Beyond CVEs: Untracked Vulnerabilities in Public Issue Trackers

Mădălin Simion
madalin.simion2000@gmail.com
TU Delft
Delft, The Netherlands

Max van der Horst
m.h.vanderhorst@tudelft.nl
TU Delft
Delft, The Netherlands

Stan Plasmeijer
s.plasmeijer@divd.nl
Dutch Institute for Vulnerability Disclosure
The Hague, The Netherlands

Yury Zhauniarovich
y.zhauniarovich@tudelft.nl
TU Delft
Delft, The Netherlands

Abstract

Software vulnerabilities – particularly in open-source software (OSS) components, which are now embedded in nearly every application – pose major security and privacy risks. Existing tools and research focus largely on vulnerabilities listed in the Common Vulnerabilities and Exposures (CVE) system, leaving those without CVE identifiers often overlooked. In this study, we aim to estimate the number of such CVE-less vulnerabilities in OSS projects by systematically analyzing project issue trackers to identify security-related reports that could qualify as CVEs. We use an AI-human collaborative approach, combining AI-based issue pre-filtering with expert validation to efficiently and accurately estimate the prevalence of these overlooked vulnerabilities.

We closely examined four large C++ projects and found that approximately 1.55% of all reported issues were classified by our model as security-related. Expert validation performed by the CVE Numbering Authority (CNA) Administrator on the gRPC project revealed that about 22% of these predicted security-related issues correspond to real, previously untracked vulnerabilities. This number is nearly five times greater than the total number of CVEs listed for this project in the National Vulnerability Database (NVD). These results reveal a gap in today's vulnerability disclosure ecosystem: many vulnerabilities are publicly disclosed in issue trackers yet never formally communicated through the CVE program, leaving them largely unexplored and potentially unaddressed.

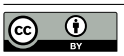
CCS Concepts

• **Security and privacy** → **Software security engineering**; **Vulnerability management**; • **Information systems** → *Open source software*.

Keywords

Open-Source Software; CVE-less Vulnerabilities; Untracked CVEs; Human-AI Collaboration

ACM Reference Format:
Mădălin Simion, Max van der Horst, Stan Plasmeijer, and Yury Zhauniarovich. 2026. Beyond CVEs: Untracked Vulnerabilities in Public Issue Trackers. In *19th European Workshop on Systems Security (EuroSec '26), April 27–30, 2026, Edinburgh, Scotland Uk.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3803525.3804993



EuroSec '26, Edinburgh, Scotland Uk

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/3803525.3804993

1 Introduction

The modern software ecosystem heavily relies on open-source software (OSS), with 97% of codebases including OSS components [8]. Given this widespread adoption, ensuring OSS security is essential. However, in practice, 86% of analyzed codebases contain at least one known vulnerability in their OSS dependencies [8]. While these statistics focus on known and registered vulnerabilities, unregistered vulnerabilities, i.e., those not tracked by systems such as CVE, receive significantly less attention. For example, Jiang et al. [21] conducted a systematic literature review of 82 studies on vulnerability prioritization and found that 70 focused exclusively on CVE-numbered vulnerabilities. The authors further note that only vulnerabilities listed in the NVD database form the foundation of most existing vulnerability prioritization frameworks.

At the same time, emerging evidence highlights the huge impact of untracked vulnerabilities. A recent report indicates that 70% of exploits observed in 2023 had no associated CVE number [5]. Similarly, VulnCheck contributed more than 100 new CVEs by auditing exploitation evidence, suggesting that untracked vulnerabilities represent a substantial and underrecognized attack vector [16].

To better understand the scope of this largely hidden threat, this work poses the following research question – *What is the extent of untracked vulnerabilities – those lacking a CVE identification number – in open-source projects?* – and takes a step toward answering it. Specifically, we analyze open-source project issue trackers, aiming to identify security-related reports that could plausibly qualify for CVE assignment. Our approach relies on an AI-human collaboration model: an AI system first performs a large-scale pre-filtering of issues to surface those with potential security relevance, after which an expert assesses whether these issues constitute actual vulnerabilities. This combined methodology enables us to estimate how many vulnerabilities are publicly disclosed in development platforms yet remain absent from centralized vulnerability databases, thereby revealing the scope of underreported or untracked security risks.

In this study, we focus on C and C++ projects for three reasons. First, they remain highly relevant, consistently ranking second and third behind Python on the TIOBE Index (as of February 2026) [35]. Second, they underpin critical, widely used projects like TensorFlow [3], React Native [2], and OpenCV [1], meaning vulnerabilities can have broad industry consequences. Finally, C and C++

are uniquely prone to severe security flaws, particularly memory-related issues [38]. While their fine-grained memory management is powerful, it frequently leads to complex bugs [29]; notably, Ray et al. found that C++ codebases on GitHub exhibit a higher rate of security issues per line of code than many other languages [30].

We applied our approach to four large C++ open-source projects – ClickHouse, OpenCV, Bitcoin, and gRPC. Using a classifier trained on issues from an unrelated project (Chromium), we identified between 0.79% and 2.40% of all issues as security-related, with ClickHouse exhibiting the lowest proportion (0.79%) and Bitcoin the highest (2.40%). In total, across all four projects, the model flagged 640 *out of* 41,181 *(1.55%) issues* as security-relevant. The detailed review of 232 issues related to the gRPC project, conducted by the CNA Administrator, revealed that 52 issues (22.4%) could plausibly qualify for a CVE identifier, which corresponds to 0.5% of all analyzed issues. At the same time, the NVD currently lists only 11 CVEs for the gRPC project, and none of these correspond to the potential CVEs flagged by the expert. Thus, our approach identified nearly five times more potential vulnerabilities than are currently tracked. However, it should be noted that the majority of these potential vulnerabilities would likely receive a low CVSS score. Interestingly, only 3 out of 52 issues identified by our model as security-related were labeled as such by project maintainers, highlighting a gap in the existing labeling process and suggesting for more automated approaches. In summary, this paper makes the following three main contributions:

(1) It provides an **empirical estimate** of prevalence of CVE-less vulnerabilities disclosed through public issue trackers.
(2) It evaluates the **effectiveness of transformer-based models** for large-scale security issue detection under realistic operational constraints.
(3) It exposes **a structural visibility gap** between open-source development platforms and the global vulnerability disclosure ecosystem, with direct implications for vulnerability management and disclosure governance.

2 Background and Related Work

Software vulnerabilities and the CVE ecosystem. Bugs reported in software projects can be *security-related* or *non-security* issues. Security-related issues may correspond to actual vulnerabilities or to non-vulnerability matters such as security feature requests. Vulnerabilities can further be classified as detected or undetected [31]. Detected vulnerabilities may be published in issue trackers or remain unpublished. Among published vulnerabilities, some receive CVE identifiers and are broadly *tracked*, while others remain *untracked* and appear only in project-specific issue trackers.

The CVE Project [33] plays a central role in vulnerability tracking and communication by providing stable identifiers allowing to reference the same issue. Most security tools and advisory systems rely on CVE identifiers for detection, prioritization, and automated risk management. Consequently, vulnerabilities without CVE identifiers often remain absent from scanners, security dashboards, and coordinated response workflows. CVE records are typically accompanied by Common Vulnerability Scoring System (CVSS) base scores [14], which standardize severity assessment and support remediation decisions.

Vulnerability detection from codebase. Automated vulnerability detection has traditionally focused on analyzing source code. For example, Li et al. [22] proposed detecting vulnerabilities by learning from semantically related code fragments using deep neural architectures. Ziems and Wu [40] proposed treating source code as text, applying transformer-based models to identify vulnerabilities in C source code files, achieving accuracy of over 93%.

Security issue classification. Ghosh et al. [18] demonstrated that textual defect reports often contain sufficient context to identify unlabeled vulnerabilities, validating the use of text-based machine-learning approaches. Prior work explored automated identification of security bug reports in large issue trackers. Gegick et al. used text mining to classify reports in an industrial setting and showed that it can be effective in distinguishing security-related issues from other types of defects [17]. Later studies refined these early approaches, addressing challenges such as class imbalance and noisy labels. Peters et al. proposed FARSEC, which filters and ranks reports before classification to improve performance under realistic conditions where security issues are rare and inconsistently labeled [28]. Zheng et al. further improved classification by incorporating domain knowledge from CWE and CVE sources, achieving better results on benchmark datasets [36, 39]. More recently, transformer-based models have advanced bug report categorization by capturing contextual and semantic relationships in text. Meher et al. showed that transformers outperform traditional machine-learning methods in multi-class bug classification tasks [25]. Thus, related work demonstrates that automated techniques can identify security-related reports and that issue trackers hold valuable signals for vulnerability discovery. Yet, most studies focus on classification and text-based detection, leaving the link between detected issues and formal vulnerability tracking largely unexplored.

Chromium issue datasets. The Chromium project has been widely used as a benchmark for evaluating security issue classification models because it provides a large, publicly available issue tracker with historical labels that identify security-related reports. Wu et al. [36] introduced a curated dataset based on these reports and used it to compare a range of machine learning approaches. Several later studies, including Zheng et al. [39] and Alqahtani et al. [6], adopted the same dataset to ensure comparability across models. This makes the Chromium dataset a natural reference point for assessing model performance before applying a classifier to larger or more diverse issue trackers.

3 Methodology

To answer the research question, we employed the methodology illustrated in Figure 1, consisting of three main stages. First, we develop an ML-based system to identify security-related issues. Second, we employ this system to identify security-related issues in projects whose data were not used for training. Finally, the security expert evaluates whether the identified security-related issues actually correspond to security vulnerabilities and assesses their severity. Below, we examine the steps of these stages in detail.

3.1 Stage I: Security Issue Identification System

In the field of software development, issue (or bug) classification plays a crucial role in supporting a wide range of activities. In

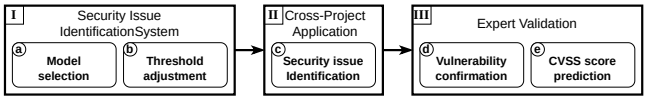


Figure 1: Methodology outline.

this study, we classify all issues as either *security-related* or *non-security-related*. This broad categorization increases the coverage of issues that may have even a slight chance of being identified later as vulnerabilities. This trade-off is acceptable, as all security-related issues undergo manual verification during the third stage.

To develop the system, we employed a publicly available Chromium dataset [36] and a curated version of the smaller one introduced in 2011 [4]. Both datasets contain bug descriptions collected between August 2008 and June 2010. We preprocessed the data by removing distinctive text that could bias the model, such as long URLs or issue numbers. In addition, we converted all text to lowercase and stripped out any HTML code. The dataset is imbalanced: it includes only 808 security-related issues, representing 1.93% of the total. To ensure a fair basis for comparing the models, we performed a stratified split into *train* and *test* datasets, with the test set containing 100 security-related issues and 5,207 non-security-related ones. To obtain unbiased evaluation metrics, we further balanced the *train* dataset, resulting in 1,416 issues in total: 708 security-related and 708 non-security-related.

Step ⓐ: Model selection. In this work, we experimented with five transformer-based models using two distinct techniques. The first technique employed few-shot prompting of two general-purpose (instruction-tuned) Large Language Models (LLMs): Gemma (`google/gemma-7b-it`) [7] and Mistral (`mistralai/Mistral-7B-Instruct-v0.2`) [26]. The second technique involved fine-tuning pretrained transformer models, specifically three BERT-based architectures: Bidirectional Encoder Representations from Transformers (BERT) [12], the Robustly Optimized BERT Approach (RoBERTa) [24], and Decoding-enhanced BERT with Disentangled Attention (DeBERTa) [20]. We specifically selected lightweight models, as they can be trained, fine-tuned, and deployed with limited hardware resources, making them a more realistic choice for integration into real-world OSS development workflows.

The prompt for general-purpose models consisted of the initial user instruction, the same six issue descriptions from the Chromium dataset (three security-related and three non-security-related from the train dataset) and the issue to classify. We used the following initial user instruction: *Given the following bug description from a C++ project, classify the bug as either "security" or "non-security".*

For training the BERT-based models, we used pre-trained default models from HuggingFace[1] and fine-tuned them on Google Colab[2] using an NVIDIA T4 GPU with 16 GB of GDDR6 memory, which is optimized for both training and inference of machine learning models [11]. We used the tokenizer associated with each model to preprocess the text, handling tokenization, token-to-ID conversion, and padding or truncation to a uniform sequence length of 512 tokens. The models were trained with gradient accumulation and half-precision (FP16) arithmetic. Training was monitored using a custom metrics function that computes accuracy, precision, recall, and F1 score. We applied early stopping to prevent overfitting, ensuring the model did not continue training beyond the point of diminishing returns. We used an early stopping based on accuracy, which is a fair choice given that the used dataset is balanced. The best model was saved and later used in further experiments.

Step ⓑ: Threshold adjustment. The threshold of a model determines the probability above which a predicted outcome is classified as positive. This threshold controls the precision-recall tradeoff, which is particularly important in security analysis where false positives and false negatives carry different costs. Higher precision reduces the effort spent validating erroneous alerts, while higher recall minimizes the risk of missing vulnerabilities. The chosen threshold thus reflects the balance between operational workload and the risk of undetected security flaws. In our context, a high-recall, low-precision model may capture most security issues but generate an impractical number of false positives, exhausting the limited capacity of security experts.

To manage the precision-recall tradeoff, we apply post-training threshold adjustment. After training the model, we analyze the distribution of predictions and modify the default classification threshold to achieve more optimal performance metrics. By tuning this threshold, we can reduce the number of false positives to a manageable level while still capturing the majority of security issues, ensuring that the resulting set of flagged vulnerabilities remains reasonable for expert validation.

3.2 Stage II: Cross-Project Application

Step ⓒ: Security issue identification. One of the major challenges in security issue classification is ensuring models' cross-project applicability. The literature generally agrees that cross-project deployment often results in a significant decline in performance [10]. However, it is unrealistic to expect open-source project maintainers to create high-quality, project-specific datasets and train models on their own data, especially for smaller projects. Therefore, this study adopts a more realistic approach: using a single, pre-trained model to predict security issues across multiple projects. Such an approach addresses real-world limitations by eliminating the need for project-specific training while still enabling identification of security-related issues across diverse codebases.

For our study, we selected projects based on three criteria: (1) they are developed in C/C++ to match the Chromium-based training data, as recommended by Gegick et al. [17]; (2) they have a substantial number of assigned CVEs to enable analysis of tracked vs. untracked vulnerabilities; and (3) they are independent of Chromium to avoid training bias. For instance, Electron was excluded due to its close ties to Chromium. For each selected project, we collected all issues but included only the initial submission, omitting subsequent comments or discussions to ensure consistency and evaluate the model using only the information available at creation.

3.3 Stage III: Expert Validation

Our primary goal is to identify potential untracked security vulnerabilities reported as issues in open-source projects. However, not all security-related issues correspond to actual vulnerabilities. To address this, we involved a security expert to evaluate all issues

[1] https://huggingface.co/
[2] https://colab.research.google.com/

Table 1: Performance of the Evaluated Models

Model	Accuracy	Precision	Recall	F1-score
Gemma	**0.97**	0.06	0.40	0.11
Mistral	0.91	0.06	0.80	0.12
BERT	0.95	0.27	0.97	0.42
RoBERTa	0.93	0.22	0.97	0.35
DeBERTaV3	0.94	**0.30**	**0.98**	**0.45**

Table 2: DeBERTaV3 Performance Across Threshold Levels

Threshold	Accuracy	Precision	Recall	F1-Score
0.50	0.9795	0.4778	0.9700	0.6403
0.70	0.9842	0.5460	0.9500	0.6934
0.80	0.9864	0.5864	0.9500	0.7252
0.90	0.9894	0.6528	0.9400	0.7705
0.95	0.9928	0.7460	0.9400	0.8319
0.98	0.9962	0.9082	0.8900	0.8990

flagged by the model as security-related. This validation step is essential for distinguishing issues related to vulnerabilities from other security-related ones. To ensure high-quality validation, we recruited a security expert who is also a CVE Numbering Authority (CNA) Administrator. The expert regularly evaluates CVE identifier requests for zero-day vulnerabilities and is well versed in the community rules governing CVE eligibility, such as the requirement that vulnerabilities be independently fixable [34].

Step ⓓ: Vulnerability confirmation. The expert was provided with a list of issues flagged by the model as security-related, each including a direct link to the corresponding issue discussion page. For each issue, the expert carefully reviewed the full discussion, including comments, related issues and pull requests. Based on this information, the expert determined whether the issue met the criteria to be classified as a CVE. This process ensured an accurate assessment, distinguishing true security vulnerabilities from general security-related issues or non-critical bugs.

Step ⓔ: CVSS score prediction. Additionally, the expert was asked to assign a CVSS score to each confirmed vulnerability. In this study, we used CVSS version 3.1 [14] to evaluate and quantify the severity of each confirmed vulnerability.

4 Results

Model selection. Table 1 summarizes the performance of all evaluated models on the Chromium dataset [36] in terms of accuracy, precision, recall, and F1-score. Each BERT-based model was trained 10 times on the same dataset, and the reported results reflect the averaged performance across these runs. The performance metrics for the general-purpose models were computed using the entire training dataset.

First, we evaluated the general-purpose LLMs, namely Gemma and Mistral. As shown in Table 1, both models achieved precision values of approximately 0.06 on the Chromium dataset, resulting in F1-scores of about 0.11-0.12 despite moderate recall. This indicates that they produced a very high number of false positives and are, therefore, unsuitable for practical application. In addition, even though few-shot examples were provided to enforce a consistent output format, the models occasionally generated responses that did not follow the template. Such deviations, often referred to as hallucinations, make it difficult to parse results automatically. From a practical standpoint, this means that, beyond handling numerous false positives, a security analyst would also need to manually inspect all malformed outputs. Finally, general-purpose models were also slower at the classification task.

The slower inference speed of these models, combined with their high rate of false positives and the need for extensive post-processing, raises concerns about using general-purpose models for our task. Although models like Mistral show promise, particularly because they do not require fine-tuning, their large-scale practical application requires careful consideration of the trade-offs between speed, accuracy, and the resources needed for post-processing. At the same time, larger and more advanced LLMs may offer substantially better performance and could even outperform fine-tuned models in accuracy; however, they would still be slower than smaller, specialized fine-tuned models, limiting their practicality in real-world settings.

As shown in Table 1, the fine-tuned BERT-based models generally outperformed the general-purpose models. Although BERT achieved a high recall of 0.97 on the full dataset, its low precision of 0.27 indicates that a large proportion of its positive predictions were false positives. RoBERTa offered a more balanced performance but was still limited by a low precision of 0.22. In contrast, DeBERTaV3 achieved both the highest recall (0.98) and the highest precision (0.30) among the discriminative transformer models, resulting in the best overall F1-score (0.45). Therefore, for the remainder of this study, we select DeBERTaV3 as our primary model.

Threshold adjustment. After selecting the model, we experimented with different threshold values to identify the most effective balance between precision and recall. The results (Table 2) show that increasing the threshold from the base value of 0.50 to 0.98 leads to a steady improvement in precision, rising from 0.48 to 0.91. However, this gain comes at the cost of reduced recall, which declines from 0.97 to 0.89. The F1-score, which reflects the balance between precision and recall, still improves as the threshold increases and reaches its peak at the threshold value of 0.98. Despite this, the drop in recall at higher thresholds indicates that the model begins to miss a non-negligible number of security-relevant issues – an unacceptable outcome in our case. Therefore, we set the threshold to 0.95 and use this value in the remainder of the study. This threshold value corresponds to an acceptable model's recall while still considerably reduces validation efforts.

Cross-project security issue predictions. Due to resource constraints, we selected four large projects – *ClickHouse*, *OpenCV*, *Bitcoin*, and *gRPC* – and applied our classifier to all issues there. Table 3 lists the projects and presents the results.

In total, 41,181 issues were analyzed (excluding empty entries). The proportion of predicted security issues relative to the total number of issues varies across projects, with ClickHouse having the lowest ratio at 0.79% and Bitcoin the highest at 2.40%. These differences are intriguing and may result from various factors, including the projects' specific use cases, architectures, or the characteristics of the communities maintaining them. Nonetheless, all four projects fall below 3%, indicating that security-relevant issues consistently

Table 3: Cross-Project Prediction Results

Project	Total Issues	Predicted Issues	Ratio (%)
ClickHouse	16,013	127	0.79
OpenCV	7,978	106	1.33
Bitcoin	7,284	175	2.40
gRPC	9,906	232	2.34
Total	**41,181**	**640**	**1.55**

constitute only a small fraction of total issue reports and reflecting the level of attention maintainers devote to security.

Across all four projects, the model predicted 640 as being security-related issues out of 41,181 in total, which gives the overall rate of security-related issues of 1.55%.

Vulnerability confirmation. To assess whether security issues predicted by the DeBERTaV3 model correspond to actual software vulnerabilities, we conducted expert validation with a CNA administrator. Due to the limited expert availability, the evaluation focused on a single project: gRPC. The expert reviewed 232 issues from this project flagged by the model as security-related and determined whether they would reasonably qualify for CVE identifiers.

The validation results are noteworthy. Out of the 232 security issues flagged by the model, 52 (22.4%) have been assessed as potential CVEs. This finding suggests that a significant number of vulnerabilities may be publicly disclosed without ever being assigned a CVE identifier. Based on the validated sample, the results suggest that roughly *0.5% of published issues may contain a previously untracked vulnerability*, corresponding to approximately one vulnerability per 190 issues. While this provides a useful estimate and provides an answer to our research question, it is, of course, conservative and specific to the gRPC project. Future work should evaluate whether this estimate is generalizable to other open-source projects.

Interestingly, at the time of study, NVD listed only 11 CVEs under the gRPC project. This discrepancy underscores the efficacy of the model in uncovering hidden CVEs candidates, with the model identifying nearly five times more CVE candidates than what is currently listed. It should be noted, however, that this analysis only covers issues flagged by the model, implying that the actual number of hidden security issues could be even higher.

We manually analyzed the 11 CVEs listed in the NVD to identify references to the gRPC GitHub repository. The analysis showed that three issues were referenced in CVE descriptions but were not predicted by the model (false negatives). Further examination revealed that these misses occurred because the initial issue posts were either very short and non-descriptive or primarily contained links to other security reports that were removed during preprocessing. This highlights the difficulty of capturing security-relevant information using only the initial text of issue reports.

Another notable finding is the limited correlation between issues identified as potential CVEs and those labeled by the maintainers as security-related (in the gRPC project, the label "area/security" is used to indicate such issues). Our analysis revealed that only 3 of the 52 issues flagged as potential CVE candidates by the model were tagged with this security label, highlighting a gap in the existing labeling process and suggesting that datasets relying solely on this label may have very low coverage.

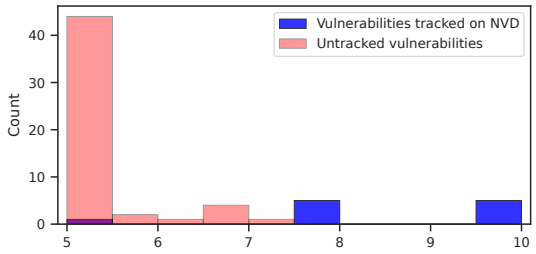


Figure 2: CVSS scores of tracked/untracked vulnerabilities.

Overall, the expert validation indicates that a substantial number of security-relevant vulnerabilities are publicly disclosed in the gRPC issue tracker without being formally tracked as CVEs. This reveals a structural visibility gap between public development platforms and centralized vulnerability reporting channels. Collectively, these findings underscore the need to further investigate how existing vulnerability coordination and disclosure mechanisms influence which publicly disclosed vulnerabilities ultimately become visible to the broader security community.

CVSS score prediction. For issues classified as vulnerabilities, the expert also provided an estimated severity score using the CVSS 3.1 base metrics, enabling a standardized severity comparison. The evaluation also included a justification for the decision and the scoring. Figure 2 shows the estimated CVSS score distribution of the tracked/untracked vulnerabilities for the gRPC project.

The most common CVSS score in the dataset is 5.1, appearing 44 times, indicating that many identified vulnerabilities are of moderate severity. While not immediately critical, such vulnerabilities still require attention to prevent potential exploitation. The dataset also includes a substantial number of higher-severity vulnerabilities, with scores such as 5.9, 6.3, 6.5, and 7.1, which could have more significant impacts if exploited. Overall, the average CVSS score across all identified issues is 5.3, largely influenced by the high frequency of scores of 5.1.

A notable disparity exists between the CVSS scores of tracked and untracked vulnerabilities, with those listed in the NVD generally receiving higher ratings, as shown in Figure 2. We hypothesize that this difference arises because more severe vulnerabilities are often disclosed through channels other than GitHub issues to reduce the risk of exploitation before patches become available.

5 Discussion

This study investigates the extent to which software vulnerabilities disclosed in public issue trackers remain untracked by formal vulnerability databases and the implications of this gap for cybersecurity practice and governance. Our results show that public issue trackers constitute an underexplored attack surface, where security-relevant issues are often disclosed outside Coordinated Vulnerability Disclosure (CVD) processes.

Public issue trackers as a persistent security risk. While OSS benefits from transparency and community oversight, this openness also exposes vulnerability disclosures to adversaries. Unpatched public reports can effectively serve as exploitation guides, leading to

breaches and operational disruption. These risks propagate through supply-chain dependencies, where a single vulnerable component can compromise multiple systems. The Log4Shell vulnerability illustrates how one open-source flaw can have global consequences across industries and public infrastructure [13]. Our findings suggest that similar conditions persist today, with vulnerabilities still being publicly disclosed in issue trackers without coordination, tracking, or timely remediation. The GitHub Octoverse 2025 report [19] further highlights the need for improved processes. It reports 1,701,552 new C++ projects created between August 2024 and August 2025. If the observed proportion of untracked vulnerabilities generalizes across these projects, this represents a substantial attack surface that must not be overlooked.

Behavioral roots of uncoordinated vulnerability disclosure. Beyond technical detection, the results highlight that untracked vulnerabilities are not merely a tooling failure but a behavioral and organizational one. Public disclosure of vulnerability information is often driven by developers rather than security researchers. While the latter are typically familiar with coordinated disclosure norms and established reporting channels [23], developers may disclose sensitive information publicly due to limited awareness, unclear reporting procedures, or frustration with perceived bureaucratic delays. This creates a systemic mismatch between software development practices and vulnerability coordination mechanisms.

Policy implications and disclosure governance. Our findings highlight the need to better align developer behavior with CVD practices. Rather than relying on regulatory enforcement or structural changes to the CVE system, influencing developer behavior may be more effective in preventing premature public disclosure of security-critical issues. The COM-B framework (Capability, Opportunity, Motivation-Behavior) provides a useful lens for understanding and shaping such behavior [27]. From this perspective, three intervention areas emerge. *Opportunity:* Projects should provide clear disclosure policies and accessible secure reporting channels, such as security tabs, encrypted email, or private issue trackers [15]. *Motivation:* Incentives, including bug bounty programs and public recognition, can encourage responsible disclosure practices [32]. *Capability:* Developers need sufficient security awareness to recognize and properly report vulnerabilities, which can be supported through accessible training and community education. Together, these measures can reduce the premature exposure of vulnerability information in public issue trackers while preserving the decentralized governance model of open-source ecosystems.

Broader implications for vulnerability management. Overall, the results posit that the CVE ecosystem does not capture the full vulnerability landscape, not only due to technical limitations but also because of upstream disclosure practices. While AI can help surface hidden vulnerabilities, it cannot compensate for weak disclosure coordination alone. Vulnerability databases, detection models, and disclosure governance should therefore be treated as a coupled socio-technical system. Without improvements in source-level reporting, untracked vulnerabilities will continue to accumulate in public repositories, exposing users, organizations, and critical infrastructure to avoidable risk. These findings highlight the need for stronger integration between open-source development practices, vulnerability coordination frameworks, and incentives that discourage premature public disclosure.

5.1 Limitations

This study has several limitations. A key constraint is its exclusive focus on C/C++ projects. While this focus likely improves detection of security issues common in C++, such as memory management vulnerabilities, it is unclear whether the model would perform similarly on projects written in other languages, such as Java, Go, or Python, where the characteristics and terminology of security-related bugs differ. However, C/C++ projects remain critical targets and are frequently used in similar studies, including a recent work by Anthropic [9].

Another limitation is the use of a single dataset for model fine-tuning, which may limit the classifier's ability to generalize effectively [10]. Future work could address this by incorporating issues from multiple projects, enabling the model to learn from more diverse sources and potentially improving cross-project applicability.

The validation of vulnerabilities was performed by a single expert, introducing potential bias, as assessments rely on individual judgment and experience [37]. While involving multiple experts would improve reliability, recruiting qualified specialists is challenging due to their scarcity and high demand. Moreover, because of limited expert time, evaluation was restricted to issues detected by the model within a single project. As a result, the findings may not fully capture variability across different projects or programming languages, and caution is warranted when generalizing these results to broader contexts.

6 Conclusion

This study set out to assess the extent to which software vulnerabilities disclosed in public issue trackers remain untracked by the CVE ecosystem and to evaluate whether machine learning-based security issue classification can help close this visibility gap. By applying a fine-tuned DeBERTaV3 classifier to large C++ open-source projects and validating predictions through a CNA Administrator review, we provide empirical evidence that a substantial number of vulnerabilities are publicly disclosed without ever being assigned a CVE identifier.

Across four major C++ projects, security-relevant issues consistently accounted for a small fraction of total issue volume (0.8%-2.4%). Yet, validation of the gRPC dataset by the CNA Administrator showed that 22.4% of the model-flagged security issues at a conservative threshold would be eligible for a CVE identifier. These discovered vulnerabilities were mainly of moderate severity, and at the time of writing, were absent in the National Vulnerability Database. This confirms that public issue trackers constitute a structurally important but systematically underutilized source of vulnerability intelligence. Methodologically, our results demonstrate that transformer models, including Large Language Models, can support large-scale discovery of hidden vulnerabilities when paired with expert validation.

Acknowledgments

This work was partially supported by the Dutch Research Council (NWO) as part of the THESEUS project (NWA.1215.18.006).

References

[1] [n. d.]. OpenCV. https://opencv.org/.
[2] [n. d.]. React Native. https://reactnative.dev/.
[3] [n. d.]. TensorFlow: An end-to-end open source platform for machine learning. https://www.tensorflow.org.
[4] 2011. Mining Software Repository Challenge. http://2011.msrconf.org/msr-challenge.html.
[5] Roi Abitboul. [n. d.]. What are CVE-Less Threats? https://raven.io/blog/what-are-cve-less-threats.
[6] Sultan S. Alqahtani. 2024. Security bug reports classification using fasttext. *International Journal of Information Security* 23, 2 (April 2024), 1347–1358. doi:10.1007/s10207-023-00793-w
[7] Jeanine Banks and Tris Warkentin. 2024. Gemma: Introducing New State-of-the-Art Open Models. https://blog.google/technology/developers/gemma-open-models/.
[8] Black Duck. 2025. Open Source Security and Risk Analysis Report. https://www.blackduck.com/resources/analyst-reports/open-source-security-risk-analysis.html
[9] Nicholas Carlini, Keane Lucas, Evyatar Ben Asher, Newton Cheng, Hasnain Lakhani, David Forsythe, and Kyla Guru. 2026. Evaluating and mitigating the growing risk of LLM-discovered 0-days. https://red.anthropic.com/2026/zero-days/.
[10] Saikat Chakraborty, Rahul Krishna, Yangruibo Ding, and Baishakhi Ray. 2022. Deep Learning Based Vulnerability Detection: Are We There Yet? *IEEE Transactions on Software Engineering* 48, 9 (2022), 3280–3296. doi:10.1109/TSE.2021.3087402
[11] NVIDIA Corporation. 2020. *NVIDIA T4 70W Low Profile PCIe GPU Accelerator*. https://www.nvidia.com/en-us/data-center/tesla/tesla-qualified-servers-catalog/ Product Brief, PB-09256-001_v05.
[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics*. 4171–4186. doi:10.18653/v1/N19-1423
[13] Douglas Everson, Long Cheng, and Zhenkai Zhang. 2022. Log4shell: Redefining the web attack surface. https://www.ndss-symposium.org/wp-content/uploads/madweb2022_23010_paper.pdf. In *Workshop Meas., Attacks, Defenses Web*. 1–8.
[14] FIRST. 2021. *Common Vulnerability Scoring System v3.1: Specification Document*. https://www.first.org/cvss/specification-document
[15] Brian Fitzgerald and Klaas-Jan Stol. 2017. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software* 123 (2017), 176–189. doi:10.1016/j.jss.2015.06.063
[16] Patrick Garrity. 2025. Expanding VulnCheck's KEV: Auditing ShadowServer, New CVE Assignments, and Source Expansion. https://www.vulncheck.com/blog/kev-expansion-2025.
[17] Michael Gegick, Pete Rotella, and Tao Xie. 2010. Identifying security bug reports via text mining: An industrial case study. (05 2010), 11–20.
[18] Rajdeep Ghosh, Shiladitya De, and Mainack Mondal. 2025. "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. In *USENIX Security Symposium*. Article 111, 20 pages.
[19] GitHub Staff. 2025. Octoverse: A new developer joins GitHub every second as AI leads TypeScript to #1. https://github.blog/news-insights/octoverse/octoverse-a-new-developer-joins-github-every-second-as-ai-leads-typescript-to-1/.
[20] Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2023. DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing. arXiv:2111.09543 [cs.CL]
[21] Yuning Jiang, Nay Oo, Qiaoran Meng, Hoon Wei Lim, and Biplab Sikdar. 2025. A Survey on Vulnerability Prioritization: Taxonomy, Metrics, and Research Challenges. arXiv:2502.11070 [cs.CR] https://arxiv.org/abs/2502.11070
[22] Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong. 2018. VulDeePecker: A Deep Learning-Based System for Vulnerability Detection. In *Network and Distributed System Security Symposium*. Internet Society. doi:10.14722/ndss.2018.23158
[23] J. Lin, B. Adams, and A. E. Hassan. 2023. On the coordination of vulnerability fixes: An empirical study of practices from 13 CVE numbering authorities. *Empirical Software Engineering* 28, 6 (2023), 151.
[24] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv:1907.11692 [cs.CL]
[25] Jyoti Prakash Meher, Sourav Biswas, and Rajib Mall. 2024. Deep learning-based software bug classification. *Information and Software Technology* 166 (2024), 107350. doi:10.1016/j.infsof.2023.107350
[26] Mistral AI. 2023. Announcing Mistral-7B. https://mistral.ai/news/announcing-mistral-7b/.
[27] Susan Mitchie, Lou Atkins, and Robert West. 2014. *The behaviour change wheel: a guide to designing interventions*. Silverback Publishing.
[28] Fayola Peters, Thein Than Tun, Yijun Yu, and Bashar Nuseibeh. 2017. Text filtering and ranking for security bug report prediction. *IEEE Transactions on Software Engineering* 45, 6 (2017), 615–631.
[29] Benjamin C Pierce. 2002. *Types and programming languages*. MIT press.
[30] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. 2014. A large scale study of programming languages and code quality in github. In *International Symposium on Foundations of Software Engineering*. 155–165. doi:10.1145/2635868.2635922
[31] G. Schryen. 2009. Security of open source and closed source software: An empirical comparison of published vulnerabilities. In *Americas Conference on Information Systems*. 387.
[32] Amutheezan Sivagnanam, Soodeh Atefi, Afiya Ayman, Jens Grossklags, and Aron Laszka. 2021. On the benefits of bug bounty programs: A study of chromium vulnerabilities. In *Workshop on the Economics of Information Security*.
[33] The MITRE Corporation. [n. d.]. Common Vulnerabilities and Exposures. https://www.cve.org. ([n. d.]).
[34] The MITRE Corporation. 2025. CVE Numbering Authority Operational Rules. https://www.cve.org/resourcessupport/allresources/cnarules
[35] TIOBE Software. 2026. TIOBE Index for February 2026. https://www.tiobe.com/.
[36] X. Wu, W. Zheng, X. Xia, and D. Lo. 2021. Data quality matters: A case study on data label correctness for security bug report prediction. *IEEE Transactions on Software Engineering* 48, 7 (2021), 2541–2556.
[37] Julia Wunder, Andreas Kurtz, Christian Eichenmüller, Freya Gassmann, and Zinaida Benenson. 2024. Shedding Light on CVSS Scoring Inconsistencies: A User-Centric Study on Evaluating Widespread Security Vulnerabilities. In *IEEE Symposium on Security and Privacy (SP)*. 1102–1121. doi:10.1109/SP54263.2024.00058
[38] Zhenbo Xu, Jian Zhang, and Zhongxing Xu. 2011. Memory Leak Detection Based on Memory State Transition Graph. In *Asia-Pacific Software Engineering Conference*. 33–40. doi:10.1109/APSEC.2011.22
[39] Wei Zheng, Zheng Chen, Xiaoxue Wu, Weiqiang Fu, Bowen Sun, and Jingyuan Cheng. 2021. A domain knowledge-guided lightweight approach for security bug reports prediction. In *International Conference on Dependable Systems and Their Applications*. 359–368.
[40] Noah Ziems and Shaoen Wu. 2021. Security vulnerability detection using deep learning natural language processing. In *IEEE Conference on Computer Communications Workshops*. 1–6.