricted functionality. This highlights a clear gap between the potential of AI/ML in security automation and its limited real-world adoption among existing bots. Table 29, Appendix-E summarizes their types and underlying mechanisms.

7.3.2 Under-adoption of bots for detecting and mitigating security-related issues

Among the 51 bots analyzed, only four *viz.* RoboCop, mend-for-github-com, github-advanced-security and Socket Security Bot were identified as actively addressing security-related issues. However, the extremely limited availability and adoption are due to two main factors:

First, all these bots are paid services, which may deter smaller teams or open-source projects from using them. Second, these bots rely heavily on static analysis techniques, which, while valuable, may not adequately address dynamic or runtime vulnerabilities. Consequently, the under-utilization of such bots highlights a significant gap in leveraging automated tools for detecting security issues within GitHub repositories of npm packages.

8 Characterizing the Interaction Between the Users and Maintainers of GitHub Repositories of npm Packages (RQ3)

We have already identified the different phases involved in the resolution of an issue. Now we check how the community interacts with the maintainers of the repositories.

8.1 During creation phase

Our qualitative analysis identified two broad themes while investigating the types of information provided by a user while reporting a security issue.

Interaction for security issues while offering a solution: This includes the pull requests made by users. Most of them have a proper description of how it solves the problem. In an issue of storybookjs [6], the user tells clearly about how the problem is solved *"What I did: Bumps mdx2-csf which resolves a security issue in 'loader-utils'"*. Moreover, some of them contain instructions regarding testing their code as in an issue of superset [13]*"### TESTING INSTRUCTIONS 1) Visit an Embedded Dashboard, 2) Generate a screenshot of the Dashboard 3) The screenshot should be downloaded successfully"*. However, sometimes they do not contain any description. Only a title is provided in such cases as in case of an issue of grafana [17] *"Issue title: Move out Server Admin to a separate menu item on SideMenu"*.

Interaction for security issues without a solution: These include issues that describe the problem that a user had while using that particular library. These are further classified based on their ability to reproduce the problem from their description. Usually, reproducible issues are provided with code snippets related to the concerned issue. In an issue of apollo-client package [7] and pulumi-azure-native [16], users have provided the code snippets to reproduce the issue. Furthermore, users often provide proper steps for the reproduction of the issue. For example, in wp-calypso [26], the initiator of the issue mentions the steps of reproduction as *"1) Starting at URL: https://wordpress.com/me/security/connected-applications, 2) Click on any of connected application 3) See list of Access Permissions, 4) Switch to another interface language and verify step 3 again"*. Apart from that, users often tend to report the error logs, as reported in an issue of expo package [24] *"Unhandled promise rejection: Error: Could not encrypt/decrypt the value for SecureStore] at node_modules/reactnative/Libraries/BatchedBridge/ NativeModules.js:106:50 in promiseMethodWrapper ... "*. Users even describe their system information in which they have faced the issues as in case of an issue reported in strapi package [19] *"### System Node.js version: 16.13.2, NPM version: 8.1.2, Yarn version: 1.22.15, Strapi version: 4.0.5, Database: PostgreSQL, Operating system: Alpine"*.

However, all issues reported are not provided with proper reproduction steps. Most of them contain some form of feature requests. For example, in their package [20], a user reports a feature request as *"Is Theia interested in expanding their ESLint config to include XSS sink scanning. Feature Description: I work on Cloud Shell (ide.cloud.google.com). We use Theia to build our editor. We have created an eslint config that has generated a list of XSS sinks in Theia (which I am currently sending fixes out for). Are you all potentially interested in having this upstreamed?"*.

8.2 During discussion phase

In the qualitative analysis of the discussion phase, repository maintainers responded to the issue in one of two broad themes: they either acknowledged it or ignored it.

Acknowledged the security issue: The maintainers acknowledge an issue either positively or negatively. Positive acknowledgments involve discussing the problem, such as recognizing they faced the same issue, accepting it, reproducing it, or requesting more details about how the attack works. In an issue of storybooks [27], a user says he has also faced the same issue *"... I'm facing the same issue in 'npm audit'. This is in version '5.3.18' of 'storybook/addon-info'..."*. They may also discuss solutions by resolving the issue in the discussion, suggesting changes, asking for corrections, or agreeing with the proposed solution. For example in an issue of pulumi [22], maintainer mentioned *"This is likely due to the way the system treats 'id' as a special property... I would suggest using the 'RandomString' resource (and then using the language-specific libraries to generate whatever appropriate output format you want in terms of using the value as base64 or hex or whatever else you like)"*

Negative acknowledgements include marking the issue as a duplicate. They may also involve disagreeing with the suggested solution, rejecting the issue as a false positive, or deeming it not relevant. A comment to saying about the duplicity of the issue in agoric [30], *"dup of #6007"*. Additionally, maintainers may request adherence to policies regarding the disclosure of security vulnerabilities. Like in an issue of aspnetcore [12], *"Thanks for contacting us. While this may be a great idea, it is not aligned with our long-term vision to make it part of the framework. For many other ideas which don't belong to the framework we encourage the community to build and ship on their own, contributing to the expanding .NET Ecosystem..."*— while this reflects a scope decision, it may feel dismissive to the user as no alternative is suggested.

Ignored the security issue: Maintainers may ignore the issue by showing a lack of interest, which includes not encouraging efforts to solve the problem or providing no comments or engagement. For example, In spartacus [8], a maintainer commented *"Dropping bug label as this is not a Spartacus bug, nor something we can fix in Spartacus. It's a backend bug. It's also a niche, and we shouldn't spend time here imho."*

The wording may sound dismissive due to lack of guidance or follow-up.

8.3 During resolution phase

During the resolution phase, repository maintainers typically close issues under two broad categories: with a valid reason or without one.

Closed with valid reason: Issues may be closed for a variety of reasons, such as false creation, successful resolution, or deferred completion. Falsely created issues include those marked as false positives or those accidentally created without verifying if they had been previously reported. For example, in SonarJS [10], a maintainer commented before closing the issue: *"It seems this was already taken care of by https://github.com/SonarSource/SonarJS/pull/4480"*. A similar, example was seen in RocketChat [29], where a maintainer commented *"Looks a duplicate of #3069"*. Successfully resolved issues are closed after resolving the problem in discussions, merging a pull request (PR), or referencing a related PR or commit made after the issue was reported. For example, in MetaMask snaps [2], a maintainer resolved the issue in discussion saying *"Consider using proxies instead of hardening each individual endowment with custom wrappers, and only harden the return if it returns something other than itself"*. Another instance where an issue is closed involves merging a pull request is found in netlify [3]. Some issues remain open for future action, categorized as "to be completed," which may include deferring the fix to a later version or deciding not to address the issue due to its complexity or infeasibility. An issue in joystream [4] is closed by mentioning *"Nothing severe"*. Another example found in npm [31] *"However! It's also not incredibly high urgency for the team. There is a security dimension to this (in that bad actors could shadow important system binaries for nefarious purposes), but that falls into the category of "high impact, low probability" threats..."*.

Closed without a valid reason: Issues are sometimes closed simply because of staleness. E.g., an issue of Dotnet [21], marked as stale with the last comment before closing *" This issue has been automatically marked as stale because it has been marked as requiring author feedback but has not had any activity for 4 days. "*. Although the closure followed an automated policy, doing so after just 4 days of inactivity may feel abrupt to users. In some cases, issues are closed even without any comment in issues of flow-core [14], Grafana [25].

9 Factors correlating the resolution of security related issues (RQ4)

Security-related issues often experience extended resolution times, as evidenced by the slower closure of the 13,835 security-tagged issues. Additionally, we observed that issues with CVE tags tend to have faster resolution. Thus, we naturally asked the question–What are the different factors that impact the resolution of a security issue?

To that end, we developed three Generalized Linear Mixed Models(GLMMs), each targeting one of the three dependent variables–time to close, staleness, and successful resolution. The independent parameters included: (i) CVE mentions, (ii) number of comments (iii) weekly downloads, (iv) number of active maintainers in the past year, (v) issue reproducibility, and (vi) bot involvement. A detailed description of these factors is presented in Table 34 of Appendix F. Repository-level variations were modelled as a random effect. Results for the models are summarized in Tables 31, 32, and 33 of Appendix F. Next, we present the key takeaways.

9.1 Presence of CVE reduces time to close

The presence of a CVE in an issue body significantly decreases the time to close. This prioritization likely stems from their critical nature and potential security implications. Issues mentioning a CVE had a mean time to close of 174.08 days and a median of 70 days, compared to a mean of 307.19 days and a median of 127 days for issues without such mentions. These findings highlight the role of CVE in expediting issue resolution and reducing the risk of stagnation.

9.2 Reproducibility prolongs closure and increases staleness, and hinders successful resolution

Reproducibility increases time to close: Reproducibility has a significant impact on the time to close, staleness, and successful resolution of issues. Reproducible issues tend to have higher time to close because they require additional time for proper verification and reproduction of the problem.

While reproducibility is typically considered helpful for resolution of issues [69, 84], our analysis reveals a counterintuitive finding: reproducible issues often remain open longer. For instance, a long-standing issue in angular.js [11] opened in 2020 remained unresolved for years and was only closed in 2024 by a bot due to inactivity. The bot commented: *"This issue has been automatically locked due to inactivity. Please file a new issue if you are encountering a similar or related problem. Read more about our automatic conversation locking policy"*. Another case, found in expressjs [1] involved a maintainer commenting, *"Yea, this pull request has just lagged around here for years, and seems like we're not going to merge it at this point, and so no reason to keep a stale pull request hanging around for no reason :)"*, which indicates how long delays may occur in issues being abandoned, even if they are reproducible. In terms of stats, reproducible issues have a mean closing time of 105.16 days (median 8 days), while non-reproducible issues close in a mean of 47.21 days (median 2 days), highlighting that reproducibility delays resolution.

Reproducibility increases staleness: Contrary to expectation, reproducibility does not always expedite resolution. Instead, reproducible issues are also more likely to become stale, especially when there is a lack of capacity to reproduce the issue or when the issue is not addressed promptly. For example, in angular.js [28], a maintainer closed the issue after seven days of no feedback, stating: *"I'm going to close this issue because we haven't got any feedback. Leave a comment if you can provide new feedback"*. This shows that when an issue cannot be reproduced or lacks updates, it is more likely to be marked as stale. Similarly, other issues have been closed due to the maintenance of repositories when the maintainers no longer considered them relevant or reproducible. One such case is found in storybookjs [5], where a comment stated, *"We're cleaning house! Storybook has changed a lot since this issue was created and we don't know if it's still valid. Please open a new issue referencing this one if this is still a problem in SB 7.x / you can provide a consistent reproduction in 7.x"*.

Reproducibility hinders successful resolution: Reproducibility can also decrease the likelihood of a successful resolution. Issues that remain unresolved for a long time due to reproducibility delays tend to be marked stale. In terms of stats, 16,543 reproducible issues were found to be marked as stale. On the other hand, such issues were also closed with an assurance of taking care of them in the next release. For example, a case found in pact-js [9] where a maintainer closed the issue saying *"Thanks for your help on this one. #282 is being released as we speak (v8.0.5), closing"*. Moreover, many of the reproducible issues were closed with limited conversation, due to prolonged inactivity or because they were already addressed in a newer release.

Why reproducibility in issues might not help resolution? We found that reproducible issues with CVE references show faster resolution times—mean time to close for reproducible issues with CVE mention is 70.75 days (median: 5 days), whereas those without CVE mention have a mean time to close of 105.6 days (median: 8 days). This further highlights that the benefit of reproducibility is contextual—it helps when developers are sufficiently motivated or equipped to act promptly. Thus even reproducible security-issues, without CVE may face challenges in achieving timely resolutions, often due to external constraints or a lack of response from the developers.

9.3 Bot involvement is correlated with increased staleness and reduced resolution

The involvement of bots in Github discussions is strongly correlated with increased staleness and lower chances of successful resolution. Bot activity appears in 72.37% of stale issues (26,945/37,230) and 37.27% of unresolved issues (287,028/770,014). Bots typically intervene during the later stages of an issue's lifecycle, often marking it as stale or automatically closing it, thereby limiting further resolution efforts. For example, in an issue of opensea-js [18], a bot commented *"This issue has been automatically marked as stale because it has not had recent activity. It will be closed in 14 days if no further activity occurs. Thank you for your contributions. If you believe this was a mistake, please comment"*. This demonstrates how bot involvement aligns with the final stages of an issue. It is potentially associated with staleness and automated closure after periods of inactivity.

9.4 No impact of active maintainers

Our analysis indicates that the resolution of an issue is independent of the number of active maintainers. This could be attributed to the limited participation of active maintainers in discussions. Moreover, we found the number of active maintainers commenting on an issue to be very low for most of the repositories. Specifically, the mean and median numbers of active maintainers are 4.83 and 2, respectively, while those actively engaged in commenting are only 3.78 and 1. To further strengthen our claim, we computed Pearson's product-moment correlation coefficient between them and found it to be statistically significant ($R = 0.9918$, $p = 0.0$). This confirms our claim that very few active maintainers are engaged in the process of resolution of an issue.

10 Implications

Our study of identifying security issues over npm packages unearths a variety of important observations. To that end, we point out the implications of our study for npm package owners, maintainers, and the open-source community in general.

Tagging of issues in GitHub is not uniform: We know that tagging of issues in GitHub is a fairly common phenomenon [52, 76] but is underutilized. Surprisingly, all repositories of our npm dataset do not use tags. Out of 37,278 unique repositories, only 13,031 utilized one or more tags. We found that out of 10,907,467 issues, around 50% (5,454,149 issues) does not contain any tags. As a result of this, we found that only 0.13% of 10M+ issues were tagged as *security*. This observation highlights the importance of implementing a standardized tagging system across repositories. It is crucial for repository maintainers to consistently and accurately apply these tags to effectively categorize issues.

Bots are inefficient in resolving security-related issues: Bots, while actively involved in creating security-related issues, typically operate based on predefined rule-based systems. Despite its limitations, these bots are also involved in addressing user-reported security issues. However, only a limited number of bots claim to support such security concerns, and most of these are paid solutions with limited effectiveness. Our analysis further reveals limited adoption of AI/ML in these bots (for both general and security-related function-

alities), which stands in contrast to prior work emphasizing the promise of AI/ML techniques for improving security testing [43, 62]. This underscores the need for the development of advanced and efficient bots to handle security challenges.

Reproducibility, CVE mention and bot involvement influence resolution of security-related issues: Security-related issues often experience extended resolution times due to additional time required for verifying and reproducing the issues. Moreover, reproducible issues are more likely to become stale because of a lack of capacity to reproduce or feedback. Thus, such issues are not being successfully resolved. On the other hand, issues with CVE mention prompt faster resolution due to its critical nature and potential security implications. Bots, however, are found to negatively impact the resolution process as they often mark the issues as stale or close them which limits further resolution efforts.

11 Recommendations for Stakeholders

Finally, we present a set of recommendations tailored for key stakeholders in the context of npm package repositories, including package owners and maintainers, developers of bots for the npm ecosystem, security researchers and end users.

Recommendation for npm package owners and maintainers: Our findings highlight that the tagging of issues in GitHub repositories is inconsistent and often underutilized. To improve this, package maintainers should implement a standardized tagging system for better categorization, especially for critical issues like security vulnerabilities. For maintainers, we recommend involving more active contributors and prioritizing CVE-tagged issues to ensure timely attention to known vulnerabilities. Automated tools can aid with consistency, while active involvement with contributors and the security research community can address the gaps. Clear tagging guidelines and regular feedback from users will enhance collaboration and enhance the security and functionality of GitHub repositories of npm packages.

Recommendation for bot developers for npm ecosystem: Our findings highlight that current bots, while involved in creating and addressing security-related issues, often operate on limited, rule-based systems that lack the flexibility to handle complex security concerns. Bot developers should improve their systems using smarter technologies involving machine learning or AI. We also recommend developing bot-based recommender systems (e.g., usable CodeQL [56] tools) to assist users and maintainers during issue reporting—such tools could identify known CVEs or similar issues from other repositories, helping reduce duplication and streamline triage. Moreover, bots must enhance message clarity by including suitable tags, informative descriptions, and reproducible steps. Furthermore, given that the entire npm ecosystem is open source, bots should be developed as free solutions, ensuring accessibility to all users while maintaining high-quality, cost-effective support for addressing security challenges.

Recommendations for the security research community: Our results show that critical security issues, despite being reported in the CVE database, are often not properly tagged in multiple issues, leading to their negligence (see Section-6). This oversight can have serious repercussions, as many packages are directly affected, along with all other packages depending on them. Therefore, better management and prioritization of security issues are needed to ensure that vulnerabilities are promptly addressed and their broader impact is fully recognized. To support this, the security research community could develop tools not only for identifying and prioritizing overlooked issues but also for automating issue reproduction, verification, and early detection of false positives—thereby improving the accuracy and efficiency of vulnerability triage for both users and maintainers.

Recommendation for users: We have already highlighted that reproducibility hinders the resolution process. To that end, the users who are raising security-related issues should provide a proper step-by-step guide towards reproducing the problem, which will aid the maintainers in understanding and verifying the issue quickly. Moreover, they should be more prompt towards replying to queries of maintainers. This will reduce the chances of such issues being marked as stale. On the other hand, users should do proper research before raising issues, as we have identified numerous cases where the issues are marked as false positives or duplicates of existing issues.

12 Conclusion

This study investigated the inefficiencies in addressing security-related issues within the npm ecosystem by analyzing 10,907,467 issues across 45,466 npm packages. Our findings revealed a stark gap in tagging practices, with only 0.13% of issues explicitly labeled as security-related, despite manual and machine-learning-based analyses identifying 14.8% as relevant to security. This difference shows that we urgently need a common way to tag issues to improve how we organise and solve them. Furthermore, we identified the constraints of automated systems, which frequently depend on inflexible, rule-based frameworks. This reliance can result in the misclassification of issues as obsolete or the premature closure of cases, thereby hindering efforts towards effective resolution. Reproducibility problems slowed down fixing issues, while CVE comments helped solve problems faster because they have a set level of importance.

These insights call for smarter tools, standardized tagging, and collaborative efforts among stakeholders to enhance security management. Addressing these gaps would facilitate a safer and secure npm ecosystem that more effectively serves both developers and users.

13 Ethical Considerations

We initially consulted the ethics committee of our institution regarding our study protocol. They indicated that ethical review was not required, as we collected and analyzed data from publicly accessible npm and GitHub repositories. However, we recognize that this exemption does not, by itself, ensure ethical research. Collecting and analysing public data can still raise ethical concerns, especially when it was not created with research purposes in mind [39]. To that end, we made every effort to conduct our research responsibly—minimizing potential harm and maximizing potential benefits for stakeholders such as npm repository developers, users who contributed content in our dataset, and the platforms (e.g., GitHub and npm) themselves.

In line with prior work on software repository analysis [44,47], we relied exclusively on data publicly shared by users in *open forums* (forums where no registration was required to access content, e.g., the npm repository and comments on their corresponding GitHub repositories) such as GitHub issues, pull requests, and discussions. We strictly adhered to the ethical principles outlined in previous studies [53], avoiding any private communications or data collection behind authentication walls. Specifically, all data was obtained via GitHub and npm's official APIs, and we strictly followed their rate limits and terms of service.

Although the use of public data is common in research [39], ethical concerns can still arise—particularly when users have not explicitly consented to having their contributions analyzed. To mitigate such risks, we took several precautions. Data was securely stored on password-protected machines physically located within our institution's firewall, with access limited to only our research team. We did not perform any analysis targeting specific developers or repositories.

Our dataset contains the reporting, discussion, and debate of issues specifically within open-source projects, often conducted in public forums like GitHub. These disclosures reflect broader open-source community norms around transparency and shared responsibility. One of our primary goals is to support and align with these norms by facilitating a more informed and nuanced conversation about the ethical use of such data to uncover high-level data-driven insights which can benefit the open source projects.

Thus, we strongly believe our research has been conducted ethically and contributes valuable insights. We hope that these findings can help strengthen collaboration and security practices within the open-source ecosystem—highlighting the potential benefits of this work to the broader community.

14 Compliance with Open Science Policy

We will adhere to open science policy. We are releasing the artifacts (models) with accompanying code so that the results of this work can be reproduced and the models can be used in further research (e.g., for discovering security-related issues). Since the raw data might contain identifying information about security issues, to address potential ethical concerns about open science compliance, we are sharing the raw datasetsxs only with verified researchers upon request, rather than releasing them publicly. We are providing clear instructions about accessing the raw data for academic research to researchers on providing informed consent about acceptable data usage. For accessing the models and instructions to access data, please visit https://doi.org/10.5281/zenodo.15614029

15 Acknowledgement

We thank the anonymous reviewers and our shepherd for their valuable feedback. We also thank Gunjan Balde for his help throughout this work. The authors thank everyone who facilitated access to the necessary resources during the data collection from GitHub. This research was (partially) funded by a Google India Faculty Research Award.

References

[1] add request.local,inside,outside getters to test where a request is coming from by blitmap · pull request #2748 · expressjs/express. https://github.com/expressjs/express/pull/2748.

[2] Additionally harden common endowments · issue #1018 · metamask/snaps. https://github.com/MetaMask/snaps/issues/1018.

[3] Allow plugins to access specific secret environment variables · issue #193 · netlify/build. https://github.com/netlify/build/issues/193.

[4] Are unsettled english auctions safe? · issue #2969 · joystream/joystream. https://github.com/joystream/joystream/issues/2969.

[5] Bug with 'addon-docs' and " html element · issue #15810 · storybookjs/storybook. https://github.com/storybookjs/storybook/issues/15810.

[6] Bump mdx2-csf dependency by shilman · pull request #19885 · storybookjs/storybook. https://github.com/storybookjs/storybook/pull/19885.

[7] cache leaks secrets · issue #3592 · apollographql/apollo-client. https://github.com/apollographql/apollo-client/issues/3592.

[8] Cannot remove coupon if the coupon code contains "." · issue #4725 · sap/spartacus. https://github.com/SAP/spartacus/issues/4725.

[9] Cannot run provider verification against https endpoint that has ca, if called from a http endpoint · issue #281 · pact-foundation/pact-js. https://github.com/pact-foundation/pact-js/issues/281.

[10] Deprecate rule s5743 · issue #4467 · sonarsource/sonarjs. https://github.com/SonarSource/SonarJS/issues/4467.

[11] Docker best practices · issue #37442 · angular/angular. https://github.com/angular/angular/issues/37442.

[12] Dynamically updating authentionschemes on authorizationpolicies · issue #54412 · dotnet/aspnetcore. https://github.com/dotnet/aspnetcore/issues/54412#issuecomment-1992596067.

[13] fix(embedded): Dashboard screenshot should use guestuser by geido · pull request #30200 · apache/superset. https://github.com/apache/superset/pull/30200.

[14] Found leaked credentials. · issue #98 · ollionorg/flow-core. https://github.com/ollionorg/flow-core/issues/98.

[15] GitHub REST API documentation. https://docs.github.com/en/rest.

[16] liststaticsitesecrets results are not marked as secrets · issue #2408 · pulumi/pulumi-azure-native. https://github.com/pulumi/pulumi-azure-native/issues/2408.

[17] Move out server admin to a separate menu item on sidemenu · issue #15591 · grafana/grafana. https://github.com/grafana/grafana/issues/15591.

[18] Opensea testnets - unsupported operand type(s) for /: "nonetype" and "nonetype" · issue #122 · projectopensea/opensea-js. https://github.com/ProjectOpenSea/opensea-js/issues/122.

[19] Passwords are not validated upon user creation · issue #12320 · strapi/strapi. https://github.com/strapi/strapi/issues/12320.

[20] Passwords are not validated upon user creation · issue #12320 · strapi/strapi. https://github.com/strapi/strapi/issues/12320.

[21] Policyevaluator doesn't have correct logic · issue #56656 · dotnet/aspnetcore. https://github.com/dotnet/aspnetcore/issues/56656.

[22] Randomid still includes unencrypted key in state file · issue #8946 · pulumi/pulumi. https://github.com/pulumi/pulumi/issues/8946#issuecomment-1005987646.

[23] registry - npm docs. https://docs.npmjs.com/cli/v8/using-npm/registry.

[24] Removing then re-installing an apk causes the securestore read to throw exceptions · issue #19018 · expo/expo. https://github.com/expo/expo/issues/19018.

[25] Secure phantomjs png rendering · issue #1619 · grafana/grafana. https://github.com/grafana/grafana/issues/1619.

[26] Security: Strings in list of access permissions of connected applications are not localized · issue #7017 · automattic/wp-calypso. https://github.com/Automattic/wp-calypso/issues/7017.

[27] [security] vulnerability of low severity in "@storybook/addon-info > marksy > marked" · issue #7842 · storybookjs/storybook. https://github.com/storybookjs/storybook/issues/7842#issuecomment-641845722.

[28] self.url method causes trouble with ie11 (routing) · issue #16825 · angular/angular.js. https://github.com/angular/angular.js/issues/16825.

[29] Setting admin_pass shows password in log · issue #4806 · rocketchat/rocket.chat. https://github.com/RocketChat/Rocket.Chat/issues/4806.

[30] Test that offer snoozing works for both sides of the psm contract · issue #6071 · agoric/agoric-sdk. https://github.com/Agoric/agoric-sdk/issues/6071.

[31] Warn at install about path lookup issues in global installs · issue #7255 · npm/npm. https://github.com/npm/npm/issues/7255.

[32] *ASE '17: Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, 2017.

[33] *ICSE-SEIP '18: Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, New York, NY, USA, 2018. Association for Computing Machinery.

[34] Ahmad Abdellatif, Mairieli Wessel, Igor Steinmacher, Marco A. Gerosa, and Emad Shihab. Bothunter: an approach to detect software bots in github. In *Proceedings of the 19th International Conference on Mining Software Repositories*, MSR '22, page 6–17, New York, NY, USA, 2022. Association for Computing Machinery.

[35] Aditya. Understanding the [cls] token in bert: A comprehensive guide, 2021.

[36] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework, 2019.

[37] Mahmoud Alfadel, Diego Elias Costa, Emad Shihab, and Bram Adams. On the discoverability of npm vulnerabilities in node.js projects. *ACM Trans. Softw. Eng. Methodol.*, 32(4), May 2023.

[38] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

[39] Amber M. Buck and Devon F. Ralston. I didn't sign up for your research study: The ethics of using "public" data. *Computers and Composition*, 61:102655, 2021. Rhetorics of Data: Collection, Consent, & Critical Digital Literacies.

[40] Noah Bühlmann and Mohammad Ghafari. How do developers deal with security issue reports on github? In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, SAC '22, page 1580–1589, New York, NY, USA, 2022. Association for Computing Machinery.

[41] Jordi Cabot, Javier Luis Cánovas Izquierdo, Valerio Cosentino, and Belén Rolandi. Exploring the use of labels to categorize issues in open-source software projects. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 550–554, 2015.

[42] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models, 2022.

[43] Roland Croft, Dominic Newlands, Ziyu Chen, and M. Ali Babar. An empirical study of rule-based and learning-based approaches for static application security testing. In *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ESEM '21, New York, NY, USA, 2021. Association for Computing Machinery.

[44] Ajoy Das, Gias Uddin, and Guenther Ruhe. An empirical study of blockchain repositories in github. In *Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering*, EASE '22, page 211–220, New York, NY, USA, 2022. Association for Computing Machinery.

[45] Alexandre Decan, Tom Mens, and Eleni Constantinou. On the impact of security vulnerabilities in the npm package dependency network. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, page 181–191, New York, NY, USA, 2018. Association for Computing Machinery.

[46] Alexandre Decan, Tom Mens, and Philippe Grosjean. An empirical comparison of dependency network evolution in seven software packaging ecosystems. *Empirical Software Engineering*, 24:381–416, 2019. Published: 10 February 2018, Issue Date: 15 February 2019.

[47] Alexandre Decan, Tom Mens, Pooya Rostami Mazrae, and Mehdi Golzadeh. On the use of github actions in software development repositories. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 235–245, 2022.

[48] Giuseppe Destefanis, Marco Ortu, David Bowes, Michele Marchesi, and Roberto Tonelli. On measuring affects of github issues' commenters. In *Proceedings of the 3rd International Workshop on Emotion Awareness in Software Engineering*, SEmotion '18, page 14–19, New York, NY, USA, 2018. Association for Computing Machinery.

[49] Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[50] Tapajit Dey, Sara Mousavi, Eduardo Ponce, Tanner Fry, Bogdan Vasilescu, Anna Filippova, and Audris Mockus. Detecting and characterizing bots that commit code. In *Proceedings of the 17th International Conference on Mining Software Repositories*, MSR '20, page 209–219, New York, NY, USA, 2020. Association for Computing Machinery.

[51] Joao P. Diniz, Daniel Cruz, Fabio Ferreira, Cleiton Tavares, and Eduardo Figueiredo. Github label embeddings. In *2020 IEEE 20th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 249–253, 2020.

[52] GitHub Docs. Managing labels. https://docs.github.com/en/issues/using-labels-and-milestones-to-track-work/managing-labels, 2023.

[53] Gunther Eysenbach and James E Till. Ethical issues in qualitative research on internet communities. *BMJ*, 323(7321):1103–1105, 2001.

[54] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*, 2020.

[55] Gabriel Ferreira, Limin Jia, Joshua Sunshine, and Christian Kästner. Containing malicious package updates in npm with a lightweight permission system. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1334–1346, 2021.

[56] GitHub. Codeql: Semantic code analysis engine. https://codeql.github.com/.

[57] GitHub. Dependabot: Keep your dependencies up to date. https://github.com/dependabot.

[58] GitHub. Github marketplace. https://github.com/marketplace, 2017.

[59] GitHub. Privately reporting a security vulnerability. https://docs.github.com/en/code-security/security-advisories, 2023.

[60] Mehdi Golzadeh, Alexandre Decan, Damien Legay, and Tom Mens. A ground-truth dataset and classification model for detecting bots in github issue and pr comments. *Journal of Systems and Software*, 175:110911, 2021.

[61] Mehdi Golzadeh, Tom Mens, Alexandre Decan, Eleni Constantinou, and Natarajan Chidambaram. Recognizing bot activity in collaborative software development. *IEEE Softw.*, 39(5):56–61, September 2022.

[62] Yuejun Guo, Constantinos Patsakis, Qiang Hu, Qiang Tang, and Fran Casino. Outside the comfort zone: Analysing llm capabilities in software vulnerability detection. In *Computer Security – ESORICS 2024: 29th European Symposium on Research in Computer Security, Bydgoszcz, Poland, September 16–20, 2024, Proceedings, Part I*, page 271–289, Berlin, Heidelberg, 2024. Springer-Verlag.

[63] Gunnar Harboe and Elaine M. Huang. Real-world affinity diagramming practices: Bridging the paper-digital gap. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, page 95–104, New York, NY, USA, 2015. Association for Computing Machinery.

[64] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*, 2020.

[65] Jinchang Hu, Lyuye Zhang, Chengwei Liu, Sen Yang, Song Huang, and Yang Liu. Empirical analysis of vulnerabilities life cycle in golang ecosystem. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA, 2024. Association for Computing Machinery.

[66] Yiheng Huang, Ruisi Wang, Wen Zheng, Zhuotong Zhou, Susheng Wu, Shulin Ke, Bihuan Chen, Shan Gao, and Xin Peng. Spiderscan: Practical detection of malicious npm packages based on graph-based behavior modeling and matching. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, page 1146–1158, New York, NY, USA, 2024. Association for Computing Machinery.

[67] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[68] Jing Jiang, Qiudi Wu, Jin Cao, Xin Xia, and Li Zhang. Recommending tags for pull requests in github. *Information and Software Technology*, 129:106394, 2021.

[69] Label.dev. What is a minimal reproducible example (mre)? https://label.dev/articles/minimal-reproducible-example/#what-is-a-mre, 2023.

[70] Piergiorgio Ladisa, Serena Elisa Ponta, Nicola Ronzoni, Matias Martinez, and Olivier Barais. On the feasibility of cross-language detection of malicious packages in npm and pypi. In *Proceedings of the 39th Annual Computer Security Applications Conference*, ACSAC '23, page 71–82, New York, NY, USA, 2023. Association for Computing Machinery.

[71] Carlene Lebeuf, Margaret-Anne Storey, and Alexey Zagalsky. Software bots. *IEEE Software*, 35(1):18–23, 2018.

[72] Kaixuan Li, Jian Zhang, Sen Chen, Han Liu, Yang Liu, and Yixiang Chen. Patchfinder: A two-phase approach to security patch tracing for disclosed vulnerabilities in open-source software. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2024, page 590–602, New York, NY, USA, 2024. Association for Computing Machinery.

[73] Renee Li, Pavitthra Pandurangan, Hana Frluckaj, and Laura Dabbish. Code of conduct conversations in open source software projects on github. *Proc. ACM Hum.-Comput. Interact.*, 5(CSCW1), April 2021.

[74] Chengwei Liu, Sen Chen, Lingling Fan, Bihuan Chen, Yang Liu, and Xin Peng. Demystifying the vulnerability propagation and its evolution via dependency trees in the npm ecosystem. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, page 672–684, New York, NY, USA, 2022. Association for Computing Machinery.

[75] Yinhan Liu. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 364, 2019.

[76] Dave Lunny. Sane github labels. https://medium.com/@dave_lunny/sane-github-labels-c5d2e6004b63, 2016.

[77] Zeyang Ma, Shouvick Mondal, Tse-Hsun Peter Chen, Haoxiang Zhang, and Ahmed E. Hassan. Vulnet: Towards improving vulnerability management in the maven ecosystem. *Empirical Software Engineering*, 29, 06 2024.

[78] Samim Mirhosseini and Chris Parnin. Can automated pull requests encourage software developers to upgrade out-of-date dependencies? In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, ASE '17, page 84–94. IEEE Press, 2017.

[79] npm, Inc. npm: The package manager for javascript. https://www.npmjs.com/, 2010.

[80] Marc Ohm, Henrik Plate, Arnold Sykosch, and Michael Meier. Backstabber's knife collection: A review of open source software supply chain attacks. In Clémentine Maurice, Leyla Bilge, Gianluca Stringhini, and Nuno Neves, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 23–43, Cham, 2020. Springer International Publishing.

[81] Gabriel P. Oliveira, Ana Flávia C. Moura, Natércia A. Batista, Michele A. Brandão, Andre Hora, and Mirella M. Moro. How do developers collaborate? investigating github heterogeneous networks. *Software Quality Journal*, 31(1):211–241, September 2022.

[82] Gede Artha Azriadi Prana, Abhishek Sharma, Lwin Khin Shar, Darius Foo, Andrew E. Santosa, Asankhaya Sharma, and David Lo. Out of sight, out of mind? how vulnerable dependencies affect open-source projects. *Empirical Softw. Engg.*, 26(4), July 2021.

[83] Rohit Raj, Mridul Newar, and Mainack Mondal. "i just hated it and i want my money back": Data-driven understanding of mobile VPN service switching preferences in the wild. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 6021–6037, Philadelphia, PA, August 2024. USENIX Association.

[84] Matthew Rocklin. Minimal bug reports. https://matthewrocklin.com/minimal-bug-reports.html, 2018.

[85] Johnny Saldana. *The Coding Manual for Qualitative Researchers*. SAGE Publications, London, England, 3rd edition, 2015.

[86] Benjamin Saunders, Julius Sim, Tom Kingstone, Shula Baker, Jackie Waterfield, Bernadette Bartlam, Heather Burroughs, and Clare Jinks. Saturation in qualitative research: exploring its conceptualization and operationalization. *Quality & quantity*, 52:1893–1907, 2018.

[87] Adriana Sejfia and Max Schäfer. Practical automated detection of malicious npm packages. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, page 1681–1692, New York, NY, USA, 2022. Association for Computing Machinery.

[88] Jared Spool. Do users change their settings? https://archive.uie.com/brainsparks/2011/09/14/do-users-change-their-settings/, 2011.

[89] Margaret-Anne Storey and Alexey Zagalsky. Disrupting developer productivity one bot at a time. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, page 928–931, New York, NY, USA, 2016. Association for Computing Machinery.

[90] Xin Tan, Minghui Zhou, and Zeyu Sun. A first look at good first issues on github. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, page 398–409, New York, NY, USA, 2020. Association for Computing Machinery.

[91] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

[92] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[93] Hacker News User. Github's dependabot is causing a ton of "spam" in our frontend (angular) reposit... https://news.ycombinator.com/item?id=27929596, 2022.

[94] Hacker News User. Honestly dependabot is so bad i'm surprised anything other than the smallest pro... https://news.ycombinator.com/item?id=31963643, 2022.

[95] Jason Watson, Heather Richter Lipford, and Andrew Besmer. Mapping user preference to privacy default settings. *ACM Trans. Comput.-Hum. Interact.*, 22(6), November 2015.

[96] Mairieli Wessel, Bruno Mendes de Souza, Igor Steinmacher, Igor S. Wiese, Ivanilton Polato, Ana Paula Chaves, and Marco A. Gerosa. The power of bots: Characterizing and understanding bots in oss projects. *Proc. ACM Hum.-Comput. Interact.*, 2(CSCW), November 2018.

[97] Mairieli Wessel, Bruno Mendes de Souza, Igor Steinmacher, Igor S. Wiese, Ivanilton Polato, Ana Paula Chaves, and Marco A. Gerosa. The power of bots: Characterizing and understanding bots in oss projects. *Proc. ACM Hum.-Comput. Interact.*, 2(CSCW), November 2018.

[98] Mairieli Wessel and Igor Steinmacher. The inconvenient side of software bots on pull requests. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, ICSEW'20, page 51–55, New York, NY, USA, 2020. Association for Computing Machinery.

[99] Susheng Wu, Wenyan Song, Kaifeng Huang, Bihuan Chen, and Xin Peng. Identifying affected libraries and their ecosystems for open source software vulnerabilities. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA, 2024. Association for Computing Machinery.

[100] Yulun Wu, Zeliang Yu, Ming Wen, Qiang Li, Deqing Zou, and Hai Jin. Understanding the threats of upstream vulnerabilities to downstream projects in the maven ecosystem. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1046–1058, 2023.

[101] Zeliang Yu, Ming Wen, Xiaochen Guo, and Hai Jin. Maltracker: A fine-grained npm malware tracker copiloted by llm-enhanced dataset. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2024, page 1759–1771, New York, NY, USA, 2024. Association for Computing Machinery.

[102] Abdelrahman Zerouali, Tom Mens, Alexandre Decan, et al. On the impact of security vulnerabilities in the npm and rubygems dependency networks. *Empirical Software Engineering*, 27(107), 2022.

[103] Junan Zhang, Kaifeng Huang, Yiheng Huang, Bihuan Chen, Ruisi Wang, Chong Wang, and Xin Peng. Killing two birds with one stone: Malicious package detection in npm and pypi using a single model of malicious behavior sequence. *ACM Trans. Softw. Eng. Methodol.*, November 2024. Just Accepted.

[104] Lyuye Zhang, Chengwei Liu, Sen Chen, Zhengzi Xu, Lingling Fan, Lida Zhao, Yiran Zhang, and Yang Liu. Mitigating persistence of open-source vulnerabilities in maven ecosystem. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 191–203, 2023.

[105] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*, 2019.

A Details of npm packages

In this section, we provide additional details related to the dataset used in our study, including examples of popular npm packages from different buckets, as shown in Table 12. Furthermore, we present the set of tags identified as similar to *security* in Table 14, along with the distribution of tags per repository depicted in Figure 2.

B Security-issues identification model

Models considered to detect security-related issues: To classify issues into two classes (related to security or not related to security), we used the following models: BERT [49], RoBERTa [75], CodeBERT [54], FLAN-T5 [42] and DeBERTa [64] and LLMs namely Mistral [67], Qwen [38], Meta-LLaMA [92], and Gemma [91] using zero-shot and few-shot

Indegree of packages	Popular Javascript packages
0	@ever-co/faker
	@musicstory/react-bootstrap-table2-filter
1-10	@aurelia/scheduler-dom
	@tanstack/svelte-table
10-100	@ckeditor/ckeditor5-vue
	create-vite
100-500	@medusajs/medusa
	gatsby-plugin-manifest
500-1000	@types/chalk
	markdown
>1000	ethers
	shelljs

Table 12: Popular packages in different buckets.

Issue age	# issues created
3 months	35284
6 months	60082
9 months	60692
1 year	71675
2 years	253022
3 years	253703
> 3 years	883280

Table 13: Distribution of issues based on their creation time.

under ICL settings. Since the task is a sequence classification task and hence, we used different architectures of models described in Table 27 along with the specification of model sizes. The hyperparameter tuning was done using Optuna [36], which is a framework for optimization for hyperparameters. The performance of the models is shown in Table 5.

We extracted and averaged attention scores from the [CLS] token across all heads in the final layer of the fine-tuned RoBERTa model to identify influential tokens, which are visualized in the word cloud below (see Figure-3) after removing stopwords and artifacts.

Error Analysis of RoBERTa: Since we achieved the best accuracy for the downstream task using the fine-tuned RoBERTa model. Class-wise performance of the RoBERTa model is shown in Table 8. We also tried to get insights into the performance of the model using error analysis and observed that the issue body often lacks sufficient information, with references to other issues by their numbers instead of detailed elaboration. Additionally, while discussions occasionally address relevant topics, they often lack the expected focus on security-related context. There is also a noticeable drift between issue titles, which focus on security, and the corresponding issue bodies, which do not align with this context, further complicating accurate classification.

Performance of LLMs: For LLM evaluation, task-specific prompts were crafted to address the classification task. Zero-shot settings relied on these prompts without additional fine-tuning, while few-shot ICL incorporated a small number of annotated examples into the prompts to provide better context. The prompts for zero-shot and few-shot evaluations are detailed in Table 18 and the specific LLM models evaluated are described in Table 16. Model outputs were assessed using a BERT-based similarity scoring approach [105] with the DeBERTa [64] model, comparing predictions against reference sentences, *security* and *non-security*. The evaluation results are given in Table 15.

Tags
category: security
type: security
security
security (category)
theme: security
severity6: security
:label: security
pr: security
:fire: security
area: security
issue: security
[type] security
security :exclamation:
topic: security
cat: security
severity: security
subj: security
security vulnerability
3: security review
security fix
impact/security
limitation: security restriction
mend: dependency security vulnerability
security blocker
external: lightning web security locker

Table 14: Set of tags that are found similar with the term *security* using Word2vec.

Model		Accuracy	Macro avg F1
Mistral	Zero Shot	0.82	0.82
	Few Shot	0.83	0.83
Qwen	Zero Shot	0.52	0.39
	Few Shot	0.67	0.66
LLaMA	Zero Shot	0.65	0.63
	Few Shot	0.78	0.77
Gemma	Zero Shot	0.60	0.53
	Few Shot	0.54	0.50

Table 15: Performance of large language models under zero-shot and few-shot ICL settings for classifying issues into security-related and non-security-related categories.

Model	Model Specification	Task Type
Gemma	google/gemma-2b-it	Text Classification
Mistral	mistralai/Mistral-7B-Instruct-v0.3	Text Classification
Qwen	Qwen/Qwen2.5-1.5B-Instruct	Text Classification
Llama	meta-llama/Llama-3.2-3B-Instruct	Text Classification

Table 16: Specification of the LLMs used for classification of security-related and non-security related issues under zero-shot and few-shot ICL settings.

C Theme identification model

For the identification of themes in reviews, we used the following models: BERT [49], RoBERTa [75], CodeBERT [54], FLAN-T5 [42] and DeBERTa [64]. Since the task can be modelled as both sequence multi-label classification or sequence generation task, hence, we used different architectures

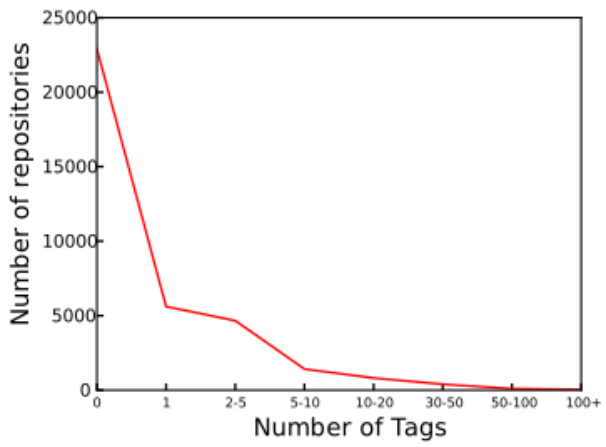

Figure 2: Distribution of repositories across different tag count ranges.

Figure 3: Wordcloud of most influential tokens identified using [CLS] token attention scores from the final layer of our fine-tuned RoBERTa model.

of models described in Table-27 along with the specification of model sizes. The hyperparameter tuning was done using Optuna [36], which is a framework for optimization for hyperparameters.

Accuracy of models: We found that RoBERTa model outperformed other models into consideration, identifying more than 80% of themes correctly in the reviews as shown in Table-7 which reports the accuracy of the models on validation set.The result of the accuracy of RoBERTa model against additional manual ground truth is reported in Table-20 .

Error Analysis of RoBERTa: Since the fine-tuned RoBERTa model predicted the highest fraction of themes correctly, we did further analysis to look into the performance of the model. Classwise performance of the RoBERTa model is shown in Table-26. Although recall for "Successfully resolved" is lower, the F1-score (0.68) shows a balanced performance, suggesting accurate predictions; our high-precision model thus provides a conservative lower-bound estimate for this theme. We also tried to get insights into the performance of the model using error analysis and observed that the model mostly commits mistakes in predicting less frequently occurring classes like

		Predicted	
		security	non security
Ground Truth	security	4	3
	non security	4	89

Table 17: Confusion matrix for our DeBERTa text classifier.

speaking against the issue/not interested in the issue. We also noted that the model committed mistakes in understanding the contextual meaning of phrases and made predictions using the phrases themselves.

D Temporal Distribution of Issues: Categorization by Creation Time

Distribution of issues based on their creation time, categorizing the number of issues created within specific time frames : 13. Also the five most frequently mentioned CVE IDs (strictly CVE-IDs are mentioned here) are in the Table 28.

E Details related to bots

An overview of various activities performed by bots in user-reported issues, along with their frequency in security-related issues, is provided in Table 21. Additionally, the classification of bots involved in user-reported security issues is detailed in Table 22. A specific categorization of bot functionalities is further described in Table 30. We also disclose a summary classification of 40 publicly available bots based on their implementation type. The Table 29 illustrates the range of bot types identified in our analysis.

F GLMM results

We present three tables showcasing the results of the Generalized Linear Mixed Model (GLMM) analysis. These tables 31, 32, 33 summarize the statistical outputs

Setting	Prompt Template	Description

Zero-shot	{
	"role": "user",
	"content": f"classify **WITHOUT EXPLANATION** whether the given issue is related to security or not. Issue: {new_issue}"
	}
		Direct prompt asking the model to classify the issue without providing any explanation.

Few Shot	{
	"role":"user", "content": "classify **WITHOUT EXPLANATION** whether the given issue is related to security or not. Issue: {issue_0}"
	},
	{
	"role":"assistant", "content": "security"
	},
	{
	"role":"user", "content": "classify **WITHOUT EXPLANATION** whether the given issue is related to security or not. Issue: {issue_1}"
	},
	{
	"role":"assistant", "content": "non security" },
	...
	{
	"role":"user", "content": "classify **WITHOUT EXPLANATION** whether the given issue is related to security or not. Issue: {new_issue}"
	}
		Few-shot prompt providing two examples each for "security" and "non-security" classifications alternatively to give the model contextual guidance before asking it to predict for the current issue.

Table 18: Prompts for zero-shot and few-shot under ICL settings for classifying issues into security-related and non security-related categories.

Labelled as security		Model predicted	
Issue Title	**Issue Body**	**Issue Title**	**Issue Body**
Webhook notification channel password field is in plain text	What would you like to be added: It would be ideal if the password field for the webhook notification channel was hidden and not exposed in plain text. Why is this needed:Anyone who has access to set up notification channels can see the password which raises a security concern for the application we send the alert to.	fix: remove unnecessary logging for webauthn auth	When authenticating using Webauthn it is unnecessary to log Log in on.... This PR hides that logging in that case.
Set Cache-control: no-cache for full page requests	All API requests already set Cache-control: no-cache to avoid browsers caching sensitive data. The full page requests, however, does not which means that all data in window.grafanaBootData can be cached by the browsers.This means that browsers can cache some data even if the user logged out.	Migrate Lambda Trigger is not triggering if USER_PASSWORD_AUTH is done on the server/backend...	Describe the bug If you use the InitiateAuth with the USER_PASSWORD_AUTH as the authflow, the Migrate Lambda Trigger will not work... I have a console.log in the lambda and it is not working. However, if I copy-paste the exact code on the browser-side, the trigger will work... SDK version number @aws-sdk/client-cognito-identity-provider@3.294.0 Which JavaScript Runtime is this issue in? Node.js Details of the browser/Node.js/ReactNative version v16.19.0
XS deep freeze conflicts with SES security constraints	The XS Object.freeze takes a second optional boolean parameter that, if truthy, causes some form of transitive freezing. But unlike harden, it does more freezing than user code can do (and inspired thepetrify notion we're still designing). As currently implemented,this enhanced Object.freeze can be used for attack.	[SECURITY] npm i logs bearer token in case there is a formatting issue.	Is there an existing issue for this?I have searched the existing issues This issue exists in the latest npm versionI am using the latest npm Current BehaviorAccidentally providing a misformed token will print the bearer token to the log-output. I wasn't sure if this is indeed a security risk but I figured it might not hurt to point it out in case it is. Please close the issue right away if this is not critical. Expected Behavior Do not print any bearer tokens to standard output.
Anonymous users are not redirected to login page after trying to access a checkout step route since 3.0.0	Old checkout bug behavior since Spartacus version 3.0.0 Checkout auth guard is not redirecting anonymous users that has no active cart id. They get stuck in a blank page if they were to try to visit the checkout route directly	Fix for SSID and passwords starting with 0x	By default string starting with 0x are parsed as hexadecimal numbers by yargs. This fix allows to configure devices with an SSID starting with the string 0x
Disable localhost: snaps outside of Flask	localhost:snaps are not something that we ever want to support in stable MetaMask, and we should ensure that they are disabled in stable distributions. Outcome: Snaps that are fetched from localhost should only be available in Flask distribution. The user should be notified that this snap is a debug one and will not be loaded.	fix(lib-storage): S3 Upload can corrupt data from readable stream	Issue In some circumstance Upload miss some data from the readable stream and corrupt the uploaded data. This has been observed in a real situation when serializing data and uploading data on the fly by using a PassThrough between the serialization and the S3 Upload. The issue has been located in lib/storage/src/data-chunk/readable-helper.ts Description In readable-helper.ts the pause/resume mechanism has a concurrency flaw between line

Table 19: Examples of issues labelled as security and issues that our model predicted to be security related.

	% themes in validation dataset	% themes in additional random sample
Correctly Predicted	82%	76%
Incorrectly predicted	18%	24%

Table 20: Performance of fine-tuned RoBERTa model (For theme classification) on validation dataset and additional random sample (with manual ground truth labels).

Activities	Frequency
comment	2826
labeled	2197
unlabeled	1079
milestoned	294
locked	287
closed	233
referenced	182
review_requested	129
deployed	63
demilestoned	45
subscribed	19
merged	19
head_ref_deleted	19
mentioned	16
removed_from_merge_queue	4
head_ref_force_pushed	3
added_to_project	3
moved_columns_in_project	2
automatic_base_change_succeeded	2
reopened	2
renamed	2
review_dismissed	1
convert_to_draft	1
review_request_removed	1
assigned	1

Table 21: Overview of the various activities performed by bots in user-reported issues along with the count of their occurrence in security-related issues.

Bot Type (L1)	Bot Type (L2)	Bot Type (L3)	No of Bots
Hosted as GitHub App	Public Apps	Public Source Codes	40
		Source Code not found	17
	Private Apps		14
Hosted as normal GitHub User accounts			49

Table 22: Classification of 120 GitHub bots in user-reported security issues based on their deployment and accessibility.

Theme	Hierarchy	F1	F2
A. Issue with solution(PR)	L1	303	826,030
A.1 Description present	L2	194	786,818
A.2 No description *	L2	109	39,212
B. Issue without solution	L1	225	791,368
B.1 Reproducibility	L2	102	418,668
B.2 Non-reproducibility	L2	123	372,700

Table 23: The hierarchy of themes derived from open coding of reviews in the category of "creation". Here, F1 refers to the number of quotes in which this theme appeared in our manual analysis and F2 refers to the number of quotes in which the theme was identified by the automated approach.* We filtered the case of Issues with solution but no description logically and did not feed to the model.

Theme	Hierarchy	F1	F2
A. Acknowledged	L1	400	1,981,510
A.1 Spoke against with issue	L2	91	194,971
A.2 Spoke for the issue	L2	309	1,786,539
B. Ignored	L1	150	486,081
B.1 Not interested	L2	46	133,035
B.2 Inconclusive	L2	104	353,046

Table 24: The hierarchy of themes derived from open coding of reviews in the category of "discussion". Here, F1 refers to the number of quotes in which this theme appeared in our manual analysis and F2 refers to the number of quotes in which the theme was identified by the automated approach.

Theme	Hierarchy	F1	F2
A. With valid reason	L1	497	1,281,922
A.1 Falsely Created	L2	60	73,670
A.2 Successfully resolved	L2	260	727,861
A.3 To be completed	L2	177	480,391
B. Without valid reason	L1	172	116,418
B.1 Closed without reason **	L2	61	79,188
B.2 Completed due to staleness	L2	111	37,230

Table 25: The hierarchy of themes derived from open coding of reviews in the category of "resolution". Here, F1 refers to the number of quotes in which this theme appeared in our manual analysis and F2 refers to the number of quotes in which the theme was identified by the automated approach.**We filtered the case of Closed without any reason logically and did not feed to the model.

Type	L2 levels	Recall	F1-score
Creation	Reproducibility	0.93	0.79
	Non reproducibility	0.79	0.87
	Description present	0.97	0.93
Discussion	Spoke for the issue	0.93	0.91
	Spoke against with issue	0.6	0.67
	Not interested	0.60	0.60
	Inconclusive	0.81	0.87
Resolution	Completed due to staleness	1.00	0.98
	Falsely Created	0.62	0.70
	Successfully resolved	0.53	0.68
	To be completed	0.94	0.72

Table 26: Class-wise performance of RoBERTa model on theme identification task on validation dataset.

Model	Model Specification	Classification of issues	Theme Identification
CodeBERT	CodeBERT-base	Sequence classification	Sequence Classification (With multiple classification heads)
BERT	BERT-base (Uncased)	Sequence classification	Sequence Classification (With multiple classification heads)
RoBERTa	RoBERTa-base	Sequence classification	Sequence Classification (With multiple classification heads)
Flan-T5	Flan-T5-base	Seq2Seq	Seq2Seq
DeBERTa	DeBERTa-v3-base	Sequence classification	Sequence Classification (With multiple classification heads)

Table 27: Specification and architecture of models used in classification of issues and for theme identification.

CVE- id	Title	Description
CVE-2023-45857	Axios Cross-Site Request Forgery Vulnerability	An issue discovered in Axios 0.8.1 through 1.5.1 inadvertently reveals the confidential XSRF-TOKEN stored in cookies by including it in the HTTP header X-XSRF-TOKEN for every request made to any host allowing attackers to view sensitive information.
CVE-2021-23337	Command Injection in lodash	lodash versions prior to 4.17.21 are vulnerable to Command Injection via the template function.
CVE-2019-10744	Prototype Pollution in lodash	Versions of lodash before 4.17.12 are vulnerable to Prototype Pollution. The function defaultsDeep allows a malicious user to modify the prototype of Object via {constructor: {prototype: {...}}}causing the addition or modification of an existing property that will exist on all objects.
CVE-2021-3807	Inefficient Regular Expression Complexity in chalk/ansi-regex	ansi-regex is vulnerable to Inefficient Regular Expression Complexity which could lead to a denial of service when parsing invalid ANSI escape codes.
CVE-2020-28469	glob-parent vulnerable to Regular Expression Denial of Service in enclosure regex	This affects the package glob-parent before 5.1.2. The enclosure regex used to check for strings ending in enclosure containing path separator

Table 28: Top 5 most frequently found CVE IDs from the un-identified security issues.

Bots Type	Description	Example of bots (Functionality)
Rule based (34 bots)	Implements deterministic logic based on predefined conditions and static rules.	mergify (PR management), google-cla (Policy & CLA), gatsby-cloud(CI/CD)
AI/ML based (4 bots)	Uses learning-based models to adapt behavior from data rather than fixed rules.	dotnetissuelabeler(issue labelling), coderabbitai(pull request summarizer)
In Beta stage (2 bots)	Includes experimental AI/ML components still under development	sonarcloud (codefix, codereviews), codecov (coverage of code)

Table 29: Classification of 40 publicly available bots based on implementation type, along with brief descriptions of their underlying logic.

L1	L2	Description
Dependency Management		Bots that automate the management and updates of project dependencies.
PR Management		Bots that handle the entire pull request lifecycle, from creation and review to merging and closure.
CI/CD		Bots that handle continuous integration and deployment pipelines for codebases.
Security		Bots focused on identifying vulnerabilities and ensuring code security.
Utility	Policy & CLA	Bots that enforce project policies or manage Contributor License Agreements (CLA).
	PR Related	Bots that automate specific tasks within the pull request process, such as labeling, commenting, or checking for conflicts.
	Project Management	Bots that streamline task tracking, milestone planning, and overall project coordination.
	Issue Related	Bots that assist in tracking, labeling, or resolving issues efficiently
	Code Coverage	Bots that analyze and report on code test coverage metrics
	Miscellaneous Utility	Bots offering diverse functionalities that extend beyond specific predefined categories.
Miscellaneous		Bots that do not fit into the predefined categories and serve diverse purposes

Table 30: Description of each class of bot classification.

Factors	**Category**	**O.R.**	**C.I.**	**p-value**
Presence of CVE	-	0.707	[0.680, 0.735]	$< 2 \times 10^{-16}$
Number of Comments	0-10	0.127	[0.109, 0.147]	$< 2 \times 10^{-16}$
	10-25	0.363	[0.312, 0.423]	$< 2 \times 10^{-16}$
	25-50	0.568	[0.486, 0.664]	1.18×10^{-12}
	50-80	0.748	[0.624, 0.898]	1.82×10^{-3}
Reproducibility of the issue	-	2.213	[2.196, 2.231]	$< 2 \times 10^{-16}$

Table 31: Results of Generalized Linear Mixed Model(glmm) examining the factors for time to close security related issues.

Factors	**Category**	**O.R.**	**C.I.**	**p-value**
Presence of CVE	-	0.775	[0.688, 0.873]	2.83×10^{-5}
Number of Comments	0-10	2.256	[1.307, 3.892]	3.472×10^{-3}
	10-25	2.561	[1.482, 4.425]	7.48×10^{-4}
	25-50	2.088	[1.197, 3.643]	9.51×10^{-3}
Number of weekly downloads	50K-100K	1.619	[1.209, 2.168]	1.216×10^{-3}
Reproducibility of the issue	-	5.060	[4.931, 5.191]	$< 2 \times 10^{-16}$
Involvement of Bots	-	14.602	[14.192, 15.023]	$< 2 \times 10^{-16}$

Table 32: Results of Generalized Linear Mixed Model(glmm) examining the factors for staleness of security related issues.

Factors	**Category**	**O.R.**	**C.I.**	**p-value**
Presence of CVE	-	0.816	[0.786, 0.848]	$< 2 \times 10^{-16}$
Number of Comments	10-25	0.781	[0.669, 0.912]	1.74×10^{-3}
Number of weekly downloads	0K-10K	1.267	[1.225, 1.311]	$< 2 \times 10^{-16}$
	10K-50K	1.073	[1.027, 1.121]	1.65×10^{-3}
Reproducibility of the issue	-	0.419	[0.415, 0.423]	$< 2 \times 10^{-16}$
Involvement of Bots	-	0.682	[0.677, 0.687]	$< 2 \times 10^{-16}$

Table 33: Results of Generalized Linear Mixed Model(glmm) examining the factors for successful resolution of security related issues.

Factor considered	**Type**	**Description**
CVE mention	Boolean	Refers to mention of CVE/CWE in the issue
Number of comments	Categorical	Number of comments on the issue categorised into 5 classes: i) 0-10, ii) 10-25, iii) 25-50, iv) 50-80, v) >80
Weekly downloads	Categorical	No. of weekly downloads of the package categorised into 5 classes: i) 0-10K, ii) 10K-50K, iii) 50K-100K, iv) 100K-200K, v) >200K
No. of active maintainers in the past year	Categorical	No. of active maintainers of the repository in the last one year, categorised into 5 classes: i) 0-10, ii) 10-50, iii) 50-100, iv) 100-200, v) >200
Reproducibility	Boolean	If the issue body has explicit instructions on reproducing the error like code snippets, step-by-step instructions or error logs etc.
Bot Involvement	Boolean	If bot is involved in the discussion or closing of the issue

Table 34: Description of factors considered in GLMM models.