SHORT-PAPER

Aligning Core Aspects: Improving Vulnerability Proof-of-Concepts via Cross-Source Insights

LINGXIAO WANG, Tianjin University, Tianjin, China

WENJING DANG, Tianjin University, Tianjin, China

MENGYAO ZHAO, Tianjin University, Tianjin, China

YUE WANG, Tianjin University, Tianjin, China

XIANZONG WU, Tianjin University, Tianjin, China

SEN CHEN, Nankai University, Tianjin, China

Aligning Core Aspects: Improving Vulnerability Proof-of-Concepts via Cross-Source Insights

Lingxiao Wang
Tianjin University
China
wlx_0613@tju.edu.cn

Wenjing Dang
Tianjin University
China
dangwenjing@tju.edu.cn

Mengyao Zhao
Tianjin University
China
mengyaozhao@tju.edu.cn

Yue Wang
Tianjin University
China
16622536761@163.com

Xianzong Wu
Tianjin University
China
wxz_buta@tju.edu.cn

Sen Chen*
Nankai University
China
senchen@nankai.edu.cn

Abstract

For vulnerabilities, Proof-of-Concept (PoC) plays an irreplaceable role in demonstrating the exploitability. PoC reports may include critical information such as specific usage, test platforms, and more, providing essential insights for researchers. However, in reality, due to various PoC templates across PoC platforms, PoC reports extensively suffer from information deficiency, leading the suboptimal quality and limited usefulness. Fortunately, we found that information deficiency of PoC reports could be mitigated by the completion from multiple sources given the same referred vulnerability.

In this paper, we conduct the first study on the deficiency of information in PoC reports across public platforms. We began by collecting 173,170 PoC reports from 4 different platforms and defined 8 key aspects that PoCs should contain. By integrating rule-based matching and a fine-tuned BERT-NER model for extraction of key aspects, we discovered that all PoC reports available on public platforms have at least one missing key aspect. Subsequently, we developed a multi-source information fusion method to complete the missing aspect information in PoC reports by leveraging CVE entries and related PoC reports from different sources. Finally, we successfully completed 69,583 PoC reports (40.18% of all reports).

CCS Concepts

• **Security and privacy**; • **Software and its engineering**;

Keywords

Proof-of-Concept, Software Security

---

*Sen Chen is the corresponding author.

---

1 Introduction

In the field of software security, vulnerability reports are the primary means by which researchers learn about security flaws. Proof-of-Concept (PoC) plays an indispensable role by offering practical exploitation methods and code examples. PoC reports usually involve a description, affected software, and additional information that helps gain a deeper understanding of the vulnerabilities at hand. PoCs assist developers in pinpointing the vulnerabilities, thereby facilitating the remediation process and evaluation of exploitability [2, 14, 16].

Furthermore, the content of PoC reports, including usage details and testing platforms, has been analyzed in a few studies [7, 9], which emphasize the significant impact of such information on the applicability of PoCs. Mu et al. [9] specifically highlighted the crucial role that PoC reports play in the process of vulnerability reproduction. However, despite their importance, the quality of PoC reports varies greatly, as noted in recent research [1]. The type and completeness of information differ significantly across platforms, resulting in frequent omissions of critical details. This inconsistency reduces the overall usability of public PoC reports. Furthermore, there is still a lack of comprehensive research and effective methodologies to systematically address the issue of missing information in vulnerability PoCs.

To bridge the gap, we conducted the first empirical study to understand the information deficiency of PoC reports across public vulnerability report platforms. Specifically, this paper investigates the prevalence of missing information in PoC reports from various sources and proposes a multi-source information completion approach to effectively rectify these gaps. First, we collate PoC reports from multiple public repositories and conduct a series of analytical studies. ① **Evaluating Information Deficiency of PoC Reports**. We identified 8 key aspects that PoC reports should encompass. For each aspect, we develop specific extraction approaches to accurately identify and extract information. We found on average only 56% of aspects were present in PoC reports, indicating a significant absence of crucial data. ② **Completion of PoC Aspects Based on Multiple Sources**. To remedy the missing information, we cross-reference PoCs with corresponding CVE entries, facilitating

a linkage between different sources of PoC reports. We categorized the PoCs into code-based and text-based, designed methods to complete PoC reports with related CVE entries and other PoC reports based on aspects. Our experiments demonstrate the effectiveness of these completion strategies.

2 Background

2.1 Diverse PoC Submission Templates for Multiple Platforms

PoC reports are published at diverse sources, including platforms like ExploitDB [10] and CXSecurity [3], where security researchers, from independents to professional firms, share PoCs for various products and vulnerabilities. While both platforms aim to facilitate security issue resolution, they differ in PoC handling, requiring distinct submission templates and mandatory information. Incomplete templates or submissions that do not fully adhere to the specified templates can result in missing information. This issue is particularly prevalent on platforms with optional template requirements, such as Packet Storm [11]. Thus, single-source PoC reports often suffer from severe information deficiencies, which not only diminish their usability but also hinder a comprehensive understanding of the relationship between the PoC and the associated vulnerability.

2.2 Motivating Example

To address the information gaps in PoC reports, we utilize complementary reports from different sources. For instance, a PoC from Packet Storm related to CVE-2003-0264 provides minimal details beyond execution code, lacking critical information such as affected software versions. In contrast, an associated PoC from ExploitDB for the same CVE [4] includes comprehensive details like the author, operating platform, and execution results, enriching the dataset.

3 Data Preparation

3.1 Data Collection

To gather the most diverse and reliable open-source PoC data possible, we first sourced links tagged with "Exploit" from the "References to Advisories, Solutions, and Tools" section on the NVD. Links associated with the vulnerabilities are tagged according to their relevance, with links labeled as "Exploit" indicating relevance to vulnerability exploitation. We crawled all links with the "Exploit" tag and traced their origins, eventually selecting the top ten data sources based on reference frequency for preliminary inclusion, and finally select four data sources with higher-quality and more structured PoC content: ExploitDB [10], Packet Storm [11], Seebug [12], and CXSecurity [3]. We employed techniques such as web scraping and the use of official data, tailoring data extraction methods specifically for each of the aforementioned 4 sources. This process involved preliminary data cleaning to remove erroneous entries that did not contain PoC information. Ultimately, over 200 man-hours were expended to collect a total of 173,170 PoC reports from these 4 data sources.

3.2 Categorizing PoC into Code and Text

Motivation for Categorizing. Given the diversity of PoC report formats, employing a uniform processing approach could lead to
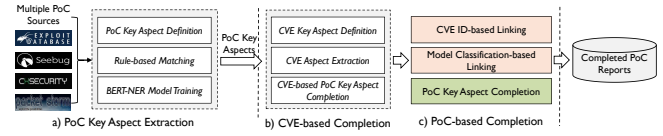

Figure 1: Overview of our study.

a substantial loss of format-specific information. We categorize PoCs into code-based, text-based, and other forms to preserve the uniqueness of each type. Code-based PoCs, such as the Python code PoC [13], involve parsing based on syntactic and functional attributes. Text-based PoCs consist of narrative descriptions, while other forms include non-textual media like images or videos, which require different identification methods [14].

Categorization Approach. Our analysis primarily focuses on code-based and text-based PoCs, as PoCs in non-textual forms are less prevalent. We distinguish these types by utilizing regular expressions to detect programming languages, such as C/C++, HTML, Java, Python, Perl, PHP, Python, Ruby, and Shell, commonly used in PoCs. If a PoC does not match these language patterns, it is classified as text-based [14].

4 Empirical Study

The overview of our study is shown in Figure 1.

4.1 Evaluating Information Deficiency of PoC Reports

4.1.1 Key Aspects of PoC. To standardize the information in PoC reports and ensure uniform processing, we identified eight key aspects crucial for understanding and utilizing them effectively. Drawing from research by Mu et al. [9], which underscores the importance of certain elements in reproducing vulnerabilities, we categorized these aspects into two groups: Exploit and Basic. **Exploit Category** represents the exploit value of PoC reports. These 4 aspects are indispensable for the usability of a PoC, including *Trigger Step*, *Verification Oracle*, *Test Platform*, *Software Version*. **Basic Category** represents the reference value, including *Title*, *Author*, *Publish Time* and *Reference*. These eight key aspects, termed the **Key Aspects** of a PoC report, furnish layered information critical for the accurate reproduction of vulnerabilities and essential references for further exploration.

4.1.2 Extraction of Key Aspects. Based on the 8 key aspects of PoC reports, we have developed extraction methods incorporating rule-based matching and NLP model-based approaches.

   (1) Rule-based Matching: We have established specific rules for *Trigger Step*, *Verification Oracle*, and *Reference* based on observations. For the *Trigger Step*, we have defined keywords such as "steps", "reproduce", and "complie with", along with regular expressions to match potential indicators and lists of numerical and alphabetical steps that may appear in a PoC report. For the *Verification Oracle*, we have set keywords like "expected output" and "PoC output" to identify possible results from program execution. *Reference* are generally in the form of links, so we utilized a regular expression that matches the structure of web URLs to extract.

Table 1: Count of completed PoC reports and aspects by CVE entries.

Sources	CVE	PoC w/ CVE	Test Platform		Software Version	
			PoCs	Aspects	PoCs	Aspects
Packet St.	7,465	8,478	1,203	9,605	7,381	19,455
ExploitDB	24,309	26,517	1,784	8,574	18,790	39,728
Seebug	11,861	14,400	666	2,215	9,292	20,158
CXSecurity	18,334	15,812	1,499	6,387	12,285	26,119
Overall	**34,024**	**65,207**	**5,152**	**26,781**	**47,748**	**105,460**

(1) *Poc w/ CVE* refers to PoC reports with CVE IDs.

(2) BERT-NER Model: Referring to the work of Guo et al [6], for extracting less structured and more nuanced information such as *Test Platform*, *Software Version*, *Title*, *Author*, and *Publish Time*, we utilized a BERT-NER model [5]. This model was trained on a subset of 2,500 manually annotated PoCs, using the BIO tagging scheme to identify these aspects effectively. The training involved 2,000 PoCs, with the remaining 500 serving as a test set. Fine-tuning parameters included a learning rate of 2e-5, ten epochs, weight decay of 0.01, label smoothing of 0.1, and Cross-Entropy Loss as the loss function. The training utilized an NVIDIA RTX 3090 GPU, ensuring efficient processing and robust model performance for accurate aspect extraction.

The overall precision and recall for extracting key aspects of PoC reports are 0.88 and 0.81 respectively. For PoC aspects, the precision and recall for *Publish Time*, *Reference*, *Trigger Step*, and *Verification Oracle* all exceeded 0.9, which ensures great extraction results for each aspect.

4.2 Completion of PoC Aspects Based on Multiple Sources

The completion rules comprise two categories: ① Completion using CVE entries and ② Completion using PoC reports.

4.2.1 PoC Completion Approach by Aspects of CVE Entries. To enhance the connectivity and completeness of PoC reports, we utilized CVE IDs as a unifying reference. We developed source-specific methods to extract CVE IDs from PoCs, recognizing that their locations vary across platforms—for example, ExploitDB houses them in a fixed field. This targeted extraction approach helps avoid the false positives associated with naive pattern matching. Our collection identified 34,024 unique CVE IDs linked to 65,207 PoC reports. Research on CVE entries [15] shows they contain comprehensive details like affected software and versions, platforms, and vulnerability types. By mapping these CVE details to the corresponding PoC reports, we can enrich PoCs lacking specific information. This is critical as PoC reports often serve to validate and reproduce vulnerabilities. We specifically correlated *Software Version* and *Test Platform* from PoCs with the **Version** and **Platform** from CVE entries, using **Product** merely for verifying associations between CVE entries and PoC reports.

We employed CVE Aspects to complete PoC Aspects. For the **Version**, our completion strategy adds version information from CVE entries related to the same CVE ID, based on the verification of the **Product** field. Since one CVE may affect multiple software,

the **Version** entries are mapped in a 1-to-n fashion. A PoC, however, is typically written for one specific version. Therefore, if the original PoC lacks version information, we directly complete its *Software Version* aspect. If it contains a partial version list, we append the missing versions for reference. The **Version** field in CVE Aspects corresponds to the *Software Version* in PoC Aspects, while the **Platform** field maps to the *Test Platform* in PoC reports. Completeness is evaluated by comparing these two aspects in each PoC against the corresponding CVE data. This includes both missing and incomplete cases in the original PoC.

The results of our completion approach are summarized in Table 1. In total, **47,748** PoC reports were completed, covering **132,241** aspects. Notably, completions for *Software Version* far exceed those for *Test Platform*. Specifically, **6,482** PoCs and **34,783** aspects were completed for *Test Platform*, while **58,222** PoCs and **133,493** aspects were completed for *Software Version*. This gap arises because not all CVE entries include **Platform** data, whereas all include affected **Product** and **Version** fields. Furthermore, a single CVE often involves multiple software and versions, making it rare for a single PoC to include all relevant version information—resulting in more opportunities for completion. In conclusion, the experiment confirms the practicality and effectiveness of leveraging CVE entries to enrich PoC reports.

4.2.2 PoC Completion Approach by other Related PoCs. To establish connections among PoCs from different data sources, we employed two distinct methods.

Based on CVE ID. For PoCs linked by a common CVE ID, we leverage the CVE as a marker to associate PoCs from varied sources, as these often pertain to the same vulnerability and share details like triggering mechanisms and exploitation logic. To enhance the completeness of PoC reports, we identify gaps in information that could be mutually filled based on their similarities. Nevertheless, PoC reports vary significantly due to different programming languages, impacting the applicability of shared information across reports. We address this by clustering PoC reports based on both CVE ID and the programming languages used, acknowledging that even similar vulnerabilities might be exploited differently across various reports. To ensure precision in information completion, we focus on PoCs with high similarity.

- **Code-Based**. We utilize a token-based method to measure code similarity across languages, including C/C++, Ruby, Python, PHP, JavaScript, Java, Perl, HTML, and Shell. This involves tokenizing the code and calculating token frequencies to establish a frequency dictionary for each PoC, with similarity assessed via Cosine Similarity [17].
- **Text-Based**. We employ the Word2Vec [8] to generate text vectors, comparing them using Cosine Similarity to gauge the similarity in described exploit processes. This methodological approach enables a structured and quantitative comparison of PoCs, facilitating the targeted completion of information within PoCs.

Based on Model classification. Linking PoCs without CVE IDs poses a challenge; hence, we developed a methodology to associate PoCs affecting the same vulnerability based on their software, version, and similarity in content. Initial analysis revealed that PoCs for the same vulnerability often impact the same software, albeit potentially different versions, and share similar titles and content.
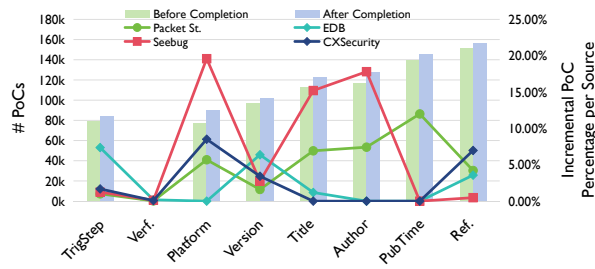
Figure 2: Overall completion results.

To establish whether two PoCs target the same software, we compared the extracted software names for exact matches. Subsequently, we used the similarity of titles and PoC content as indicators of identical vulnerabilities. For this purpose, we trained a BERT classification model [5] to differentiate whether two PoCs are associated with the same vulnerability. The model inputs include the titles and content of the PoCs. We constructed the training dataset using pairs of PoCs: those with the same CVE ID served as positive samples, and those with different CVE IDs as negative samples. Given that PoCs with CVE IDs comprise only 13.8% of our dataset, we selected 600 positive and 5,400 negative samples, maintaining an 8:1:1 training split. The model achieved an accuracy of 97.3% on the test set, validating the effectiveness of our approach in linking PoCs based on content similarity and software identity.

4.2.3 Results of Completion by Related PoCs. To ensure that paired PoCs accurately reflect the same exploit process, we conducted threshold-setting experiments and established distinct thresholds for PoCs linked by CVE ID. For code-based PoCs, the threshold was set at 0.5, while for text-based PoCs, we set it at 0.95. Utilizing these specific thresholds and completion rules, we carried out an information completion exercise across multiple PoC sources. The outcomes of this exercise are illustrated in Figure 2. The displayed bar graph details the number of PoCs containing each aspect both before and after the completion process, and the line graph highlights the incremental completions sourced from various databases.

We completed a total of 27,901 PoC reports across four datasets, focusing on those originally lacking specific aspect information. Furthermore, we completed 4,613 to 12,640 PoC reports by completing seven different aspects that were originally missing from these PoCs. Due to the originally small number of *Verification Oracles*, the number of PoCs that could be completed was limited to just 184. For the significantly lacking *Trigger Step*, we completed over 5,002 PoCs. Additionally, for the crucial aspects of *Test Platform* and *Software Version*, which are vital for reproducing vulnerabilities, we completed more than 12,640 and 4,613 PoC reports, respectively.

From the completion results across different sources, it is clear that the four datasets showed varying degrees of improvement across the eight aspects. Notably, Seebug exhibited significant enhancement, with completions for *Test Platform* increasing by over 20%, and both *Title* and *Author* exceeding 15%. These gains are largely due to substantial missing information in the original PoCs. In summary, our approach of leveraging related PoCs for aspect completion has proven highly effective, significantly enriching the quality and completeness of PoC information across multiple sources.

5 Conclusion

In this study, we collected PoC reports from 4 different sources and defined 8 key aspects of PoCs, revealing that on average only 56% of these aspects are present across sources. We used CVE entries linked via CVE IDs to supplement missing aspects in PoCs, successfully completing 58,222 PoCs with 133,493 additional aspects. We utilized PoCs from various sources for information completion, employing methods based on CVE IDs and model recognition to establish connections between different PoCs. To ensure the reliability of our approach, we further classified PoCs into text and code types, assigning specific similarity calculation methods to each type. Ultimately, this strategy enabled the successful completion of 27,901 PoC reports using related PoCs from different sources. In total, we completed 69,583 PoC reports, demonstrating the feasibility of this method and significantly enhancing the reliability of PoC data.

References

[1] 2024. Qualys Survey of Top 10 Exploited Vulnerabilities in 2023 | Qualys Security Blog. https://blog.qualys.com/qualys-insights/2023/09/26/qualys-survey-of-top-10-exploited-vulnerabilities-in-2023.
[2] Mehran Bozorgi, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. 2010. Beyond heuristics: learning to classify vulnerabilities and predict exploits. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. 105–114.
[3] Cxsecurity. 2024. CXSECURITY.COM Free Security List. https://cxsecurity.com/.
[4] Exploit Database. 2024. Seattle Lab Mail (SLmail) 5.5 - POP3 'PASS' Remote Buffer Overflow (1). https://www.exploit-db.com/exploits/638.
[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
[6] Hao Guo, Sen Chen, Zhenchang Xing, Xiaohong Li, Yude Bai, and Jiamou Sun. 2022. Detecting and augmenting missing key aspects in vulnerability descriptions. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 3 (2022), 1–27.
[7] Seongkyeong Kwon, Seunghoon Woo, Gangmo Seong, and Heejo Lee. 2021. OC-TOPOCS: automatic verification of propagated vulnerable code using reformed proofs of concept. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 174–185.
[8] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).
[9] Dongliang Mu, Alejandro Cuevas, Limin Yang, Hang Hu, Xinyu Xing, Bing Mao, and Gang Wang. 2018. Understanding the reproducibility of crowd-reported security vulnerabilities. In *27th USENIX Security Symposium (USENIX Security 18)*. 919–936.
[10] OffSec. 2024. Exploit Database. https://www.exploit-db.com/.
[11] Packet. 2024. Packet Storm. https://packetstormsecurity.com/.
[12] Seebug. 2024. Seebug. https://www.seebug.org/.
[13] Packet Storm. 2024. SLMail 5.1.0.4420 Remote Code Execution. https://packetstormsecurity.com/files/161526.
[14] Octavian Suciu, Connor Nelson, Zhuoer Lyu, Tiffany Bao, and Tudor Dumitraș. 2022. Expected exploitability: Predicting the development of functional vulnerability exploits. In *31st USENIX Security Symposium (USENIX Security 22)*. 377–394.
[15] Jiamou Sun, Zhenchang Xing, Xin Xia, Qinghua Lu, Xiwei Xu, and Liming Zhu. 2023. Aspect-level information discrepancies across heterogeneous vulnerability reports: Severity, types and detection methods. *ACM Transactions on Software Engineering and Methodology* 33, 2 (2023), 1–38.
[16] Yan Wang, Chao Zhang, Xiaobo Xiang, Zixuan Zhao, Wenjie Li, Xiaorui Gong, Bingchang Liu, Kaixiang Chen, and Wei Zou. 2018. Revery: From proof-of-concept to exploitable. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1914–1927.
[17] Jun Ye. 2011. Cosine similarity measures for intuitionistic fuzzy sets and their applications. *Mathematical and computer modelling* 53, 1-2 (2011), 91–97.