The Eurasia Proceedings of Science, Technology, Engineering and Mathematics (EPSTEM)
ISSN: 2602-3199

The Eurasia Proceedings of Science, Technology, Engineering and Mathematics (EPSTEM), 2026

Volume 40, Pages 193-201

ICBAST 2026: International Conference on Basic Sciences and Technology

Improving Accuracy of OWASP Dependency-Check Through Optimized CPE Matching to Reduce False Positives and False Negatives

Wahyu Francesco Toldo Hutabarat
Bandung Institute of Technology

Yani Widyani
Bandung Institute of Technology

Abstract: In today's digital era, the use of open-source dependencies in modern software development has become commonplace. However, this practice increases security risks due to vulnerabilities hidden within the open-source components being used. Software Composition Analysis (SCA) is one of the approaches that can be utilized to detect and mitigate the risks arising from the use of open-source dependencies. Nevertheless, existing SCA tools still face a fundamental challenge in the form of false positives (reported vulnerabilities that are not actually relevant) and false negatives (vulnerabilities that remain undetected), which can degrade the accuracy of detection results and hinder security analysis as well as mitigation decisions. This study focuses on improving the accuracy of one widely used SCA tool, OWASP Dependency-Check, by highlighting one of its main sources of error: the Common Platform Enumeration (CPE) matching process between project dependencies and vulnerability entries in the Common Vulnerabilities and Exposures (CVE) database. The objectives of this research are to analyze CPE matching error patterns, design optimization mechanisms to improve the matching process, and evaluate the impact of these optimizations.

Keywords: Software composition analysis, Dependency-check, Open-source security, CPE matching, Vulnerability detection, Software engineering

Introduction

The use of open-source software (OSS) has become an essential part of modern software development. Many systems rely on external libraries and frameworks to accelerate development and reduce cost. However, this practice also introduces security risks, as vulnerabilities in third-party components can propagate across the software supply chain and affect the overall system security (Shu et al., 2025). As software systems become more complex, managing these risks has become a critical challenge.

Software Composition Analysis (SCA) tools are widely used to identify dependencies and detect known vulnerabilities in OSS components. These tools analyze project structures, extract dependency information, and match them with vulnerability databases such as the National Vulnerability Database (NVD) (Shu et al., 2025). One commonly used tool is OWASP Dependency-Check, which maps dependencies to vulnerabilities using the Common Platform Enumeration (CPE) format. Despite their importance, previous studies have shown that SCA tools still face significant accuracy issues, including false positives and false negatives (Imtiaz et al., 2021; Zhao et al., 2023). Existing research has mainly focused on improving detection techniques, such as enhancing static analysis, applying machine learning, or combining multiple analysis approaches. While these methods can improve detection performance, they often overlook the complexity of real-world scenarios, such as multi-language ecosystems, binary dependencies, and adversarial conditions (Shu et al., 2025). As a result, SCA tools may perform well in controlled environments but fail to maintain accuracy in more complex situations.

One critical component that is often overlooked is the matching process between detected dependencies and vulnerability records. In SCA tools, this process usually relies on heuristic-based CPE matching, including string similarity, vendor matching, and product name matching. However, these approaches can produce incorrect associations, especially when multiple candidate CPE entries exist. This may lead to irrelevant vulnerabilities being included (false positives) or relevant ones being missed (false negatives).

The limitations of current SCA tools indicate that accuracy issues are not only caused by detection weaknesses but also by problems in the matching mechanism. As highlighted by Shu et al. (2025), SCA tools still struggle to balance precision, recall, and robustness, particularly in complex environments. This suggests that improving only the detection stage is not sufficient to achieve reliable results. However, limited studies have specifically examined the impact of CPE matching accuracy on SCA results. Most research treats the matching process as a secondary step, even though it plays a key role as a bridge between dependency identification and vulnerability reporting. This creates a research gap, as improving the matching process has the potential to significantly enhance overall accuracy without modifying the detection engine.

The objectives of this research are threefold. First, to analyze common error patterns in the existing CPE matching mechanism. Second, to design an optimized matching approach. Third, to evaluate the effectiveness of the proposed approach using a ground truth dataset derived from the National Vulnerability Database. The proposed matching approach is implemented through four main components: artifact normalization, vendor prioritization, similarity scoring, and version filtering.

The results of this study demonstrate that optimizing the CPE matching process can significantly reduce false positives and false negatives and produce results that are closer to the ground truth. This finding highlights the importance of the matching layer in SCA systems and suggests that future improvements should consider both detection and matching mechanisms to achieve better accuracy.

Method

This study adopts an experimental approach to evaluate and improve the accuracy of vulnerability detection in SCA. The focus is on analyzing and optimizing the CPE matching process used by OWASP Dependency-Check. The method consists of four main stages: data collection, baseline extraction, ground truth construction, and optimized matching with evaluation.

Figure 1 illustrates the overall workflow of the proposed method. The process starts with data collection and baseline extraction using OWASP Dependency-Check. Then, a ground truth dataset is constructed from the NVD. After that, error analysis is performed by comparing baseline results with the ground truth to identify false positives and false negatives. Based on this analysis, the CPE matching process is optimized using four main components. Finally, the results are evaluated by comparing the detection accuracy before and after optimization.

Data Collection

The data used in this study consists of several open-source Java projects obtained from publicly available repositories. The selected projects include Apache Tomcat, Apache HttpClient, Mockito, Apache Commons Lang, Gson, Log4j, Jackson Databind, Netty, and Apache Struts. These projects were chosen to represent a diverse set of software dependencies, including widely used frameworks, utility libraries, and components with known vulnerability histories. The selection was not based on predefined error characteristics, but rather to ensure variation in project size, dependency structure, and usage context. For each project, vulnerability data is collected using OWASP Dependency-Check, which generates a JSON report containing detected dependencies and associated CVE entries. This report serves as the baseline data for further analysis.

In addition, vulnerability data from the National Vulnerability Database (NVD) is collected to construct a ground truth dataset. The ground truth is generated using exact CPE matching and version constraints to identify vulnerabilities that are truly applicable to each dependency. This combination of baseline data and ground truth enables a comparative evaluation of detection accuracy before and after the proposed CPE matching optimization.

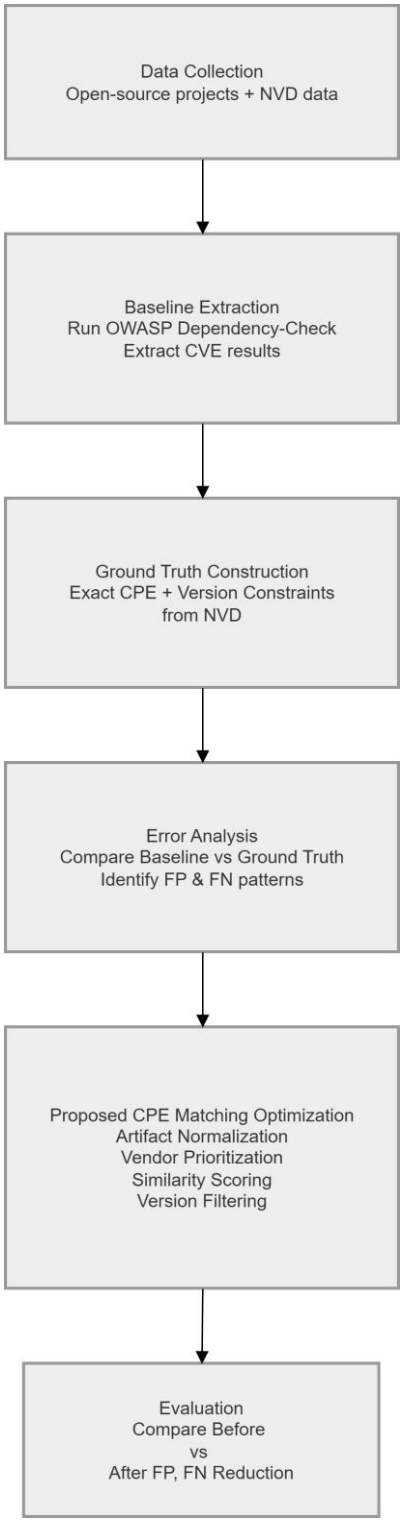

Figure 1. Proposed CPE matching workflow

Baseline Extraction

The first step is to extract vulnerability data from the raw output of the SCA tool. The JSON report generated by OWASP Dependency-Check is processed using a custom script (extract_odc.py) to retrieve all reported CVE entries for each dependency. Two types of data are collected:

- Total CVE entries (including duplicates),
- Unique CVE identifiers.

This step provides the baseline result of the SCA tool, which is later used for comparison with the optimized results.

Ground Truth Construction

To evaluate the accuracy of the SCA results, a ground truth dataset is constructed using vulnerability data from the National Vulnerability Database. The ground truth is generated using a custom script (build_ground_truth.py) based on the following criteria:

- exact matching of vendor, product, and version using the Common Platform Enumeration format,
- filtering based on version constraints such as versionStartIncluding and versionEndExcluding,
- inclusion of only vulnerabilities that are explicitly associated with the target software component.

This approach ensures that the ground truth represents a reliable reference for identifying valid vulnerabilities for each dependency.

Proposed CPE Matching Optimization

The core contribution of this study lies in improving the CPE matching process. The optimization is implemented using a custom script (enhanced_odc_matcher.py) that refines the mapping between dependencies and vulnerability entries.

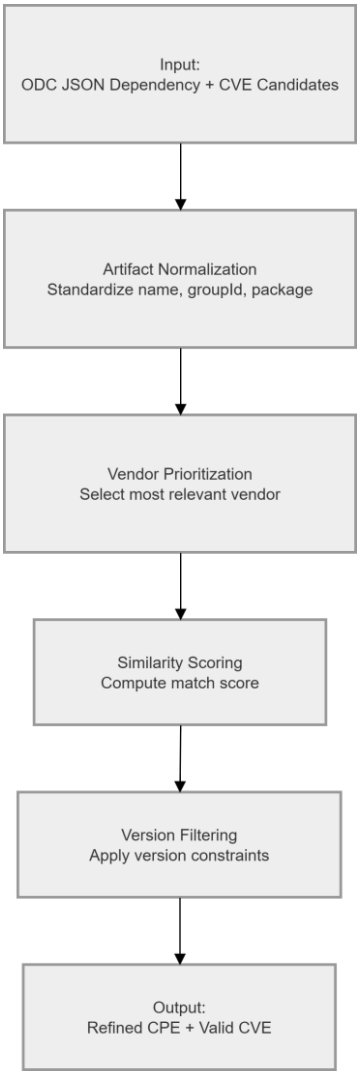

Figure 2. CPE matching optimization components

The proposed optimization consists of four main components, as shown in Figure 2. First, artifact normalization standardizes dependency metadata extracted from the OWASP Dependency-Check JSON report into a consistent format aligned with CPE representation. Second, vendor prioritization selects the most relevant vendor using rule-based mapping from groupId and known ecosystem patterns. Third, similarity scoring evaluates the relevance between dependency attributes and candidate CPE entries using a weighted scoring function based on name, vendor, and product similarity. Finally, version filtering ensures that only vulnerabilities applicable to the specific dependency version are retained based on version constraints from the NVD.

Artifact Normalization

This step standardizes the representation of metadata dependency information from JSON OWASP, including groupId, artifactId, name. The goal is to reduce inconsistencies between dependency metadata and CPE naming conventions, which often lead to incorrect matching.

Example (Before Normalization):

```
{
  "groupId": "org.apache.commons",
  "artifactId": "commons-lang3",
  "name": "commons-lang3-3.17.0.jar"
}
```

After Normalization:

```
vendor = apache
product = commons_lang
version = 3.17.0
```

Proposed Standard representation:

```
<vendor>:<product>:<version>
example: apache:commons_lang:3.17.0
```

Vendor Prioritization

In many cases, a dependency can be associated with multiple potential vendors in the CPE database. This component introduces a prioritization mechanism to select the most relevant vendor based on contextual information from the dependency, such as groupId and known ecosystem mappings. This helps reduce incorrect associations with unrelated vendors.

Example Candidates :

```
cpe:apache:commons_lang
cpe:oracle:commons_lang
cpe:netapp:commons_lang
```

This step uses a rule-based mapping. Vendor is selected based on GroupId.

Example rule:

```
org.apache.* → vendor = apache
com.fasterxml.* → vendor = fasterxml
```

Example:

```
groupId = org.apache.commons

The selected vendor is apache
```

Similarity Scoring

A similarity scoring mechanism is applied to evaluate the relevance between dependency attributes and candidate CPE entries. The weights are assigned using a heuristic approach, where the dependency name is considered the most important factor, followed by the vendor and the product. This is because the name usually provides the strongest indication of the correct match. The selected weights can be adjusted and further refined in future experiments.

Scoring Formula :

$$Score = 0.5 \times Similarity_{name} + 0.3 \times Similarity_{vendor} + 0.2 \times Similarity_{product}$$

Example:

```
Dependency: commons-lang3
CPE: commons_lang

Name similarity ≈ 0.9
Vendor match = 1
Product similarity ≈ 0.9

Score = (0.5 x 0.9) + (0.3 x 1) + (0.2 x 0.9) = 0.93
```

From the remaining candidates, the system selects the most representative CPE based on the highest valid score and contextual validation. Only the best candidate is selected.

Version Filtering

This step ensures that the matched vulnerabilities are applicable to the specific version of the dependency. Version constraints from the vulnerability database, such as versionStartIncluding and versionEndExcluding, are used to filter out irrelevant vulnerabilities that fall outside the valid version range.

Example from NVD:

```
versionStartIncluding = 2.0
versionEndExcluding = 3.0

Dependency Version
Version = 3.17.0 (not in range, then removed)
Version = 2.5 (In range, valid)
```

Only vulnerabilities that match the dependency version are retained.

Results and Discussion

Baseline vs Optimized Results

This section presents the comparison between the baseline results produced by OWASP Dependency-Check and the optimized results obtained using the proposed CPE matching approach. The evaluation is conducted across multiple open-source projects with varying characteristics.

Overall, the results show that the proposed method significantly improves the accuracy of vulnerability detection by reducing both false positives and false negatives. In several cases, the optimized results are fully aligned with the ground truth derived from the National Vulnerability Database.

The impact of the proposed CPE matching optimization on reducing false positives is presented in Table 1. Table 1 shows the comparison of false positive results before and after applying the optimized CPE matching method. In the baseline OWASP Dependency-Check, several datasets produced high false positive rates, such as Mockito and Struts, which reached 100%, and Commons Lang with 80%. After applying the optimization, all false positives were successfully removed across all datasets. This indicates that the proposed method can effectively filter irrelevant vulnerabilities and improve the accuracy of vulnerability detection.

Table 1. False positive reduction after optimization

Project	Baseline CVE Before	Ground Truth	FP Before	FP (%) Before	Enhance CVE After	FP (After)	FP (%) After
Tomcat 7.0.0	85	74	11	12.94%	74	0	0%
Mockito 5.15.0	4	0	4	100%	0	0	0%
Commons Lang 3.17.0	5	1	4	80%	1	0	0%
Jackson 2.4.0	48	47	1	2.08%	47	0	0%
Struts 2.3.1.1	6	0	6	100%	0	0	0%

After applying the proposed optimization, all false positives are successfully eliminated across all evaluated datasets, resulting in a false positive rate of 0%. This demonstrates that the optimized CPE matching approach is effective in filtering out irrelevant vulnerability matches by enforcing stricter validation on vendor, product, and version attributes. Consequently, the overall precision of the vulnerability detection process is significantly improved.

Table 2. False negative reduction after optimization

Project	Baseline CVE Before	Ground Truth	FN Before	FN (%) Before	Enhance CVE After	FN (After)	FN (%) After
Gson 2.8.0	0	1	1	100%	0	0	0%
Netty 3.9.0	0	3	3	100%	0	0	0%

The optimized CPE matching approach not only reduces false positives but also eliminates false negatives, ensuring that all relevant vulnerabilities are correctly identified. Table 2 shows the comparison of false negative results before and after applying the optimized method. In the baseline OWASP Dependency-Check, both Gson and Netty datasets failed to detect existing vulnerabilities, resulting in a false negative rate of 100%. After applying the optimization, all false negatives were successfully eliminated. This demonstrates that the proposed method improves the ability to identify vulnerabilities that were previously missed.

Conclusion

The proposed optimization consists of four main components, as shown in Figure 2. First, artifact normalization standardizes dependency metadata extracted from the OWASP Dependency-Check JSON report into a consistent format aligned with CPE representation. Second, vendor prioritization selects the most relevant vendor using rule-based mapping from groupId and known ecosystem patterns. Third, similarity scoring evaluates the relevance between dependency attributes and candidate CPE entries using a weighted scoring function based on name, vendor, and product similarity. Finally, version filtering ensures that only vulnerabilities applicable to the specific dependency version are retained based on version constraints from the NVD. This result indicates that the main source of error in OWASP Dependency-Check lies in the CPE matching stage rather than the detection stage. However, the approach relies on rule-based mapping, which may require adaptation for different ecosystems. The system selects only one most representative CPE from the remaining candidates based on the highest valid score.

Recommendations

Future work can evaluate the proposed method on larger and more complex software projects to test its scalability and robustness. Although the results show strong improvements, the current approach is mainly based on rule-based matching and predefined normalization strategies. Therefore, more advanced techniques, such as semantic analysis or machine learning, can be explored to handle more complex and ambiguous cases that are not covered by the current rules. In addition, integrating the proposed method into existing SCA tools can help improve their practical performance in real-world environments.

Scientific Ethics Declaration

* The authors declare that the scientific and ethical responsibility of this article belongs to the authors. This study does not involve human participants or animals and therefore does not require ethical approval.

Conflict of Interest

* The authors declare that they have no conflicts of interest

Funding

* This work received financial support from the Indonesia Endowment Fund for Education (LPDP), Ministry of Finance of the Republic of Indonesia, which fully funded the author's participation in the International Conference on Basic Science and Technology in Konya, Türkiye.

Acknowledgements or Notes

* This article was presented as an oral presentation at International Conference on Basic Science and Technology (ICBAST) (https://2026.icbast.net/). The conference will take place on April 2-5, 2026 in Konya, Türkiye.

* The author gratefully acknowledges the financial support provided by the Indonesia Endowment Fund for Education (LPDP), Ministry of Finance of the Republic of Indonesia, which fully funded participation in the *International Conference on Basic Science and Technology* held in Konya, Türkiye. The author also expresses sincere appreciation for the opportunity to present this work as an oral speaker at the conference.

References

Akremi, A. (2021). Software security static analysis false alerts handling approaches. *International Journal of Advanced Computer Science and Applications, 12*(11).

Aloraini, B., Nagappan, M., German, D. M., Hayashi, S., & Higo, Y. (2019). An empirical study of security warnings from static application security testing tools. *Journal of Systems and Software, 158*, 110427.

Banerjee, S., Cui, S., Emmi, M., Filieri, A., Hadarean, L., Li, P., Luo, L., Piskachev, G., Rosner, N., Sengupta, A., Tripp, O., & Wang, J. (2023). Compositional taint analysis for enforcing security policies at scale. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 1985–1996). New York, NY: ACM.

Böttner, L., Hermann, A., Eppler, J., Thüm, T., & Kargl, F. (2023). Evaluation of free and open source tools for automated software composition analysis. In *Proceedings of the Computer Science in Cars Symposium*. New York, NY: ACM. https://doi.org/10.1145/3631204.3631862

Cheirdari, F., & Karabatis, G. (2018). Analyzing false positive source code vulnerabilities using static analysis tools. In *2018 IEEE International Conference on Big Data (Big Data)* (pp. 4782–4788). IEEE.

Dietrich, J., Rasheed, S., Jordan, A., & White, T. (2023). *On the security blind spots of software composition analysis caused by cloning and shading*. arXiv:2306.05534.

Foo, D., Yeo, J., Xiao, H., & Sharma, A. (2019). *The dynamics of software composition analysis*. arXiv:1909.00973.

Guo, Z., Tan, T., Liu, S., Liu, X., Lai, W., Yang, Y., Li, Y., Chen, L., Dong, W., & Zhou, Y. (2023). Mitigating false positive static analysis warnings: Progress, challenges, and opportunities. *IEEE Transactions on Software Engineering, 49*(12), 5154–5173.

Imtiaz, N., Thorn, S., & Williams, L. (2021). A comparative study of vulnerability reporting by software composition analysis tools. In *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 1–11). New York, NY: ACM. https://doi.org/10.1145/3475716.3475769

Kharkar, A., Zilouchian Moghaddam, R., Jin, M., Liu, X., Shi, X., Clement, C., & Sundaresan, N. (2022). Learning to reduce false positives in analytic bug detectors. In *Proceedings of the 44th International Conference on Software Engineering*. New York, NY: ACM.

Marashdih, A. W., Zaaba, Z. F., & Suwais, K. (2023). An enhanced static taint analysis approach to detect input validation vulnerability. *Journal of King Saud University–Computer and Information Sciences, 35*(2), 682–701.

Medeiros, I., Neves, N., & Correia, M. (2016). Detecting and removing web application vulnerabilities with static analysis and data mining. *IEEE Transactions on Reliability, 65*(1), 54–69.

Ning, Y., Zhang, Y., Ma, C., Guo, Z., & Yu, L. (2024). Empirical study of software composition analysis tools for C/C++ binary programs. *IEEE Access, 12*, 50418–50430.

Ponta, S. E., Plate, H., & Sabetta, A. (2020). Detection, assessment and mitigation of vulnerabilities in open source dependencies. *Empirical Software Engineering, 25*, 3175–3215.

Reynolds, Z. P., Jayanth, A. B., Koc, U., Porter, A. A., Raje, R. R., & Hill, J. H. (2017). Identifying and documenting false positive patterns generated by static code analysis tools. In *2017 IEEE/ACM 4th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)* (pp. 54–61). IEEE. https://doi.org/10.1109/SER-IP.2017.20

Shu, C., Chen, W., Fan, G., Yu, H., Huang, Z., & Liang, Y. (2025). Tool or toy: Are SCA tools ready for challenging scenarios? *Computers & Security, 158*, 104624.

Wagner, J., Müller, S., Näther, C., Steghöfer, J.-P., & Both, A. (2025). Towards effective complementary security analysis using large language models. In *2025 IEEE International Conference on Intelligence and Security Informatics*. IEEE.

Wang, J., Cheng, S., Cao, J., & He, M. (2024). An empirical application of user-guided program analysis. *China Communications, 21*(7), 325–333.

Xu, Y., Zhang, C., & Pu, G. (2025). Revisiting the combination of static analysis error traces and dynamic symbolic execution: A potential approach for true positive confirmation. In *Proceedings of the 34th ACM SIGSOFT International Symposium on Software Testing and Analysis Companion* (pp. 124–132). New York, NY: ACM.

Yang, Y., Wen, M., Gao, X., Zhang, Y., & Sun, H. (2024). Reducing false positives of static bug detectors through code representation learning. In *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE. https://doi.org/10.1109/SANER60148.2024.00075

Zhao, L., Chen, S., Xu, Z., Liu, C., Zhang, L., Wu, J., Sun, J., & Liu, Y. (2023). Software composition analysis for vulnerability detection: An empirical study on Java projects. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, NY: ACM.

Author(s) Information

Wahyu Francesco Toldo Hutabarat
Bandung Institute of Technology
Jl. Ganesa No. 10, Indonesia
ORCID iD: https://orcid.org/0009-0004-9447-3446
*Contact e-mail: 23524028@mahasiswa.itb.ac.id

Yani Widyani
Bandung Institute of Technology
Jl. Ganesa No. 10, Indonesia
ORCID iD: https://orcid.org/0000-0002-8510-003X

*Corresponding author(s) email

To cite this article:

Hutabarat, W. F. T., & Widyani, Y. (2026). Improving accuracy of OWASP dependency-check through optimized CPE matching to reduce false positives and false negatives. *The Eurasia Proceedings of Science, Technology, Engineering and Mathematics (EPSTEM), 40*, 193–201.