Drop the Golden Apples: Identifying Third-Party Reuse by DB-Less Software Composition Analysis

Lyuye Zhang
Nanyang Technological University
Singapore, Singapore
zh0004ye@e.ntu.edu.sg

Chengwei Liu
Nanyang Technological University
Singapore, Singapore
chengwei.liu@ntu.edu.sg

Jiahui Wu
Nanyang Technological University
Singapore, Singapore
jiahui004@e.ntu.edu.sg

Shiyang Zhang
Tianjin University
Tianjin, China
zsy_1@tju.edu.cn

Chengyue Liu
Nanyang Technological University
Singapore, Singapore
chengyue001@e.ntu.edu.sg

Zhengzi Xu
Imperial Global College
Singapore, Singapore
z.xu@imperial.ac.uk

Sen Chen
Nankai University
Tianjin, China
senchen@nankai.edu.cn

Yang Liu
Nanyang Technological University
Singapore, Singapore
yangliu@ntu.edu.sg

Abstract

The prevalent use of third-party libraries (TPLs) in modern software development introduces significant security and compliance risks, necessitating the implementation of Software Composition Analysis (SCA) to manage these threats. However, the accuracy of SCA tools heavily relies on the quality of the integrated feature database to cross-reference with user projects. While under the circumstance of the exponentially growing of open-source ecosystems and the integration of large models into software development, it becomes even more challenging to maintain a comprehensive feature database for potential TPLs. To this end, after referring to the evolution of LLM applications in terms of external data interactions, we propose the first framework of DB-Less SCA, to get rid of the traditional heavy database and embrace the flexibility of LLMs to mimic the manual analysis of security analysts to retrieve identical evidence and confirm the identity of TPLs by supportive information from the open Internet. Our experiments on two typical scenarios, native library identification for Android and copy-based TPL reuse for C/C++, especially on artifacts that are not that underappreciated, have demonstrated the favorable future for implementing database-less strategies in SCA.

CCS Concepts

• **Security and privacy → Software security engineering**.

ACM Reference Format:
Lyuye Zhang, Chengwei Liu, Jiahui Wu, Shiyang Zhang, Chengyue Liu, Zhengzi Xu, Sen Chen, and Yang Liu. 2025. Drop the Golden Apples: Identifying Third-Party Reuse by DB-Less Software Composition Analysis. In *33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25), June 23–28, 2025, Trondheim, Norway*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3696630.3728525

*Chengwei Liu is the corresponding author.

FSE Companion '25, Trondheim, Norway

ACM ISBN 979-8-4007-1276-0/2025/06
https://doi.org/10.1145/3696630.3728525

1 Introduction

In modern software development, the widespread use of third-party libraries (TPLs) and code reuse increases the risk of security vulnerabilities and compliance issues. Software Composition Analysis (SCA) mitigates these risks by detecting TPLs in projects and cross-referencing them with databases of known threats and licensing conflicts. Popular tools such as Snyk [6], Synopsys Black Duck [1], and WhiteSource [5] are widely adopted for securing the software supply chain.

Many researchers [11, 15, 22, 23, 25, 29] have explored improving and adopting SCA tools across diverse scenarios. However, most validate their tools using limited prototype databases, often based on only tens of thousands of repositories with over 100 stars [21–23], and report that incomplete component databases reduce tool accuracy. In practice, third-party reuse spans far beyond such datasets, challenging the validation of SCA tools. This problem is particularly critical for C/C++ libraries due to the lack of a centralized repository, unlike Python's PyPI [17], and the abundance of scattered libraries online [19]. As a result, accurately mapping detected code to known libraries remains difficult, complicating open-source software maintenance.

Constructing a comprehensive SCA database is non-trivial, especially with the rapid growth of open-source ecosystems and the rising integration of large models and frameworks. Maintaining an extensive and reliable feature database has become increasingly burdensome for SCA tools. Commercial SCA tools [1–3, 6, 7] depend on continuously updated databases, but these typically focus on centralized repositories like PyPI and often miss self-hosted or obscure libraries. Consequently, their coverage is limited, reducing effectiveness in detecting dependencies and analyzing vulnerabilities comprehensively. The widespread adoption of LLMs [12] has inspired new perspectives. LLM interactions with external data have evolved from heavy approaches like fine-tuning on specialized

datasets to lighter methods such as retrieval-augmented generation (RAG), which leverages smaller, focused datasets and the generalizability of LLMs. More recently, flexible solutions like LLM agents integrate search engines into task workflows by utilizing LLMs' reasoning capabilities.

To this end, we introduce *DB-Less SCA*, which departs from traditional database-dependent approaches. Instead of cross-referencing with a comprehensive database, our method emulates how security analysts manually identify TPLs, substituting this process with LLM-powered agents that follow a well-orchestrated chain of tasks. To move toward a database-less SCA solution, several challenges remain: **C1) Inadequate Identity Evidence.** Unlike traditional SCA tools that rely on pre-defined database features, manual TPL identification depends on expert intuition and diverse clues, making it difficult to systematically gather such identity evidence. **C2) Comprehensive Information Searching.** Analysts search for documentation or claims supporting the identity evidence to trace suspected TPLs. Automating this requires capturing analyst insights and orchestrating precise information retrieval pipelines. **C3) Insufficient Validation.** Validating the originality of TPLs based on external claims is complex. Integrating analyst knowledge into automated reasoning is essential for reliable assessment.

We propose the first live Database-less Software Composition Analysis (LiveSCA), which leverages real-time Internet data and combines LLMs with human expertise to re-orchestrate the SCA process for TPLs. To address C1, we introduce a general evidence-collection step that captures the *textual semantics* of target libraries, beyond traditional low-level features like code structure, enabling LLM agents to better understand library functionality. For C2, we adopt a multi-agent framework that iteratively scrapes, interprets, and compares search results. This mimics human analysis by identifying the most relevant webpages based on prior evidence, executed efficiently by LLM agents. To tackle C3, a dedicated validation agent independently confirms findings and provides feedback to the summarization agent. This loop refines search objectives in real time, improving the precision and relevance of results. LiveSCA achieved success rates of 59.20% and 57.50% on two typical SCA tasks. While not matching traditional tools with rich databases, these results highlight the promise of DB-less SCA, especially for overlooked TPLs, showing that the concept is still feasible.

2 Related Work

Traditional SCA tools: Both academic and industry tools considered state-of-the-art conduct Software Composition Analysis (SCA) through either Software Bill of Materials (BOM) [16] detection or code clone mapping [14, 24, 26–28, 30]. BOM detection tools, like OWASP [2] and Sonatype [3], extract TPLs listed in SBOM files, which detail dependency names, vendors, and versions—effective mainly for languages with official package managers. However, C/C++ lacks a universal package manager, limiting BOM-based detection. Thus, leading tools such as BlackDuck [1], Snyk CLI [6], ATVHunter [25], LibPecker [29], LibScout [11], and LibD [15] use code clone mapping by comparing control flow and other features against a database. CENTRIS [22] and OSSFP [23] improve accuracy using features from popular GitHub repositories. Yet, none rely on a complete TPL list, making them prone to false negatives.

Compiling a complete list of C/C++ TPLs is highly challenging. There is no official package manager for formal library registration, and many TPLs are scattered across personal websites or niche platforms. This fragmentation makes building a comprehensive TPL list impractical for improving current C/C++ SCA tools. Since all SCA tools rely on a feature database, its completeness directly affects accuracy. Our *DB-less* approach breaks from traditional methods that depend on large, often burdensome databases. Despite its advantages, this strategy remains largely unexplored in both industry and academia.

3 The Live DB-Less SCA Framework

LiveSCA enables database-less SCA by removing dependence on pre-established databases. As shown in Figure 1, it employs a multi-agent framework to autonomously identify and trace the origins of open-source libraries. Each agent, denoted as A_n, operates independently in its own LLM session unless otherwise noted. LiveSCA uses Internet-based retrieval to determine library identities, beginning with evidence extraction from target projects. It then summarizes core descriptions of suspected libraries and iteratively crawls and analyzes webpages to trace their origins. Finally, LiveSCA ranks the results and validates the identified sources.

Given the diverse scenarios involved in SCA for TPL identification, we selected two typical scenarios to demonstrate LiveSCA: identifying native libraries in Android application packages (APKs) [9] and detecting cloned TPLs in C/C++ projects [22]. The identification of native libraries in Android apps is particularly crucial because these libraries, often written in C/C++, are commonly utilized as third-party components without uniform package management. It often results in insufficient evidence regarding how these libraries are compiled and incorporated, leading to challenges, such as vulnerabilities inherent in these libraries, posing security risks to the Android application. The second scenario is the identification of cloned source code in C/C++ libraries. Due to the absence of a uniform package manager for C/C++, libraries are often directly cloned into projects unlike other programming languages like Java. Therefore, it is crucial to identify the cloned code in C/C++ projects and their sources for subsequent analysis concerning security and licensing. Contemporary tools such as Centris [22] and OSSFP [23] are designed to extract precise features for library identification. However, these tools depend on an existing database that incorporates features from previously identified libraries, which can be overcome by LiveSCA.

3.1 Evidence Extraction

The initial step in the SCA process involves extracting features from the target projects. Notably, SCA tools may accept various formats, such as source code, bytecode, or binary files. Thus, this extraction step is tailored to specific software formats. Regardless of the formats, the fundamental objective remains the same: to gather evidence that aids in the identification of libraries and the retrieval of their origins. This phase also involves filtering out extraneous noise, such as excessive strings extracted from ELF files. Given the dependency on format-specific procedures, detailed implementations are discussed in the Implementation Section 3.4. The general evidence includes dependency configurations, file locations, API

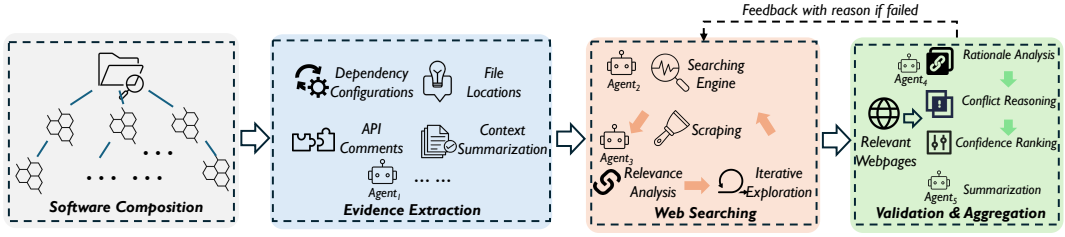

Figure 1: Overview of LiveSCA

comments or documentation, and the relevant code context, but the availability depends on specific tasks.

3.2 Web Searching

The subsequent step involves searching for the origins of the target libraries, such as their homepages and repositories. This process is initiated with inputs derived from the previously extracted evidence. LiveSCA employs a multi-agent strategy to heuristically analyze evidence and dynamically adjust the search process. Specifically, three agents are deployed for optimized searching. Agent A_1 processes the evidence to identify relevant keywords that encapsulate the core functionalities of the libraries. Based on these keywords, A_1 retrieves and saves the first two pages of results from Google, including titles, snippets, and URLs. For example, the keywords for Glad were *glad, OpenGL, loader, library, and CMake.*

Agent A_2 then collects the search results and iteratively scrapes the web pages by Scrapy [8] into texts only. Considering that webpage contents can be extensive, A_2 is designed to synthesize and summarize each page given the page content from the scraping, producing a concise summary that highlights the primary uses of the webpage. Consequently, each webpage is tagged with a brief summary and, as a fallback, a snippet from Google if the webpage crawling does not succeed. An example of Glad[1] is *"GLAD is a multi-language Vulkan/GL/GLES/EGL/GLX/WGL loader-generator based on official OpenGL specifications. It is commonly used to load OpenGL functions and extensions in graphics programming."*

Finally, Agent A_3 assesses the alignment between the library's evidence and webpage content to identify the original source. To filter out non-authoritative mentions from forums and forks, A_3 ranks pages by relevance and authenticity. In the Glad case, it correctly ranked the official homepage[1] first.

3.3 Validation and Aggregation

The final step in LiveSCA involves aggregation and dynamic validation, coordinated by two specialized agents. Agent A_4 performs independent validation to avoid bias, comparing top-ranked webpages with the initially extracted features. If misalignments are found, A_4 provides reasoning that is fed back to refine evidence keywords. This feedback loop runs up to three times to improve search results. Once validated or if the loop limit is reached, Agent A_5 aggregates the results, summarizes the target library, and outputs metadata suitable for inclusion in an SBOM.

[1]https://github.com/Dav1dde/glad

3.4 Implementation

LiveSCA is implemented using the widely adopted multi-agent framework, Dify [13], which orchestrates workflows based on LLMs with flexible, customized tools. The entire reproducible package of Dify DSL is packed and stored online [1] with detailed prompts. Within this framework, agents are implemented using GPT-4o [4], the state-of-the-art (SOTA) model renowned for its efficiency in handling a variety of text-based tasks.

For specific scenarios, the evidence extraction of LiveSCA varies significantly, necessitating customized tools tailored to software formats. In the context of native library identification in Android applications, LiveSCA initiates the process by extracting the APK [10] files and examining the ELF [20] files in SO [18] format contained within. Recognizing that the file name of an SO file often hints at the native library name, LiveSCA initially saves this file name. Following this, LiveSCA employs the `strings` command to extract readable strings from the SO file. Given the variability in how vendor or version information is represented across different SO files, LiveSCA selectively filters these strings based on their length (shorter than 10 characters are discarded), as they typically lack meaningful information.

For TPL recognition in C/C++ source projects, LiveSCA leverages on the file structure to pinpoint potential TPL locations. Utilizing the `tree` command, LiveSCA extracts the file structure of the target projects and inputs this data into an LLM agent. This agent then identifies potential TPLs and provides a list of the top five files deemed worth inspecting. Subsequently, LiveSCA reads the content of these specified files for each TPL. The contents of these files are extracted and used as preliminary evidence for further processing.

4 Experiments

This experiment is designed to address two pivotal research questions with two typical SCA scenarios:

- RQ1: How effectively does LiveSCA identify native libraries within Android applications?
- RQ2: How effectively does LiveSCA recognize open-source libraries in C/C++ projects?

4.1 Dataset Collection

For RQ1, our dataset collection involved Android applications from Google Play. Initially, we gathered the top 100 applications and extracted their corresponding SO files, resulting in a total of 11,205 files with duplicates. These files were associated with 108 different libraries, where various versions corresponded to distinct SO files.

[1]https://drive.google.com/file/d/1BsbohzZjfCkw2VQIhWCsuFmh6EAie4eM/

Table 1: Effectiveness of LiveSCA for Typical SCA Scenarios

Scenario	#Total	#Collected URL	#Correct	#Hints Found
SO in APK	108	64	64	102
Clone C/C++	200	119	105	121

To validate the effectiveness of LiveSCA, we randomly selected one SO file from each library, yielding a sample of 108 SO files.

For RQ2, we began by gathering a dataset of C/C++ projects from GitHub by filtering for projects that not only had over 100 stars but were primarily developed in C/C++, yielding 23,568 projects. For manual confirmation, we randomly inspected projects and manually confirmed the usage of TPLs of them until the projects reached 100. Then, we randomly selected 200 confirmed TPL as the dataset.

4.2 RQ1: Native library Identification

In the RQ1, we aimed to identify the sources, including homepages and repositories, of 108 native libraries based solely on given SO files. The results show that out of the 108 libraries, 64 (59.25%) were correctly located. A manual analysis of the URLs of the collected web pages validated the accuracy by the first three authors.

Further analysis was conducted on the unsuccessful cases. LiveSCA encountered network failures in 6 instances, leading to aborted processes. Excluding these, LiveSCA returned results for 102 cases, with 64 successful identifications included. For the 38 cases where sources could not be located, LiveSCA had correctly collected vendor information, but no accessible URLs were available. Specifically, 26 of these cases involved source URLs that led to `404` errors, indicating discontinued maintenance. Additionally, manual searches for the remaining 12 cases did not yield any results, underscoring the limitations of relying solely on Google search capabilities. Even in the 38 cases where sources could not be located, LiveSCA successfully returned vendor information deduced from the extracted strings. This information can still be leveraged to manually locate additional related webpages or undertake further searches. Thus, even the unsuccessful cases contribute valuable data for refining LiveSCA's capabilities and for potentially guiding manual follow-up searches or other fine-grained tools to uncover sources that are not readily accessible via LiveSCA.

Overall, this task took an average time of 27.99s with 10,196.85 tokens with an estimated cost of less than 3 USD.

4.3 RQ2: Cloned TPL in C/C++ Projects

In the RQ2 subsection, we randomly sampled 200 distinct cases to verify LiveSCA's performance. As indicated in Section 4.2, out of these cases, LiveSCA successfully identified the sources for 115 cases (57.50%). However, it initially found sources for 119 TPLs, but 4 of these were inaccurately identified upon manual verification. These inaccuracies primarily stemmed from the broad or ambiguous meanings associated with certain library names, which could refer to multiple different entities. Common issues included overly short names or names that overlapped with common terms. For example, the incorrect TPL homepage for *semver.com* was identified because the library name was `semver`, which, in the specific context, referred to a C/C++ library implementing Semantic Versioning. In contrast, *semver.com* hosts general documentation about Semantic Versioning rules, not a specific library.

This error highlights a challenge in using LLM agents: distinguishing between closely related concepts that have nuanced differences can be difficult. *semver.com* was flagged by the LLM due to its strong relevance to Semantic Versioning, albeit not as a direct source for the C/C++ library. Upon further manual investigation, the correct URL for the C/C++ library was identified as *https://github.com/h2non/semver.c*. This case exemplifies the need for enhanced understanding in LLMs to differentiate between contextually similar but distinct entities.

We analyzed failure cases where the LLM agent returned no URL and identified three main causes: (1) insufficiently distinctive features in the extracted source code, often due to incorrect enumeration, which led to irrelevant search queries; (2) inaccessible websites among the retrieved results, rendering URLs unusable; and (3) ambiguous keywords or generic library names that yielded numerous unrelated pages, obscuring the true source.

The average time LiveSCA took to complete the dataset is 29.36s with 15,166.62 tokens with an estimated cost of less than 8 USD.

5 Discussion

5.1 Limitations

As a preliminary prototype, our framework, LiveSCA, exhibits several limitations that impact its effectiveness. The primary limitation lies in the LLM's capacity to handle large text corpora. This constraint may lead to the omission of critical details, adversely affecting overall performance. To mitigate this, we have designed a standalone agent specifically to comprehend and summarize the essence or semantics of the corpus for further processing.

The second limitation concerns LiveSCA's reliance solely on Internet searches, restricting the scope to publicly available information. Private homepages, repositories, or data not currently available on the Internet remain inaccessible. Extending the search scope to include non-public data sources and historical webpage snapshots could enhance the tool's performance.

A third limitation is that our DB-less SCA framework primarily focuses on coarse-grained TPL source pinpointing. Fine-grained identification, which necessitates detailed detection of specific versions, is not currently supported due to the reliance on coarse evidence. For more precise identification, a more detailed extraction of library evidence is required to support subtle version differences.

Lastly, the framework lacks a comprehensive end-to-end validation process. Ideally, the DB-less SCA framework should validate collected sources at the semantic level, allowing the returned sources to be fed back into the evidence extraction step. This would enable the extraction of adjusted evidence from the returned sources for cross-verification with initial evidence.

5.2 Threats to validity

One of the primary threats to the validity of our work concerns the ground truth labeling process. Due to the necessity of manually reviewing collected webpages to understand the potential usage relationships between the user's project and the identified libraries, the risk of errors in labeling is significant. To mitigate this risk, we have implemented a majority voting scheme conducted by the first

three authors. This approach aims to minimize subjective biases and errors by ensuring that at least two out of the three authors agree on the relevance and accuracy of each labeled instance.

Another potential threat lies in the dataset construction, as the current dataset includes only a limited number of samples, which may affect its representativeness. To mitigate this issue, we selected widely used Android applications along with distinct native libraries, as well as popular C/C++ repositories, to ensure diversity and relevance. In future work, we plan to incorporate a more extensive dataset to support a more comprehensive evaluation.

The settings and configurations of employed techniques may pose a threat to validity. For example, the output of Scrapy for web scraping depends on its configuration, which affects the extracted text. Similarly, parameters like the depth of the `tree` command and file extension filters influence the scope and relevance of retrieved evidence. The temperature setting of GPT-4o can also affect generation consistency. To mitigate these risks, we adopted widely recommended configurations and set the temperature to zero.

6 Conclusion

In conclusion, the development and evaluation of LiveSCA have demonstrated its capability to effectively identify native libraries in Android apps and recognize cloned C/C++ TPLs, achieving success rates of 59.20% and 57.50% respectively. These results underscore the viability of a multi-agent, DB-less SCA framework, showcasing its potential to operate efficiently without reliance on traditional databases and overcome the current bottleneck of the incomplete pre-built database.

Acknowledgment

This research is supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 1 (RG96/23). It is also supported by the National Research Foundation, Singapore, and DSO National Laboratories under the AI Singapore Programme (AISG Award No: AISG2-GC-2023-008); by the National Research Foundation Singapore and the Cyber Security Agency under the National Cybersecurity R&D Programme (NCRP25-P04-TAICeN) and CyberSG R&D Cyber Research Programme Office; and by the National Research Foundation, Prime Minister's Office, Singapore under the Campus for Research Excellence and Technological Enterprise (CREATE) programme. Any opinions, findings and conclusions, or recommendations expressed in these materials are those of the author(s) and do not reflect the views of National Research Foundation, Singapore, Cyber Security Agency of Singapore as well as CyberSG R&D Programme Office, Singapore.

References

[1] 2022. BlackDuck. https://www.synopsys.com/software-integrity/security-testing/software-composition-analysis.html.
[2] 2022. OWASP. https://owasp.org/www-project-dependency-track.
[3] 2022. Sonatype. https://www.sonatype.com/.
[4] 2024. GPT-4o. https://openai.com/index/hello-gpt-4o/.
[5] 2024. Mend.io (formerly WhiteSource) - Start Managing Application Risk. https://www.mend.io/ [Online; accessed 2025-01-17].
[6] 2024. Snky CLI. https://snyk.io/.
[7] 2025. Dependabot. https://github.com/dependabot.
[8] 2025. Scrapy. https://scrapy.org/.
[9] Sumaya Almanee, Arda Ünal, Mathias Payer, and Joshua Garcia. 2021. Too quiet in the library: An empirical study of security updates in android apps' native code. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1347–1359.
[10] Android Developers. 2020. Application Fundamentals. Archived from the original. https://developer.android.com/guide/components/fundamentals Retrieved on 3 December 2018 from Android Developers. Archived on 21 November 2020..
[11] Michael Backes, Sven Bugiel, and Erik Derr. 2016. Reliable third-party library detection in android and its security applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 356–367.
[12] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. 2024. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology* 15, 3 (2024), 1–45.
[13] Dify. 2025. Dify. https://dify.ai/.
[14] Jinchang Hu, Lyuye Zhang, Chengwei Liu, Sen Yang, Song Huang, and Yang Liu. 2024. Empirical Analysis of Vulnerabilities Life Cycle in Golang Ecosystem. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.
[15] Menghao Li, Wei Wang, Pei Wang, Shuai Wang, Dinghao Wu, Jian Liu, Rui Xue, and Wei Huo. 2017. Libd: Scalable and precise third-party library detection in android markets. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 335–346.
[16] National Telecommunications and Information Administration. 2022. Software Bill of Materials. Archived from the original. https://www.ntia.gov/SBOM Retrieved on 2021-01-25. Archived on 2022-11-30..
[17] Pypi. 2025. Pypi. https://pypi.org/.
[18] Kenneth Rosen, Douglas Host, Rachel Klee, and Richard Rosinski. 2007. *UNIX: The Complete Reference* (2 ed.). McGraw Hill Professional. 707 pages. Dynamically linked libraries are also called shared objects (.so). Retrieved on 2017-06-08..
[19] Wei Tang, Zhengzi Xu, Chengwei Liu, Jiahui Wu, Shouguo Yang, Yi Li, Ping Luo, and Yang Liu. 2022. Towards understanding third-party library dependency in c/c++ ecosystem. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–12.
[20] Tool Interface Standards Committee. 1995. *Executable and Linkable Format (ELF) Specification*. Specification. Tool Interface Standards Committee. Available from SCO, 190 River Road, Summit, NJ 07901, USA.
[21] Seunghoon Woo, Eunjin Choi, Heejo Lee, and Hakjoo Oh. 2023. {V1SCAN}: Discovering 1-day Vulnerabilities in Reused {C/C++} Open-source Software Components Using Code Classification Techniques. In *32nd USENIX Security Symposium (USENIX Security 23)*. 6541–6556.
[22] Seunghoon Woo, Sunghan Park, Seulbae Kim, Heejo Lee, and Hakjoo Oh. 2021. CENTRIS: A Precise and Scalable Approach for Identifying Modified Open-Source Software Reuse. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 860–872.
[23] Jiahui Wu, Zhengzi Xu, Wei Tang, Lyuye Zhang, Yueming Wu, Chengyue Liu, Kairan Sun, Lida Zhao, and Yang Liu. 2023. Ossfp: Precise and scalable c/c++ third-party library detection using fingerprinting functions. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 270–282.
[24] Yulun Wu, Zeliang Yu, Ming Wen, Qiang Li, Deqing Zhou, and Hai Jin. 2023. Understanding the Threats of Upstream Vulnerabilities to Downstream Projects in the Maven Ecosystem. In *45th International Conference on Software Engineering*. 1–12.
[25] Xian Zhan, Lingling Fan, Sen Chen, Feng We, Tianming Liu, Xiapu Luo, and Yang Liu. 2021. Atvhunter: Reliable version detection of third-party libraries for vulnerability identification in android applications. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1695–1707.
[26] Lyuye Zhang, Chengwei Liu, Sen Chen, Zhengzi Xu, Lingling Fan, Lida Zhao, Yiran Zhang, and Yang Liu. 2023. Mitigating persistence of open-source vulnerabilities in maven ecosystem. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 191–203.
[27] Lyuye Zhang, Chengwei Liu, Zhengzi Xu, Sen Chen, Lingling Fan, Bihuan Chen, and Yang Liu. 2023. Has My Release Disobeyed Semantic Versioning? Static Detection Based on Semantic Differencing. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering* (Rochester, MI, USA) *(ASE '22)*. Association for Computing Machinery, New York, NY, USA, Article 51, 12 pages. doi:10.1145/3551349.3556956
[28] Lyuye Zhang, Chengwei Liu, Zhengzi Xu, Sen Chen, Lingling Fan, Lida Zhao, Jiahui Wu, and Yang Liu. 2023. Compatible Remediation on Vulnerabilities from Third-Party Libraries for Java Projects, In Proceedings of the 45th International Conference on Software Engineering (Melbourne, Victoria, Australia). *arXiv preprint arXiv:2301.08434*, 2540–2552. doi:10.1109/ICSE48619.2023.00212
[29] Yuan Zhang, Jiarun Dai, Xiaohan Zhang, Sirong Huang, Zhemin Yang, Min Yang, and Hao Chen. 2018. Detecting third-party libraries in android applications with high precision and recall. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 141–152.
[30] Lida Zhao, Sen Chen, Zhengzi Xu, Chengwei Liu, Lyuye Zhang, Jiahui Wu, Jun Sun, and Yang Liu. 2023. Software Composition Analysis for Vulnerability Detection: An Empirical Study on Java Projects. In *Proceedings of the 2023 31th acm sigsoft international symposium on foundations of software engineering*.