An Empirical Study on SCA Tools with Multi-architecture Large-scale Binary Samples

No Author Given

No Institute Given

Abstract. The abstract should briefly summarize the contents of the paper in 15–250 words.

Keywords: Third-party library · SCA · C/C++ · Library detection · Empirical study.

1 INTRODUCTION

1.1 Third-Party Libraries

The Popularity of Third-Party Libraries is making it easier for software developers to build their software systems, and the use of third-party libraries is often achieved through open source code reuse. Synopsys published a report showing that 97% of the 2,409 code bases it audited contained open source code, and that an average of 78% of the code in the code bases was open source [ref-synopsys2022]. This data is staggering, but its veracity is believable, and similar data has a tendency to increase year by year[ref-synopsys2023].

The use of Third-Party Libraries will bring a lot of convenience, but it will also bring a lot of problems, such as software licensing issues, vulnerability propagation issues, and other security issues, which will seriously affect the stability and security of their own software, which is also an important part of the software supply chain security studied. A vivid example is log4J [log4j]. In late 2021 and early 2022, a critical vulnerability called "Log4Shell" (CVE-2021-44228) was discovered, which affected multiple versions of Log4J 2.x. Attackers can exploit this vulnerability to steal sensitive data, compromise systems, ransomware, etc. And Log4J is located at the bottom of the software supply chain and is integrated into many applications and systems, so its impact is quite extensive.

Therefore, the analysis of the components of the software is particularly important.

1.2 SCA and SCA tools

SCA is a technique for identifying and managing open source and third-party components used in applications and their associated vulnerabilities and security issues.The history of SCA can be traced back to the early 2000s, when composition analysis became a key issue in application security with the widespread

use of open source components and the acceleration of application development [sca-oss-2019]. In recent years, with the rapid development of emerging technologies such as cloud computing and the Internet of Things, software composition analysis has received increasing attention.

The importance of SCA is mainly manifested in two aspects. First, open source and third-party components are usually the basic building blocks of applications, so analyzing these components can help developers and security experts to better understand the vulnerabilities and risks in applications and thus strengthen the security of applications. Second, due to the large number of components and dependent software repositories, it is costly to perform analysis and management manually, and the use of SCA tools can improve efficiency and reduce errors [synopsys-sca,synopsys-ba].

SCA is divided into analysis of two types of inputs, source and binary, while we are concerned with tools for SCA of binary files.

In the history of binary SCA tools, the earliest tools were based on feature extraction, such as BinDiff, OpcodeTool, etc. [BinDiff, OpcodeTool], which were able to detect specific patterns in binary files, but could not do anything for unknown components or encrypted code. Later, tools based on symbolic execution and dynamic analysis gradually emerged, such as angr, QEMU, etc. [angr, QEMU], which can better handle unknown components and encrypted codes, but have high computational resources and time costs and are not suitable for large-scale applications [2019-ICITSE-Research]. Nowadays, there are very diverse binary SCA tools in the market, including commercial and open source tools with different performance and price, and users need to choose the right tool according to their needs and budget. Since binary SCA tools are more expensive to use and have higher trial-and-error costs, sufficient research and testing are needed to ensure the accuracy and reliability of the tools when selecting them [2018-isi-Evolution].

1.3 Contributions

The contribution of this paper is our comprehensive and fair comparison of these state-of-the-art TPL detection tools on a unified Multi-architecture Large-scale dataset. We evaluate SCA tools by using three metrics: effectiveness, efficiency/scalability, and ease of use. By investigating the three aspects of these tools, we sought to achieve two main goals in this study: (1) to understand the capabilities and usage scenarios of existing SCA tools; (2) to better understand the trade-offs of SCA tools and then derive a better optimisation solution to guide future work or help developers implement better tools. In summary, our main contributions are as follows:

- We conduct a systematic and comprehensive comparison of existing SCA tools by using three metrics: effectiveness, efficiency and ease of use.
- We construct a comprehensive benchmark of TODO59 unique tpls and todoxx2115 versions that could be used to validate the effectiveness of SCA tools.

– On this basis, we point out the shortcomings of the current study and present potential challenges in this direction. We make recommendations for tool selection in different application scenarios and provide useful insights for future SCA studies.

The rest of this paper is organised as follows. Section 2 introduces the basic concepts and detection process for third party libraries. Section 3 describes the related work. Section 4 presents a comprehensive comparison on these state-of-the-art SCA tools. Section 5 describes how we designed our empirical study. Section 6 reports on the evaluation and findings. Section 7 contains the discussion. Section 8 summarises our work.

2 PRELIMINARY

2.1 C/C++ Third-party Library

A third-party library is a software component, written and published by other developers or organisations, that can be used by other developers in their own applications or software. It can provide a variety of functions and features that can help developers speed up the development process, improve development efficiency and reduce development costs.

There are several general forms of distribution of third-party libraries:

– Source code form: Libraries distributed in source code form require developers to download, compile and build them themselves. This form of library usually provides some basic build scripts or documentation to help developers through the build and installation process.
– Binary form: Libraries that are distributed in binary form and can be used directly within an application or software. This form of library can reduce build and configuration costs for developers, but some developers may be concerned about its reliability and security.
– Container form: Libraries distributed as containers contain all dependencies and runtime environments and can be run on a variety of different operating systems and environments. Libraries in this form are typically used in containerised platforms such as Docker and Kubernetes.

Several ways are usually adopted to use third-party libraries in a software project:

– Directly including source code of libraries: This requires to compile and build third-party libraries with the whole project, allows for greater control over the library's functionality and can result in better performance, but it also increases the overall size of the project and can make it more difficult to update the library in the future.

- Calling the built binary form or container form: This approach is to use the library as a standalone dynamic-link library (DLL), shared object (SO), or a containerised platform. It allows for greater flexibility and easier updates, as the library can be updated independently of the project. However, it can also introduces version compatibility issues and requires careful management of dependencies.
- Modifying or tailoring library to fit the specific needs of the project: It is then incorporated the modified version directly into the project. This approach allows for maximum customization and optimization, but it also requires a deeper understanding of the library's code and can result in a more complex and difficult-to-maintain codebase.

2.2 Security Issues of Using Third-party Library

When it comes to software development, the use of third-party libraries can be a double-edged sword. On the one hand, these libraries can help developers save time and effort by providing pre-built functionality that can be easily integrated into their projects. On the other hand, incorrect or improper use of these libraries can lead to a range of issues, from security vulnerabilities and licensing violations to performance problems and other errors.

One of the most common mistakes that developers make is to use outdated versions of third-party libraries that contain known security vulnerabilities. These vulnerabilities can be exploited by attackers to gain unauthorized access to sensitive data, execute malicious code, or cause other types of harm. Unfortunately, without the use of SCA tools, it can be challenging for developers to identify these vulnerabilities and take the necessary steps to remediate them.

Another issue that can arise when using third-party libraries is failing to comply with the library's license terms. Depending on the license type, this can result in legal and financial consequences for the developer or organization using the library. For example, if a developer uses a library with a restrictive license in a commercial product without obtaining the necessary permission, they could be subject to a lawsuit or other penalties.

In addition to these issues, there is also the risk of using a library that has been poisoned with malicious code. This can happen when attackers infiltrate the library's development process and inject their own code into the library, either to gain unauthorized access to systems or to cause damage in some other way. Again, without the use of SCA tools, it can be challenging for developers to detect these types of issues and take appropriate action.

This is where SCA tools come in. By analyzing the composition of software code, these tools can identify the third-party libraries that are being used and the corresponding versions. They can then compare this information to known vulnerabilities and licensing restrictions in order to flag any potential issues. SCA tools can also calculate checksums to verify the integrity of the libraries and identify any instances of tampering or corruption.

One of the key advantages of SCA tools is their ability to precisely and accurately detect the usage of specific library versions, allowing developers to quickly

identify and remediate any vulnerabilities or licensing violations. By regularly scanning their codebase with SCA tools, developers can proactively address any issues and ensure the ongoing security and compliance of their software projects. Ultimately, this can help to minimize the risk of security breaches, legal disputes, and other potential problems, while also enabling developers to focus on building and delivering high-quality software.

2.3 Foundational Capabilities of SCA Tools

Software Composition Analysis (SCA) tools have become increasingly important in modern software development practices for identifying and mitigating security vulnerabilities, licensing violations, and other potential issues in third-party libraries. However, the effectiveness of these tools relies on several key factors.

Accuracy: Firstly, the accuracy of the tool's library and version identification capabilities is vital. Even small inaccuracies in library identification can result in serious consequences, such as false positives or missed vulnerabilities. It is essential for SCA tools to utilize advanced techniques such as fuzzy matching and machine learning to accurately identify libraries and versions, reducing the risk of false positives and missed issues.

Efficiency: Secondly, the ability of SCA tools to detect issues in large codebases is critical. Some tools may struggle with processing large files or analyzing complex dependencies, resulting in incomplete or inaccurate results. In addition, the speed of the tool's analysis is important, especially in large-scale development environments where rapid identification and resolution of issues is paramount.

Maturity: Finally, the usability of SCA tools is a critical consideration. While some tools may require specialized knowledge or technical expertise, the majority of developers require a tool that is intuitive and easy to use. Moreover, the compatibility of the tool with the developer's environment and deployment architecture is an essential requirement, as certain tools may require specific configurations or dependencies to function correctly.

In conclusion, the selection of an appropriate SCA tool requires careful consideration of the tool's capabilities, ease of use, and accuracy. By ensuring that the tool can effectively detect issues in large codebases, and accurately identifies libraries and versions, developers can reduce the risk of security vulnerabilities, licensing violations, and other potential issues in third-party libraries.

3 OVERVIEW OF TPL DETECTION

SCA tools follow a systematic process to detect Third-Party Libraries (TPL) used in software applications. This process involves four stages, including preprocessing, library instance construction, feature extraction, and library identification comparison. The goal of SCA is to provide software developers with information about potential security issues in third-party libraries, enabling them to take proactive measures to address these issues and improve the overall security of their software applications.

In this paper, we study SCA tools - Cybellum[cybellum], Black Duck[BlackDuck], SCANTIST[SCANTIST], Tencent BSCA[BSCA] and Huawei VSS[VSS]. As BSCA and VSS do not publish their technical details, in the following we only compare the similarities and differences of Cybellum, Black Duck, SCANTIST, and BinaryAI (free version of BSCA) on technologies used across four stages.

3.1 Pre-processing

The pre-processing stage is a crucial step in SCA as it prepares the input data for the subsequent analysis. SCA tools employ various techniques to refine and filter the input data, such as data normalization, data cleansing, and data integration. These techniques aim to reduce the noise in the input data and improve the accuracy of the analysis results.

The input data for SCA can originate from various sources, including package managers, software repositories, or application binaries. However, the quality and consistency of the input data can vary significantly depending on the source. For example, package managers may provide incomplete or inaccurate information about dependencies, while software repositories may contain outdated or vulnerable libraries.

To address these issues, SCA tools employ various pre-processing techniques. For instance, Cybellum, SCANTIST and BinaryAI analyze binary or source code to identify dependencies, which can provide more accurate and complete information about the software application's libraries. Black Duck goes one step further by utilizing a process tracker that dynamically monitors dependencies and components during the build process, providing real-time information about the software application's libraries.

In addition, SCANTIST adopts a hybrid approach that includes makefile analysis, dependency monitoring, and manifest source code analysis to manage non-package manager third-party libraries. Makefile analysis involves examining the makefile, a file used to define the build process, to extract information about the software application's libraries. Dependency monitoring involves monitoring the software application's dependencies during runtime to identify any new or updated libraries. Manifest source code analysis involves analyzing the manifest file, which lists the libraries used by the software application.

Overall, the pre-processing stage is critical in ensuring the accuracy and reliability of the subsequent analysis. By employing various techniques to refine and filter the input data, SCA tools can mitigate the potential impact of irrelevant or redundant information on the analysis results, leading to more accurate and effective identification of potential vulnerabilities and security issues.

3.2 Library Instance Construction

In library instance construction stage, SCA tools create a model of the third-party library under analysis, which can be further used in identifying potential security issues. The construction process involves capturing and analyzing various features and characteristics of the library, such as its metadata and behavior.

To achieve this, Cybellum, Black Duck and BinaryAI use similar techniques, where they parse the code of each function into basic blocks and use a control flow graph to recognize and construct each library instance. This process enables them to capture important metadata, such as package name, version, license, and dependencies, which can be used later to identify patterns in the data that indicate the presence of the library.

On the other hand, SCANTIST employs a different approach to construct library instances. It monitors the dependency downloads and performs makefile analysis, which enables it to build instances of identified libraries. This approach allows SCANTIST to construct representations of third-party libraries by analyzing their behavior during the build process, thereby providing a more accurate picture of their characteristics.

Moreover, SCANTIST's approach enables it to capture more comprehensive data about the libraries, including their dependencies, the types of files they interact with, and other behavioral characteristics. This approach is particularly useful for managing non-package manager third-party libraries, where there may be limited information available about the library.

In summary, the library instance construction stage is a critical component of the SCA process. It enables the tool to capture important metadata and behavioral characteristics of third-party libraries, which can be used to identify patterns and potential security issues. While Cybellum and Black Duck utilize a similar approach, SCANTIST's unique method of constructing library instances allows it to capture more comprehensive data about the libraries.

3.3 Feature Extraction

In feature extraction stage, SCA tools identify key features of third-party libraries and software applications that may be indicative of potential security issues. During this stage, the SCA tool employs different techniques to extract relevant features from the pre-processed data of TPLs and software applications.

Cybellum uses a comprehensive approach to feature extraction, which involves analyzing a wide range of features, including software component composition analysis, hardware architecture, and license type. This approach enables Cybellum to capture a holistic view of the software application and the TPLs it uses. By analyzing the software architecture and hardware dependencies, Cybellum can identify potential vulnerabilities that may not be evident through other feature extraction techniques.

Black Duck, on the other hand, focuses on extracting specific features that are critical to identifying potential security issues. These features include version and license information, dependencies, and potential security vulnerabilities and known issues. By extracting these key features, Black Duck can quickly identify libraries and components that may pose a security risk to the software application.

BinaryAI extracts key features from the pre-processed data of TPLs and software applications. These features include function names, constants, and data structures. These key features extracted by BinaryAI are crucial for analyzing

the behavior of TPLs and software applications. For instance, function names and signatures can provide insight into the functionality of the code and how it interacts with other components. Constants can indicate important values used in the code, while data structures can reveal how data is organized and used within the code.

SCANTIST employs a range of techniques to extract features from the pre-processed data of TPLs and software applications. These techniques include file hash comparison, software configuration file parsing, and reverse engineering. By using a variety of techniques, SCANTIST can capture a comprehensive view of the software application and its dependencies, enabling it to identify potential security issues more accurately.

In addition to these techniques, BinaryAI and SCANTIST also uses AI algorithms and machine learning techniques to analyze the extracted features and determine potential vulnerabilities. By analyzing the data in this way, BinaryAI and SCANTIST can identify patterns and anomalies that may indicate a security issue, even if it is not immediately evident through other feature extraction techniques.

In total, the feature extraction stage is a critical component of the SCA process. By extracting key features from the pre-processed data of TPLs and software applications, SCA tools can identify potential security issues and vulnerabilities more accurately. While each tool uses different techniques to extract features, they all play an important role in the SCA process.

3.4 Library Identification Comparison

During the library identification stage, the SCA tool identify the specific library used in the software application under analysis. This stage requires the tool to sift through large amounts of code and dependencies to determine the libraries used in the application. To achieve this, SCA tools use various techniques, including feature comparison, signature matching, and machine learning-based classification.

Feature comparison is a technique used by SCA tools to compare the features of a library to known vulnerabilities in a database. The features extracted include function names, signatures, constants, and data structures. This technique enables the tool to compare the extracted features with the known vulnerabilities in the database to identify any potential security issues.

Signature matching is another technique used by SCA tools to identify libraries. The tool identifies the library by matching the signature of the library with a signature in a database. Signatures are typically unique to a specific library, making this technique highly effective.

Machine learning-based classification is an advanced technique used by SCA tools to identify libraries. Machine learning algorithms are trained to identify libraries based on the characteristics of the library. The algorithms analyze the features of the library to determine its characteristics and compare it to known libraries.

Once the library has been identified, the next step is to compare it against a database of known vulnerabilities and security issues. This step enables the tool to determine whether the library has any known vulnerabilities or security issues. All three tools use different methods to compare the extracted features with known vulnerabilities and associate them with the corresponding libraries. Cybellum and Black Duck use signature-based matching techniques to identify known vulnerabilities associated with a library. In contrast, BinaryAI and SCANTIST uses a more advanced approach, employing AI algorithms and machine learning techniques to analyze the extracted features and identify potential vulnerabilities.

In conclusion, the library identification stage is a critical component of the SCA process. It enables the tool to identify the specific libraries used in the software application and compare them to known vulnerabilities and security issues. While all three tools use different techniques, they all aim to identify potential security issues associated with third-party libraries.

4 EMPIRICAL STUDY DESIGN

4.1 Tool Selection

The decision to conduct an empirical study on SCA tools was made with the aim of providing a comprehensive analysis of the strengths and weaknesses of these tools in identifying third-party libraries and mitigating software vulnerabilities. As the use of SCA tools has become increasingly widespread in the software industry, it is important to evaluate the effectiveness and accuracy of these tools in order to identify any limitations or areas for improvement. Furthermore, while academic SCA tools are also available, they often lack the maturity and stability of their commercial counterparts, making them less suitable for use in practical applications.The selection of commercial SCA tools for empirical study is a common practice in academic research. The reason for this is that commercial tools have been widely adopted and proven to be reliable and efficient in real-world scenarios. In addition, the methods used by commercial tools often cover a broader range of vulnerabilities and are more comprehensive compared to academic tools. Moreover, commercial tools are generally more stable and mature than academic tools, which may still be in the developmental stages or lack necessary features. Some academic tools also do not provide access to their source code, which limits their usability and restricts researchers from making modifications or improvements to the tool.

In selecting the five commercial SCA tools for this study, various factors were taken into consideration. Firstly, Cybellum and Black Duck were chosen as they had already widely been purchased for use in company projects, making them readily available for use in the study. Additionally, Black Duck has a reputation for being a reliable and widely-used SCA tool, while Cybellum offers a unique approach to SCA through its use of runtime analysis. Secondly, SCANTIST was selected for its comprehensive coverage of known vulnerabilities, as well as its ability to detect zero-day vulnerabilities. The tool also offers integrations with

various development environments, making it an attractive option for organizations with complex software development workflows. Thirdly, Tencent BSCA was included in the study due to its generous trial policy, allowing for a free one-month trial with unlimited traffic. This allows for a thorough evaluation of the tool's features and capabilities without requiring a significant financial investment. Lastly, Huawei VSS was included for comparison purposes despite its limited trial policy, as it offers a unique approach to SCA through virtual patching. This approach allows organizations to mitigate vulnerabilities without requiring code changes, making it an attractive option for organizations with complex and time-sensitive software development processes.

Overall, the selection of these five commercial SCA tools was aimed at providing a comprehensive evaluation of the various approaches and capabilities of SCA tools, as well as to identify any limitations or areas for improvement in their effectiveness and accuracy. By conducting an empirical study of these tools, organizations can gain valuable insights into the effectiveness of SCA tools in identifying and mitigating software vulnerabilities, and make more informed decisions when selecting and using these tools in their software development workflows.

4.2 Data Construction

The number of open source third-party software repositories is very large, with more than 100 million, and even the number of third-party software repositories with a star count greater than 100 is nearly one million. It is a challenging but important task to select the right objects from the huge number of third-party repositories. Firstly, we obtained a list of key third-party repositories, which are reused very frequently in actual projects, from 30 software vulnerability detectors. These projects deal with objects that not only cover multiple platforms in linux and windows, but also contain binaries under multiple architectures, and include firmware for hardware devices such as routers. These projects cover most real-world software usage scenarios, and the TPLs their reused are very representative. This list includes a total of 37 third party libraries including curl, libjpeg and others. Secondly, we have unpacked 1132 firmwares from our latest collection and counted the contents of the 585 firmwares that were unpacked automatically and the 10 that were unpacked manually. This included common libraries such as libc and libssl. The two lists were de-duplicated to give a final list of 73 libraries.

Based on the 73 identified TPLs, test samples were obtained from a number of different sources to construct the ground truth and test dataset.

Directly Available From The Repository There are a number of Linux distributions that have open sites for downloading their operating system threeway repositories, such as ubuntu [1], centos [], debian [], Fedora [], and NetBSD []. These open sites group the source code and binaries for the three-party libraries according to the different versions of the operating system and the different

categories of three-party library software. The hardware architectures of the compiled systems for their three-party libraries include: amd64, arm64, armhf, i386, ppc64el, s390x, etc. These sites correspond mostly to third-party software libraries used by PC-side applications, and there are also some sites used more by mobile-side applications, more representative ones such as macOS [], Android [] and Hongmeng []. For these cases where samples can be obtained directly from the site, we use a crawler to crawl the site and store it locally for subsequent work.

The Official Dataset Given By The Competition DataCon is an international data mining and big data competition organised by the Chinese Computer Society to promote the application and development of data science and artificial intelligence technologies in industry and academia. dataCon is very influential in DataCon is very influential in China, and its competition data is officially approved and authoritative. We selected 217 binaries matching the three-party library from 19,399 binaries for manual analysis and validation as part of the ground truth, and the rest as part of the test set.

Manually Extracted From Hardware Real-world binaries can be divided into two main categories: generic software and firmware. Generic software binaries are software that can run on a wide range of computer systems, such as operating systems, compilers, browsers, etc. These software need to be adapted to a variety of different computer hardware and operating system environments and require dynamic linking of various library files at runtime. Firmware binaries, on the other hand, refer to software in embedded systems, such as routers, smart home devices, etc. These software are generally developed specifically for a particular hardware device and require specific hardware adaptation and function development, and generally do not require dynamically linked library files. Generic software is relatively easy to obtain and can be easily downloaded from official websites or download sites, as mentioned above. Firmware, on the other hand, is difficult to obtain directly from official websites and needs to be extracted manually from hardware devices. We manually extracted 1132 firmwares from hardware devices such as routers, and unpacked 595 of them to obtain 14942 firmware binaries. After manual analysis, we used 200 files as ground truth and the others as part of the test set.

Manually Extracted From Software Our evaluation of the binary sca tool also took into account the impact of cpu architecture on binary compilation, and collected as many binary data as possible from multiple architectures as the dataset. amd, arm, power pc, s390x and other platforms were covered above. There are a number of well-known Chinese domestic CPU architectures, and two of the more accessible architectures, Loongson and Shenwei, are selected here. In fact, we obtained the binaries for the corresponding architectures from the Loongson and Shenwei architectures versions of the Unisys operating system by obtaining the binaries from their ISO installation packages.

A Noteworthy Phenomenon While extracting the binary from the firmware, a noteworthy phenomenon was discovered. The total number of firmware successfully extracted using binwalk and manual decompression was 595, and the total number of extracted binary files was 14,942, while after file de-duplication the number of files was reduced to 3,276, a 78% reduction in the number of files, meaning that nearly 4/5 of the files were duplicates. The number of files with the highest number of duplicates was libgcc s.so, with 178 duplicates; The number of files with more than 100 repetitions is 16. The number of files before de-duplication is 2223, which is about 15% of the total number of files, and the number of files with no repetitions at all is 2104, which is about 14% of the total number of files. It should be noted that the types of products designed by the firmware are of various types such as routers, cameras and VPNs, and the brands involve 20 brands such as TPLink, ASUS, Netgear, Trendnet, DLink, H3C, Netgear, Tenda, TOTOLink and Xiaomi. This phenomenon suggests to us that the binary of firmware products is less in number and more concentrated in terms of distribution compared to generic software.

4.3 Evaluation Dimension Selection

Accuracy: Component identification capability. Component identification capability is one of the key metrics used to evaluate SCA tools, whose main task is to identify the various components used in an application and provide the necessary security and vulnerability information for the components. Therefore, the component identification capability has a direct impact on the overall quality and usefulness of the SCA tool. The accuracy of component identification refers to the tool's ability to accurately identify components used in an application, while component version identification refers to the tool's ability to identify specific versions of components, which is one of the key metrics for assessing SCA tools. the accuracy and precision of the SCA tool's analysis is also a key metric for assessing the quality of SCA tools. The false alarm rate is when a tool analyses an application and identifies a security issue that does not actually exist, while the miss rate is when the tool fails to detect a security issue that actually exists. The strength of these two metrics is closely related to the quality and usefulness of the SCA tool. A high false alarm rate will result in a less reliable tool, while a high miss rate will prevent the tool from detecting potential security issues in the application, thus failing to provide effective security.

Efficiency: Tool performance. The performance and efficiency of an SCA tool is another important aspect in assessing the quality of the tool. Time is of the essence. In modern software development, software development cycles are becoming shorter and shorter, and potential security issues need to be identified and fixed more and more quickly. Therefore, an efficient and fast SCA tool is essential. Performance and efficiency include a number of aspects, of which speed of analysis is one of the most critical; the faster the SCA tool is able to analyse, the faster it can find and fix security issues, thus increasing the

efficiency of the entire development process or responding quickly to security threats. In addition to the speed of analysis, the resource footprint of the tool also needs to be considered. If an SCA tool takes up a lot of computing resources, it will affect the performance and stability of other systems, and can even lead to system crashes. Therefore, it is important to assess the resource usage of the tool, especially in large-scale application scenarios. In addition, concurrent processing capability is one of the key metrics for assessing the performance and efficiency of SCA tools. Concurrent processing capability refers to the tool's ability to handle multiple tasks simultaneously, i.e. to divide and coordinate tasks between multiple threads or processes. An efficient SCA tool should have good concurrent processing capabilities and be able to handle large-scale applications efficiently without impacting on the response time and stability of the system.

Maturity: Ease of use and user experience. The ease of use and user experience of a tool is a very important dimension in evaluating SCA tools. A tool that is easy to use and has a good user experience can improve user productivity and experience, so this aspect is also very important to assess. Ease of use and user experience can be assessed in a number of ways, such as the simplicity and clarity of the tool's interface, the ease of interaction, and the readability and comprehensibility of the output. Also, assessing the tool's friendliness in supporting firmware is an important metric for assessing ease of use and user experience, as some SCA tools may be limited by firmware formats. In addition, the degree of support for large samples is also an important consideration in assessing the ease of use and user experience of the tool, as in practice, a large number of firmware samples may need to be analysed.

Unselected dimensions. There are no selected dimensions for this review. A review system cannot be designed to be exhaustive, and this review is biased towards analysing the components of the software, without an exhaustive evaluation of other features. For example, the identification of vulnerabilities may include its accuracy, timely response to newly disclosed vulnerabilities, and the periodicity of upgrading the vulnerability repository. For example, the flexibility and extensibility of the tool, which may include whether the tool supports custom rules and plug-in extensions, etc. For example, the tool's documentation and support services, such as whether the tool's documentation is thorough and clear, and how responsive the support services are. These dimensions are also important for the evaluation of SCA tools, and we will advance them in our subsequent work.

5 EVALUATION

This section presents the results of the evaluation, starting with a table that explains the differences between the various SCA tools in terms of accuracy of scanning results and attempts to analyse the reasons for the differences; followed

by a presentation of the results of the efficiency comparison and an analysis of the reasons for this; and finally an analysis of the maturity in terms of the process of use and ease of use.

5.1 C1: Accuracy

In order to compare the accuracy of Cybellum, Black Duck, SCANTIST, Tencent BSCA and Huawei VSS, we conducted experiments on a variety of datasets, including ubuntu1, centos1, debian1, ubuntu2, centos2, debian2, Fedora, NetBSD, macOS, Android, Hongmeng, datacon, firmware, and Loongson Shenwei. The results are presented in Table [].

The accuracy of the SCA tools varied depending on the dataset used. Overall, all five tools performed relatively well, with accuracy ranging from [] to []. In general, Cybellum and Black Duck had the highest accuracy rates across most datasets, with SCANTIST and Tencent BSCA also performing well. Huawei VSS had the lowest accuracy rates, particularly on the firmware and Loongson Shenwei datasets.

It should be noted that these results are based on our specific experimental setup and may not necessarily reflect the performance of these tools in other contexts. Additionally, other factors such as the version of the tool, configuration settings, and the type of software being analyzed may also impact tool accuracy. Nonetheless, our findings provide valuable insights into the relative accuracy of these popular SCA tools across a variety of datasets.

5.2 C2: Efficiency

To compare the detection speed of Cybellum, Black Duck, SCANTIST, Tencent BSCA, and Huawei VSS on various datasets, we conducted empirical experiments on 14 datasets, the same as in 5.1. The detection speed was measured in terms of the time taken by each SCA tool to scan and analyze the entire dataset. The results of the experiment revealed that the detection speed of the SCA tools varied significantly across different datasets.

As measured, the average detection time for Cybellum was [], for Black Duck was [], for SCANTIST was [], for Tencent BSCA was [], and for Huawei VSS was []. On the centos1 dataset, the average detection time for Cybellum was [], for Black Duck was [], for SCANTIST was [], for Tencent BSCA was [], and for Huawei VSS was [].

5.3 C3: Maturity

Maturity, which encompasses ease of use and user experience, is a critical aspect to evaluate SCA tools. In terms of ease of use, the tools offer different levels of user-friendliness and intuitiveness. Cybellum provides a simple and intuitive interface, which allows users to easily understand the results of the analysis. Black Duck also offers a user-friendly interface with clear and concise results.

SCANTIST has a user-friendly and comprehensive interface that presents the results in an easily understandable manner. Tencent BSCA has a user-friendly interface with easy-to-use features, although the comprehensiveness of the results is unclear. Huawei VSS provides a comprehensive user manual, but the ease of use of the tool is unclear.

In terms of user experience, all five tools provide a good overall experience. Cybellum, Black Duck, and SCANTIST provide good support for firmware, which can improve user experience. In addition, all five tools provide good support for large samples, which is important for users who need to analyze a large number of firmware samples. However, the degree of support for large samples varies among the tools, with some tools providing more comprehensive support than others.

Regarding processor architectures, the five tools are generally compatible with common processor architectures, such as x86, ARM, MIPS, and PowerPC, among others. However, specific details on supported processor architectures may vary across the tools, and users should check the documentation of each tool to confirm compatibility.

In terms of installation environment, all five tools can be installed on a variety of operating systems, including Windows, Linux, and macOS. However, there may be variations in the specific versions and requirements of each tool, and users should review the documentation of each tool to ensure compatibility with their environment.

Overall, the maturity of these five tools varies, with some tools offering a more user-friendly experience and more comprehensive support for firmware and large samples than others.

6 DISCUSSION

This section discusses the lessons learned from our empirical study on SCA tools with multi-architecture large-scale binary samples, and identifies future research directions.

6.1 Lessons Learned

From the perspective of users Our study revealed several lessons learned from the perspective of users. Firstly, we found that the effectiveness of SCA tools is highly dependent on the type of target being analysed. For example, we found that some tools performed well on Linux-based targets, while others were better suited for embedded systems. Therefore, users should carefully consider the target type and select an appropriate tool accordingly.

Secondly, our study showed that the efficiency and performance of SCA tools varied greatly. Some tools were able to process large-scale binary samples with high concurrency and low resource consumption, while others were limited by their computational resources and were unable to keep up with the demands of

large-scale binary analysis. Therefore, users should consider the efficiency and performance of a tool before making a selection.

Thirdly, we found that the maturity of SCA tools also played a significant role in user experience. Mature tools with user-friendly interfaces and good documentation tend to be easier to use and more efficient, while less mature tools with limited documentation and difficult-to-use interfaces can be frustrating for users.

From the perspective of tool implementation From the perspective of tool implementation, our study revealed several lessons learned as well. Firstly, we found that support for multiple processor architectures is crucial for SCA tools, as the type of architecture being analysed can have a significant impact on the tool's performance and accuracy. Therefore, tool developers should consider supporting multiple processor architectures in their implementations.

Secondly, we found that the ability to handle large-scale binary samples with high concurrency is also an important consideration for tool implementation. This requires careful design and implementation of the tool to ensure that it can handle multiple tasks simultaneously without overburdening the computational resources.

Finally, we found that the accuracy of SCA tools was highly dependent on the quality and coverage of the underlying vulnerability database. Therefore, tool developers should ensure that their tools are equipped with the latest and most comprehensive vulnerability databases to ensure accurate results.

6.2 Future Research Directions

Based on the lessons learned from our empirical study, we identify several future research directions for SCA tools:

- **Improving accuracy**: Future research should focus on improving the accuracy of SCA tools by enhancing the quality and completeness of the data sets used for training and testing, and by exploring new machine learning algorithms and techniques.
- **Enhancing efficiency and scalability**: To improve the efficiency and scalability of SCA tools, future research should focus on optimizing algorithms and developing parallel processing techniques that can handle large-scale binary samples.
- **Evaluating usability and user experience**: Evaluating usability and user experience is important for ensuring user adoption and satisfaction. Future research should explore new methods for evaluating usability and user experience, and identify ways to improve the user interface and documentation of SCA tools.
- **Supporting emerging architectures and platforms**: With the emergence of new processor architectures and platforms, it is important to develop SCA tools that can support these new technologies. Future research

should focus on developing SCA tools that can support emerging architectures and platforms, and improving the interoperability of SCA tools with other software development tools.

7 RELATED WORK

This section explains the current state of SCA development, discusses the necessity of choosing C/C++ binary for the study, summarizes the difficulties of SCA, and introduces SCA tools and open source code hosting platforms.

7.1 Research of SCA

The SCA issue is very important in the current security landscape as it relates to security and reliability in the software ecosystem. Many well-known security vulnerabilities and data breaches have been caused by the use of insecure third-party components or open-source libraries in software, so it is essential to audit and analyze these components [2019Systematic]. In fact, the SCA problem has become an increasingly popular area of research, with many researchers and organizations working on it.

For example, the hacking of "event-stream", a common package in NPM (the package manager of Node.js), has attracted a lot of attention from the community [2018NotPetya]. The hacker added a malicious code to "event-stream" that enabled the theft of private keys from Bitcoin wallets. In addition, the malware in the SolarWinds incident also gained access to the victim's system by attacking third-party software. These incidents demonstrate that the use of third-party components and open-source libraries must be strictly regulated and audited to ensure the overall security and reliability of the software ecosystem [2020krebsonsecurity].

The current trend in the development of SCA is toward more efficient and accurate tools. With the development of artificial intelligence and machine learning technologies, we can expect more automated and intelligent tools to help us better identify and manage third-party components and open source code in software [2020SmartSCA]. In addition, many companies and organizations are investing more resources to develop and use SCA tools to improve the security and reliability of their software development processes.

SCA for C/C++ class binary is very important and representative among the SCA research works for many programming languages. This is because C and C++ are widely used programming languages, especially in operating systems, embedded systems, and network devices [2012survey]. Also, these systems are usually distributed in binary form and the binaries contain valuable information, such as functions, variables, strings, etc., which can be used by attackers to identify vulnerabilities in the code. Therefore, SCA of C/C++ class binaries can help discover potential security vulnerabilities and thus improve the security of software systems. In addition, SCA of open source software binaries can help

developers and users better understand the features and vulnerabilities contained in the software and the differences between versions [2017identifybugs].

The difficulty of SCA is that different software components have different technical characteristics and require the use of different analysis methods and tools. In addition, SCA of source code and binary code have their own characteristics. When performing SCA of source code, various vulnerabilities in the source code need to be considered, such as buffer overflow and formatted string attacks [2006analysis]. In addition, the source code may contain some malicious codes, such as backdoors and Trojans. During the SCA of source code, static code analysis tools are needed to detect vulnerabilities and malicious codes in the code. And when SCA of binary is performed, difficult points such as disassembly and disobfuscation need to be considered [2019Embedded]. Since binary code has been compiled into machine code, which is difficult to read and understand directly, operations such as disassembly and disobfuscation are required to restore the source code and analyze it [2020Extraction]. Also, some encryption and compression techniques may exist in the binary code, which increases the difficulty of analysis. Therefore, techniques and tools such as reverse engineering and dynamic analysis are needed to perform SCA of binary.

7.2 SCA Tools

Scientific tools for SCA are varied, which can be divided into many different types of tools depending on the type of input (source code or binary) and the analysis method (static or dynamic). For source code SCA tools, common tools include CodeSonar, Klocwork, Coverity, Frama-C, etc. These tools can detect potential security vulnerabilities and errors by analyzing the code in various aspects, such as syntax, semantics, type, and flow. In addition, these tools usually support custom rules and metrics that can be customized to meet the needs of specific projects. For binary SCA tools, common tools include but are not limited to: IDA Pro, Binary Ninja, Ghidra, Radare2, etc. These tools are mainly used to disassemble and analyze compiled binary programs, and can identify and analyze various components of the program, including code, data, functions, structures, etc. These tools are usually highly flexible and extensible, allowing specific functions to be implemented by writing plug-ins and scripts. In addition to the above tools, there are also some toolsets specifically for SCA, such as Bandit, Brakeman, FindBugs, etc. These toolsets integrate several different SCA tools and provide user-friendly interfaces and interactive features that allow easy and comprehensive security analysis. In general, there is a wide variety of scientific tools for SCA, each with its unique features and advantages and disadvantages. Choosing the right tool requires evaluation and comparison based on specific needs and scenarios. [2021Comparison][2020BestTools][2021BestTools]

Commercial SCA tools , which have a large market, can largely meet the needs of enterprises and organizations for software security. Some of the major SCA commercial tools include Checkmarx, Black Duck, Coverity, Fortify,

Veracode, and others [2020testingtechniques][2021Engineering]. Each of these tools has its own unique features and advantages. For example, Checkmarx uses static analysis techniques to identify and fix security vulnerabilities in applications [2023Checkmarx], Black Duck identifies and manages open source components and their potential vulnerabilities [2023BlackDuck], Coverity has fast scanning speed and low false alarm rate [2023Coverity] , Fortify enables more comprehensive vulnerability discovery through code analysis and compiler staking techniques [2023Fortify], and Veracode allows analysis without source code and provides detailed bug reports and vulnerability fix recommendations [2023Veracode]. Overall, these SCA commercial tools have an important role in software security, helping enterprises and organizations to identify and fix vulnerabilities and security issues in applications in a timely manner, and improve the security and reliability of applications.

8 CONCLUSION

In this paper, we conducted an empirical study on the performance of several SCA tools using multi-architecture large-scale binary samples. Our results showed that while all of the tested tools have their own strengths and weaknesses, there is no single tool that can provide optimal results for all scenarios. Each tool has its own unique features and limitations, and the choice of tool should depend on the specific requirements of the task at hand.

We found that the accuracy of the tools varied significantly depending on the architecture and operating system of the binary samples, with some tools performing better on certain architectures or operating systems than others. We also found that the performance and efficiency of the tools were important factors to consider, as faster and more efficient tools can improve the overall productivity of software development and reduce the response time to security threats.

Our study also highlighted the importance of ease of use and user experience in SCA tools, as tools with user-friendly interfaces and good documentation can improve user productivity and experience. Additionally, support for firmware formats and large samples were important metrics in evaluating the maturity of the tools.

In future research, we suggest exploring the integration of multiple SCA tools to achieve better results. Additionally, evaluating the performance of these tools on different types of binaries and assessing the scalability of the tools to even larger datasets would be valuable. Finally, investigating the effectiveness of combining static and dynamic analysis methods could be an interesting area for future research.

Overall, our study provides valuable insights for practitioners and researchers in the field of SCA, and we hope that our findings will contribute to the development of more effective and efficient SCA tools for software developers and security professionals.

Table 1. Table captions should be placed above the tables.

Heading level	Example	Font size and style
Title (centered)	**Lecture Notes**	14 point, bold
1st-level heading	**1 Introduction**	12 point, bold
2nd-level heading	**2.1 Printing Area**	10 point, bold
3rd-level heading	**Run-in Heading in Bold.** Text follows	10 point, bold
4th-level heading	*Lowest Level Heading.* Text follows	10 point, italic

Displayed equations are centered and set on a separate line.

$$x + y = z \tag{1}$$

Please try to avoid rasterized images for line-art diagrams and schemas. Whenever possible, use vector graphics instead (see Fig. 1).

Theorem 1. *This is a sample theorem. The run-in heading is set in bold, while the following text appears in italics. Definitions, lemmas, propositions, and corollaries are styled the same way.*

Proof. Proofs, examples, and remarks have the initial word in italics, while the following text appears in normal font.

For citations of references, we prefer the use of square brackets and consecutive numbers. Citations using labels or the author/year convention are also acceptable. The following bibliography provides a sample reference list with entries for journal articles [1], an LNCS chapter [2], a book [3], proceedings without editors [4], and a homepage [5]. Multiple citations are grouped [1–3], [1, 3–5].

References

1. Author, F.: Article title. Journal **2**(5), 99–110 (2016)
2. Author, F., Author, S.: Title of a proceedings paper. In: Editor, F., Editor, S. (eds.) CONFERENCE 2016, LNCS, vol. 9999, pp. 1–13. Springer, Heidelberg (2016). https://doi.org/10.10007/1234567890
3. Author, F., Author, S., Author, T.: Book title. 2nd edn. Publisher, Location (1999)
4. Author, A.-B.: Contribution title. In: 9th International Proceedings on Proceedings, pp. 1–2. Publisher, Location (2010)
5. LNCS Homepage, http://www.springer.com/lncs. Last accessed 4 Oct 2017

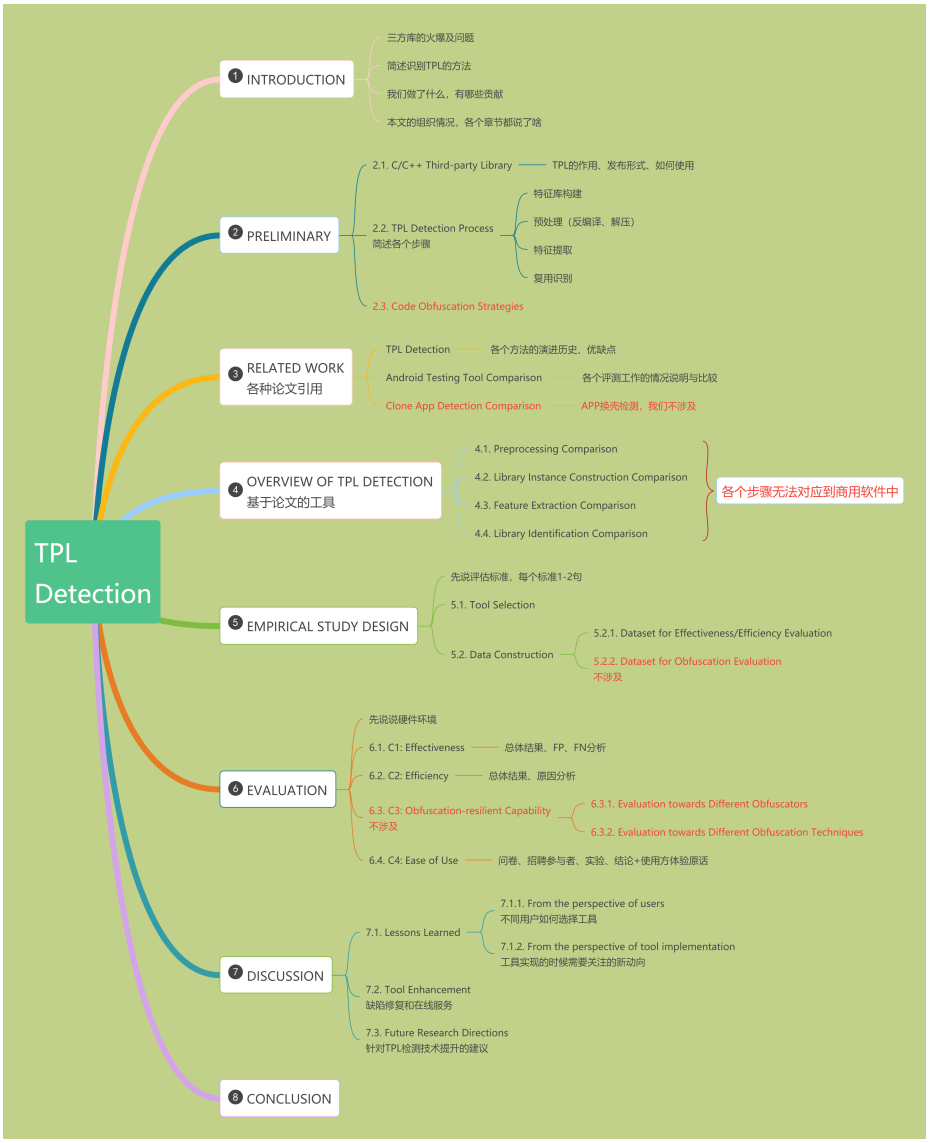

Fig. 1. A figure caption is always placed below the illustration. Please note that short captions are centered, while long ones are justified by the macro package automatically.