國立臺北科技大學
National Taipei University of Technology

資訊工程系碩士班

碩士學位論文

Department of Computer Science &
Information Engineering
Master Thesis

利用靜態分析強化自動化測試工具以改善行動應用程式品質

Enhancing Automated Testing Tools with Static Analysis for Improved Mobile Application Quality

研究生：余承諺

Researcher: Yu, Cheng-Yan

指導教授：陳香君 博士

Advisor: Chen, Shiang-Jiun, Ph.D.

July 2025

國立臺北科技大學
研究所碩士學位論文口試委員會審定書

本校 資訊工程 研究所 余承諺 君

所提論文，經本委員會審定通過，合於碩士資格，特此證明。

學位考試委員會

委　　員：吳和庭

馬奕葳

陳香君

指導教授：陳香君

所　　長：劉建宏

中 華 民 國　一百一十四　年　七　月　九　日

ABSTRACT

Keywords: Static Application Security Testing, Android, Detection

This research aims to develop LiSD (Light Weight Software Detection), an efficient and practical tool designed to detect vulnerabilities in Android mobile applications. LiSD uses static application security testing (SAST) as its core detection method. LiSD stands out for its lightweight nature and flexibility, ensuring efficient performance with minimal overhead. It also integrates seamlessly into developer workflows. Quantitatively, LiSD averages around 10 minutes of processing time and approximately 10 GB of peak memory consumption for most APKs during typical operations.

To evaluate LiSD's usability and effectiveness, we will conduct an empirical analysis on Android packages (APKs) from two sources. From Google Play, we will select the ten most downloaded applications across various categories based on popularity. From GitHub, we will choose open-source projects recognized for their high reputation and community trust, specifically those with the highest star ratings and a significant number of active contributors.

We will assess LiSD's performance using key metrics such as analysis time and memory consumption to ensure its suitability for practical deployment. A key focus of this assessment will be demonstrating LiSD's flexible scan capabilities, which are achieved through script-driven automation to address diverse vulnerability detection scenarios.

TAIPEI TECH

Acknowledgements

這篇論文能順利完成，首先要非常謝謝我的指導教授陳香君博士。在研究跟寫論文的這段時間，她給了我很大的幫助跟很多好建議，讓我在參與計畫的時候學到超級多東西，也累積了不少經驗。再來，我要謝謝我的口試委員吳和庭博士跟馬奕葳博士。口試的時候，兩位老師很精準地指出了論文的問題，也提供了很多很棒的修改方向，讓這篇論文變得更完整、更仔細。我也要特別謝謝大學專題的指導教授石佳宏博士。在學 Android 的過程中，他給了我很多實用的建議和啟發，也讓我發現，實驗室不只是一個做研究的地方，它也可以是一個讓人覺得很溫暖、很放鬆的避風港。

在讀書的這段時間，很謝謝研究所的同學們、朋友們還有學弟們，不管是在學術上討論，還是生活上的大小事，你們都幫了我很多忙，給了我很大的支持。也很謝謝大學實驗室的夥伴們，不只介紹我認識了很多好用的軟體和工具，下課後還會陪我一起玩遊戲，這些都是我讀書生涯裡很美好的回憶。

最後，我想謝謝我的家人們，你們一直給我很多意見，也不斷支持我，讓我可以很安心地把學業完成。

余承諺 僅誌於

台北科技大學資訊工程系碩士班

中華民國 114 年 7 月 23 日

TAIPEI TECH

Table of Contents

TAIPEI TECH

TAIPEI TECH

List of Tables

TAIPEI TECH

List of Figures

TAIPEI TECH

Chapter 1 Introduction

In recent years, malware has emerged as a significant concern for digital security. With over 560,000 new instances detected daily and ransomware attacks hitting four companies per minute, the threat landscape is becoming increasingly complex. Trojans, comprising 58% of malicious activities, contribute significantly to this threat. Despite ongoing efforts to address vulnerabilities, they persist, as evidenced by Google's weekly identification of 50 malware-infested sites. These trends indicate an urgent need for vigilance, robust security measures, and best practices to mitigate these threats. According to Forbes predictions, the costs associated with ransomware are expected to rise substantially, potentially reaching up to $265 billion by 2031 [1].

Moreover, a recent report from Kaspersky's security bulletin indicates that in 2024, there was a significant rise in mobile financial cyber threats, with the number of affected users increasing by 102% compared to 2023. Additionally, the report highlights the growing sophistication of mobile ransomware, predicting that future attacks will not only encrypt data on mobile devices but also manipulate or introduce fraudulent information, posing severe risks to financial transactions and personal data integrity. Furthermore, the report notes an increase in mobile-focused credential theft, with widely used information-stealing tools such as Lumma, Vidar, and Redline continuously evolving to target mobile platforms more effectively [2].

This research aims to develop a practical software detection tool, named LiSD (Light Weight Software Detection), to address the increasing security threats faced by mobile applications on the Android platform. To this end, we employ static application security testing (SAST) [3] as LiSD's core detection method, in conjunction with lightweight techniques. LiSD is characterized by lightweight and flexible, enabling its efficient operation in detecting Android vulnerabilities. We will validate the usability and effectiveness of LiSD by analyzing application APKs from Google Play [4], selected based on their popularity, and source code from Github [5],

TAIPEI TECH

chosen based on high reputation. The performance will be evaluated by measuring its analysis time and memory usage.

The rest of chapters architecture of this thesis are organized as follows: Chapter 2, reviews the literature and technologies related to this study; Chapter 3, introduces the platform's overall architecture and workflow, explaining the functionality and operation of each module; Chapter 4, details the setup of the development environment, tool usage, and technical aspects of functionality realization; Chapter 5, presents the experimental results, analyzes system performance, and evaluates the achievement of the research objectives; Chapter 6, summarizes the findings and contributions, and discusses limitations as well as potential future improvements and applications.

TAIPEI TECH

Chapter 2 Related Work

In this chapter, we will explore the techniques related to static analysis of Android, focusing on four main areas. First, in the Android Security section, we will discuss the security challenges of the Android platform, analyzing its inherent risks and protection methods. Second, the Detection section focuses on code-level detection techniques for identifying security vulnerabilities in applications. Next, the Tools section introduces commonly used static analysis tools that play an important role in transforming and inspecting code. Finally, the Static Application Security Test section provides an overview of other static application security testing techniques commonly used in research to understand how to effectively identify potential risks in applications.

2.1 Android Security

This section focuses on the security challenges of the Android system itself, including issues such as application privilege management and data protection, and explores possible solutions to improve the security of the platform as a whole.

In 2024, OWASP's top 10 list of security risks for mobile applications M. S. Thakur et al. [6] include the addition of security challenges such as "Improper Credential Usage", "Inadequate Supply Chain Security", "Insufficient Input/Output Validation", and "Inadequate Privacy Controls", which did not receive enough attention in the 2016 release. In addition, the increased rankings of "Insecure Authentication/Authorization", "Insufficient Binary Protections" and "Security Misconfiguration" indicate that security issues in these areas are becoming more pressing. Notably, while certain risks were not included in the 2024 top 10, they are likely to be important to consider in the future, including data leakage, hardcoded secrets, insecure access control, path overwrite and path traversal, and unprotected endpoints (e.g., Deeplink, Activity, Service, etc.). In addition, unsafe sharing behavior is also considered a potential threat. These potential risks reveal more challenges that mobile applications may face in terms of security protection, which may become emerging key threats in the future. Therefore, in-depth research

TAIPEI TECH

and evaluation of these risks will help to enhance the security of mobile applications and adapt to the changing threat environment.

2.2 Detection

Detection focuses on finding security problems in the application code. This approach examines the code structure of an application and identifies potential vulnerabilities such as insecure code implementations and improper privilege configurations.

X. Xu et al. [7] conduct an extensive investigation into the success rate of decompiling Android applications, exploring the influence of obfuscation and evaluating the efficacy of various decompilation tools. This research proposes a novel technique that leverages the strengths of generative models while mitigating model biases to recover meaningful variable names from decompiled code of stripped binaries. We build a prototype, GENNM, from pre-trained generative models CodeGemma-2B, CodeLlama-7B, and CodeLlama-34B. GENNM is fine-tuned on decompiled functions and taught to leverage contextual information. When querying a function, GENNM includes names from callers and callees, providing rich contextual information within the model's input token limit. We mitigate model biases by aligning the model's output distribution with developers' symbol preferences. Our results show that GENNM improves the state-of-the-art name recovery accuracy by 5.6-11.4 percentage points on two popular datasets, and in the most challenging setting, GENNM improves the state-of-the-art by 32% (from 17.3% to 22.8%) where ground-truth variable names are unseen in the training data.

H. Rathore, S. Chari, N. Verma, S. K. Sahay, and M. Sewak [8] present a comprehensive survey of data mining-based Android malware detection techniques. The study discusses the evolution of Android malware, current detection methods, and provides an in-depth analysis of each phase in data mining-based malware detection, including data acquisition, preprocessing, feature extraction, learning algorithms, and evaluation. Notably, the survey covers novel unsupervised and reinforcement learning-based malware detection frameworks, addressing limitations in previous studies. Furthermore, the authors highlight challenges and open issues in the field, such as explainability and IoT solutions, offering insights for future research directions.

2.3 Tools

This section covers various tools for static analysis of Android applications and the importance of Kotlin language support in modern Android application security. Tools for static analysis are primarily to help analyze application code and transform it into an easy-to-understand format to make code inspection and vulnerability discovery more efficient. For the language support, we will explore the comparative analysis of Kotlin and Python for security scanning applications.

N. Mauthe, U. Kargén, and N. Shahmehri [9] conducted a large-scale study of Android app decompilation success rate. They evaluated four different decompilers on three datasets consisting of 3,018 open-source apps from the F-Droid repository, 13,601 apps from a recent crawl of Google Play, and a collection of 24,553 malware samples. The results show that the jadx decompiler achieves an impressively low failure rate of only 0.02% failed methods per app on average, but only manages to recover source code for all methods in 21% of the Google Play apps. The study also found that code obfuscation is quite rarely encountered, even in malicious apps, and that decompilation failures mostly appear to be caused by technical limitations in decompilers, rather than by deliberate attempts to thwart source-code recovery by obfuscation. This study contributes to a better understanding of the practical applicability of Android decompilation and provides directions for future improvements of decompilation tools.

J.-M. Mineau and J.-F. Lalande [10] proposed that reproducibility and reusability are crucial for research in mobile platform security. However, the evolution of Android applications poses challenges to the reusability of static analysis tools. They reviewed static analysis research from 2011 to 2017 and evaluated the reusability of 22 tools on a dataset of 62,525 Android applications. The study found that 54.5% of the tools were no longer usable, while the success rate of executable tools was only 54.9%. Key factors affecting tool reusability include the size

TAIPEI TECH

of the application bytecode and the minimum SDK version. Additionally, malware samples had a higher completion rate compared to benign applications. The study highlights the difficulty of existing tools in handling modern Android applications and suggests future work to further analyze tool error types, particularly for Java-based tools, and incorporate more recent static analysis tools into research.

Kotlin for Modern Android Development:

- **Static Typing & Compile-Time Checks**: Kotlin is statically typed, so many bugs can be caught at compile time, increasing reliability.
- **Seamless Java Ecosystem Integration**: Kotlin can directly use Java libraries (e.g., OWASP tools), which means it benefits from a mature security ecosystem.
- **Faster Runtime Performance**: As a compiled language, Kotlin often runs faster than Python, which is interpreted.
- **Enterprise-Grade Development**: Kotlin is well-suited for large, maintainable codebases, especially in enterprise environments.

Table 1 provides a more detailed comparison of Kotlin and Python across several key development metrics:

Table 1: Comparative Analysis of Kotlin and Python Features

Feature	Kotlin	Python
Development Speed	Medium	Fast
Runtime Performance	High	Moderate
Ecosystem for Security Tools	Moderate (via Java)	Very rich and mature
Community & Resources	Growing	Huge
Ideal for Rapid Prototyping	Not ideal	Excellent
Code Maintainability	High (static typing, IDE support)	Medium (dynamic, needs testing)

2.4 Static Application Security Testing

Static application security testing is an analysis technique that does not require the execution of an application, and identifies potential security risks by examining the code. This technique is designed to detect problems early and help developers reduce the security risk of applications during the development process.

N. Abolhassani and W. G. Halfond [11] present a multi-level, component-sensitive static analysis technique designed to more effectively and accurately compute Intent information within Android application packages. Existing static analysis techniques, such as IC3 and ICCBot, have limitations in their accuracy when dealing with Intents constructed from received Inter-Component Communication (ICC) information. To address this issue, this method introduces placeholders in the intent model and resolves these placeholders in subsequent analysis phases, thereby improving analysis accuracy and efficiency. Experimental results show that, compared to existing techniques, this method exhibits higher precision and recall in the analysis of Intents and ICC links, while maintaining a competitive execution time.

Rahul [12] explained that static code analysis is a technique for predicting program output without actually running the program. Static code analysis typically refers to program understanding, which involves understanding the program and detecting problems within it, including syntax errors, type mismatches, performance bottlenecks, potential errors, and security vulnerabilities. The main steps in static analysis include Scanning, which breaks down the code into smaller lexical units (tokens); tokens are similar to words in a language, and can be individual characters, literals (such as integers, strings), or reserved keywords of the language; Parsing, where a parser takes the tokens, checks that their order of appearance conforms to the grammar, and organizes them into a tree structure called an Abstract Syntax Tree (AST). The AST abstracts away low-level details, focusing only on the logical structure of the program, which is ideal for static analysis; and Analyzing ASTs, where Python provides the ast module to simplify AST analysis. Analyzing an AST typically requires an AST "walker" to traverse the tree. The walker implements specific "visitor" methods to analyze node types of interest.

TAIPEI TECH

J. Zhu, K. Li, S. Chen, L. Fan, J. Wang, and X. Xie [13] introduced VulsTotal, the first unified evaluation platform for Android static application security testing (SAST) tools, to address the challenges of evaluating their performance. Their study systematically selected 11 open source SAST tools and introduced a standardized taxonomy of 67 common vulnerability types. To ensure consistency in evaluation, they restructured vulnerability reports and developed a CVE-based benchmark by analyzing Android-specific CVEs. Their comprehensive analysis revealed that no tool fully covered all vulnerability types, with the highest coverage reaching only 67%. They also found a significant gap between real-world vulnerabilities and synthetic benchmarks, highlighting the need for improved detection capabilities. Their findings provide valuable insights into the strengths and limitations of existing SAST tools, guiding future improvements in Android application security testing.

TAIPEI TECH

Chapter 3 Architecture

This chapter provides a detailed overview of the architecture and testing process of LiSD. First, we will discuss the overall design architecture, including the functionalities and interrelationships of each module. Then, we will delve into the analysis of the testing process, explaining the execution sequence and logic of each step in detail.

3.1 Architecture of LiSD (Light Weight Software Detection)

LiSD is divided into five modules: the processing unit, logic analysis, code analysis, management module, and user interface. (Figure 1)

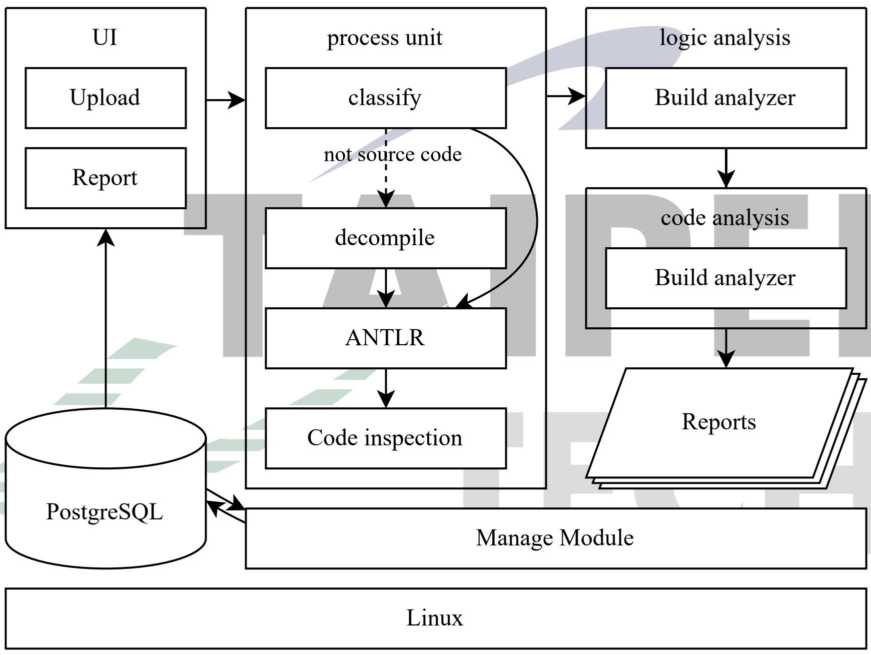


Figure 1: LiSD Architecture

1. Processing Unit

   The processing unit preprocesses and inspects the code to identify vulnerabilities using functions for classification, decompilation, ANTLR [14] (third-party feartures), and code inspection. Details as follows:

   (1) Classification:

In this step, the processing unit will classify whether the input file is already compiled or not, if the file is a source file, then it will send the file to step (3), otherwise it will send it to the next step.

An APK file is a compressed package for Android applications, containing directories like META-INF, lib, res, and assets, as well as files like AndroidManifest.xml, classes.dex, and resources.arsc [15]. This structure can be used to identify whether a file is an APK.

An APK file is a compressed package format used for Android applications. It contains various directories and files, each serving a specific purpose Table 2:

Table 2: APK File Content

Name	Description
META-INF	Contains the manifest file and signature-related information, ensuring the integrity and authenticity of the APK file.
lib[1]	Stores compiled native libraries for different processor architectures, such as ARM or x86.
res[1]	Includes uncompiled resources like layouts, images, and raw files that the application uses.
assets[1]	Contains raw files bundled with the application, accessible via the AssetManager API at runtime.
AndroidManifest.xml	A critical XML file describing the application's configuration, permissions, activities, services, and other components.
classes.dex classesN.dex[1]	Houses the compiled Dalvik bytecode that the Android runtime executes.
resources.arsc	Compiles pre-processed resources, including string values and references used by the application.

[1] It may not exist.

This structure makes it possible to identify whether a given file is an APK.

(2) Decompilation:

In this step, the processing unit will decompile the file to source code that input from step (1), then send the decompiled file to the next step.

(3) ANTLR:

In this step, the processing unit will use ANTLR to split the whole project into a tree format then send to the next step.

(4) Code inspection:

In this step, the processing unit rearranges the tree input from step (3) and modifies its format to make later processing easier.

2. Logic Analysis

In logic analysis will analyze logic problem detected by code inspection.

Common logic problem includes:

(1) Null Pointer Exception(NPE):

For example:

```
String name = null;
System.out.println(name.length());
```

In this case “name.length()” will throw NPE because name is null, this problem might happen on variable not initialized normally or wrong setting variables, mostly happen on developer didn’t check the value is null on nullable variables.

(2) Incorrect Conditional Logic:

For example:

```
int age = 18;
if (age >= 18) {
  System.out.println("You are an adult");
}
else {
  System.out.println("You are not adult");
}
```

In this case “`age >= 18`” will always be true, everything in else block will not be reached, if this problem happen in while condition, it can cause infinite loop. This problem usually happen on developer update wrong value on variables or the variables not updated in the loop.

3. Code Analysis

Code analysis analyzes the bad coding style detected by code inspection.

In software development, the quality of the program code not only affects the readability and maintainability of the program, but is also directly related to the development efficiency and stability of the final product.

Menshawy, Rana S. et al. [16] identified common code smells and their classifications. Below are examples of code smells that may cause issues:

(1) Overly Long Methods and Classes

If a method or class contains too much logic, it reduces readability and makes maintenance difficult. For example, a class that handles user registration, validation, emailing, and logging should be broken into smaller functional units.

(2) Hard-Coded Values

It is a common mistake to hard code sensitive information into a program. For example, database passwords are written directly into the code. If this code is made public, an attacker can easily access this sensitive information and launch an attack on the system, especially without knowing it, which can seriously compromise the security of the system.

(3) Redundant and Duplicate Code

The same or similar logic is repeated in multiple places, for example, the same validation logic appears in multiple modules. This code should be extracted into common methods. If the logic needs to be changed, the duplicated code increases the amount of work required to change it, and it is also easy to miss some of the updates, resulting in inconsistent behavior.

(4) Poor Error Handling

If a program lacks proper error handling, anomalies may not be handled properly and may even reveal internal system information. For example, catching an anomaly and displaying the detailed error message directly to the user may allow an attacker to gain access to the internal details of the system and then exploit the vulnerability.

(5) Unused Code

Keeping unused variables or methods in a program can make it more confusing. For example, if an old version of validation logic is replaced with new logic, but the developer does not remove the useless code, this not only adds complexity to the program, but it can also lead other developers to believe that the logic is still in use and to maintain it incorrectly, increasing the risk of vulnerabilities.

4. Management Module

The management module manages the status of all modules and handles the communication between each module.

5. User Interface

In the user interface, the user can upload the file and view the report.

3.2 Process of testing

Firstly, the overall structure of the test is illustrated in Figure 2.

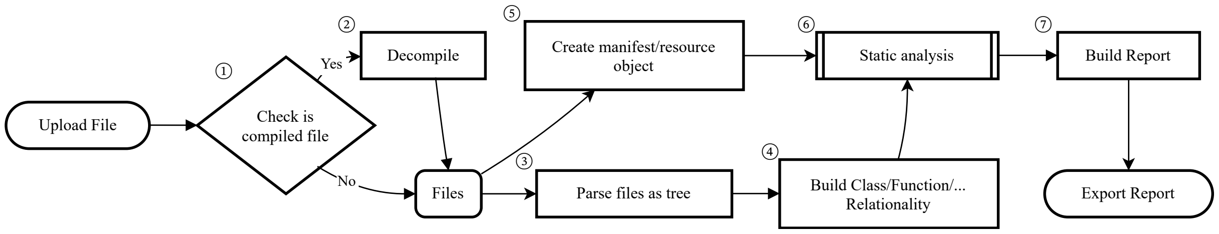


Figure 2: Process of Testing

The preceding diagram enables us to summarize the steps as follows:

1. **Verification of its file format**

TAIPEI TECH

Once the file has been uploaded, our system will determine whether it is a compiled file based on the file extension. If the file is already compiled, proceed directly to step 2. If the file is source code, go to steps 3 or 5.

2. **Decompile**

   Decompile the file, then, go to steps 3 or 5.

3. **Parse file as tree**

   For later analysis, we will use ANTLR (ANother Tool for Language Recognition) [14] to parse the file into a tree structure.

   For example, the code "`int num = 10`" is tokenized to five tokens, and these tokens can be parsed as a syntax tree according to the rules shown in Figure 3.

   The rules are defined in a file with the extension g4, which allows you to tell ANTLR how to tokenize content and parse those tokens. In this rules file, you can specify keywords such as "`int`" or patterns such as "`[0-9]+`" to represent numbers. Unlike regular expressions, ANTLR allows you to assign names to each rule, so rules can be reused throughout the grammar. When ANTLR encounters a rule in the provided content, it records the matched content, notes which rule it matches, and identifies the location of the match within the original input. This feature increases the flexibility and readability of your grammar, making it easier to manage complex parsing tasks.

TAIPEI TECH

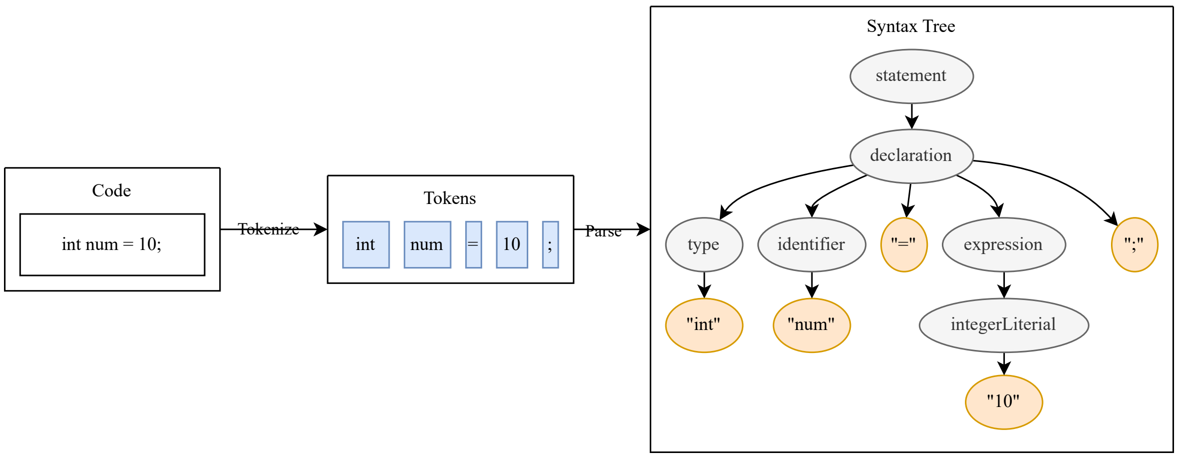


Figure 3: ANTLR Converts Code into a Syntax Tree.

Using ANTLR to parse files allows for more precise analysis because it handles syntax like a compiler.

For example, using regular expression to find the instantiation that instantiates `Intent`, “`new Intent()`”, we can use “`new\s+Intent\(.*\)`”, but this will also find the code like “`Log.i("new Intent() on this line")`”. If we use ANTLR, we can check the statement more closely to prevent misuse.

4. **Reclass the content**

   Based on the results of the above analysis, we will reconstruct the classes according to the relationships between their functions and other classes. For example, if a class uses another class, we will document which classes and methods it depends on. Once all classes have been rebuilt, we will compile a comprehensive list of all classes and methods, while also identifying which classes serve as libraries. After completing this step, we will continue to step 6.

5. **Re-manifest file to index**

   Build the manifest object to index file according to the permissions and activities defined in manifest file. In `AndroidManifest.xml`, it records what permissions will be used and what features must be provided by device, also, four important components: Activity, Service, BroadcastReceiver and ContentProvider, are defined in this file (Figure 4). After completing this step, we will continue to step 6.

TAIPEI TECH

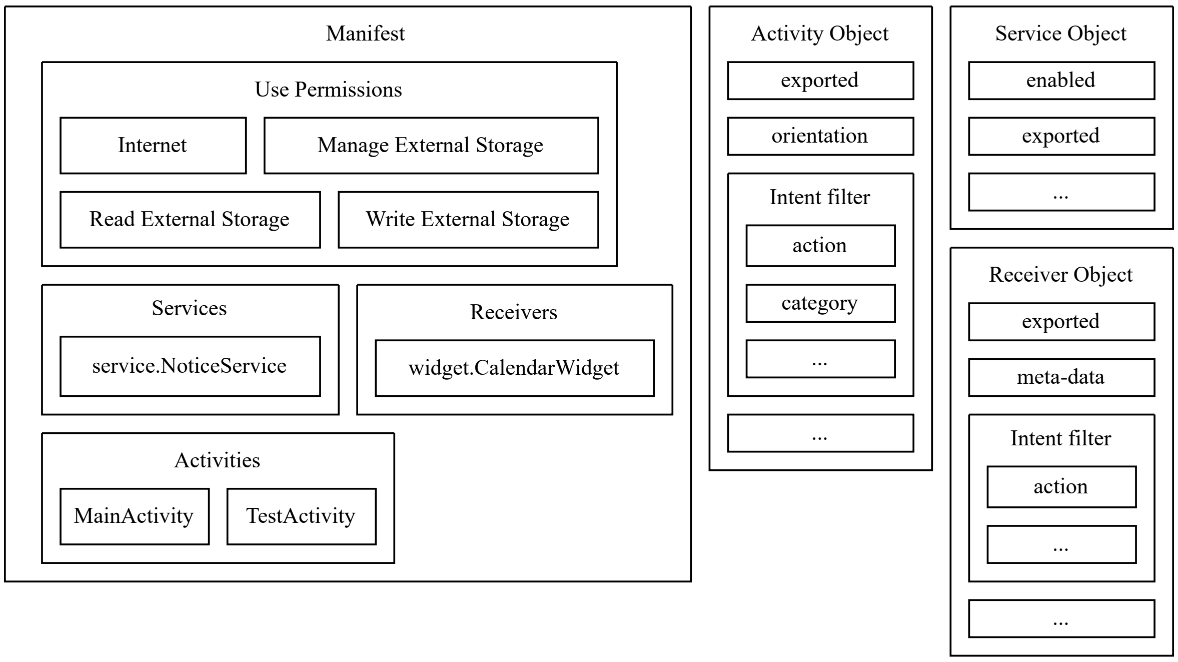


Figure 4: Manifest Object Example

6. **Perform static analysis (Figure 5)**

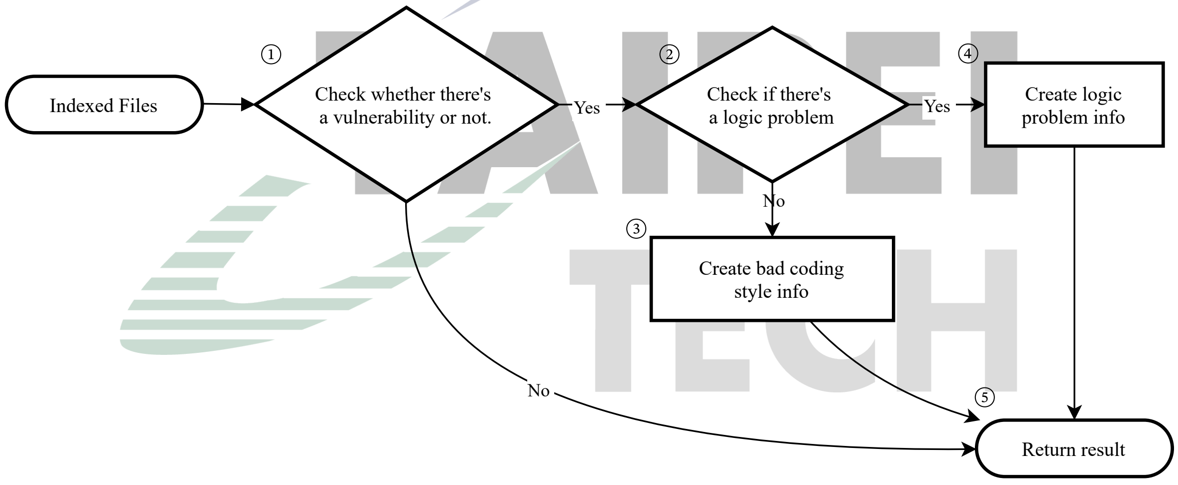

Indexed Files
①
Check whether there's a vulnerability or not.
Yes
②
Check if there's a logic problem
Yes
④
Create logic problem info
No
③
Create bad coding style info
No
⑤
Return result

Figure 5: Static Analysis

(1) The first thing to do is to determine if there is a weak point, if not, go to step v, otherwise continue.

We will provide the scripts and the ASTs to be scanned for vulnerabilities. Then, the script will scan the ASTs to identify patterns.

(2) Determine if there is a logic problem, if so, go to step (4), otherwise continue.

(3) Create bad coding style info, go to step 7.

(4) Create logic problem info, then continue.

(5) Go to step 7.

7. **Generate report**

The report includes a detailed description of the location of the problem, including the file path, row number, and column number. It also provides a comprehensive description of the problem, accompanied by a detailed assessment of its severity, type, code snippet, cause, and recommendation.

TAIPEI TECH

Chapter 4 Implementation

In this chapter, we introduce the environment and explain how to set up the necessary components for creating LiSD. We begin by detailing the required hardware specifications, essential tools, compatible software versions, and configuration settings. Next, we will guide you through a step-by-step process to establish the environment. Finally, we will provide a comprehensive explanation of how to implement the LiSD.

4.1 Establishment of the environment

This section details the implementation process. Firstly, we will introduce the hardware and software requirements necessary for this study to ensure the system can run smoothly. Subsequently, we will explain the third-party software and open-source resources used, and clarify their roles within the system. Finally, we will provide a step-by-step guide on how to set up the basic environment, laying the groundwork for subsequent development and testing.

TAIPEI TECH

4.1.1 Hardware requirements

Table 3 presents the hardware specifications used in this study's implementation, including the processor, memory, and storage. The processor, Intel Core i7-10700, was selected to ensure performance efficiency. The system is equipped with 48 GB of RAM to handle large datasets and support the smooth operation of the development environment. Additionally, a 1 TB disk provides ample storage capacity for data and applications.

Table 3: Hardware Specification

Name	**Description**
CPU	Intel Core i7-10700
RAM	48 GB
Disk	1 TB

4.1.2 Third party/Open Source

To implement LiSD, we will use the Ktor framework, a server framework for the Kotlin programming language, along with PostgreSQL as a database to store detection scripts and reports. We will also use HTML-DSL and Content-Negotiation Plugin in Ktor to provide both a browser interface and API connectivity. The project will be compiled using JDK 21 and packaged as a Docker image to ensure that LiSD can run on different operating systems (Table 4).

Table 4: Third Party Software

Name	Version	Licensing	Description
ANTLR [14]	4.13.2	BSD-3-Clause license	A powerful parser generator for reading, processing, executing, or translating structured text or binary files.
Ktor [17]	3.0.1	Apache License 2.0	A framework for building asynchronous server-side and client-side applications with ease.
PostgreSQL [18]	17.2	PostgreSQL Licence	PostgreSQL is a powerful, open source object-relational database system with over 35 years of active development that has earned it a strong reputation for reliability, feature robustness, and performance.
Exposed [19]	0.57.0	Apache License 2.0	Exposed is a lightweight SQL library on top of a

Name	Version	Licensing	Description
			JDBC driver for the Kotlin programming language.
H2 [20]	2.3.232	Mozilla Public License Version 2.0 Eclipse Public License	H2 is an embeddable RDBMS written in Java.
Jackson [21]	2.18.1	Apache License 2.0	Jackson has been known as “the Java JSON library” or “the best JSON parser for Java”. Or simply as “JSON for Java”.
Docker Engine [22]	4.37.0	Apache License 2.0	Docker Engine is an open source containerization technology for building and containerizing your applications.
Amazon Corretto [23]	21.0.7.6.1	GPL-2.0 license	Amazon Corretto is a no-cost, multiplatform, production-ready distribution of the Open Java Development Kit (OpenJDK).
Kotlin Script [24]	2.1.0	Apache License 2.0	Kotlin scripting is the technology that enables executing Kotlin code as scripts without prior compilation or packaging into executables.
JADX [25]	1.5.1	Apache License 2.0	Dex to Java decompiler

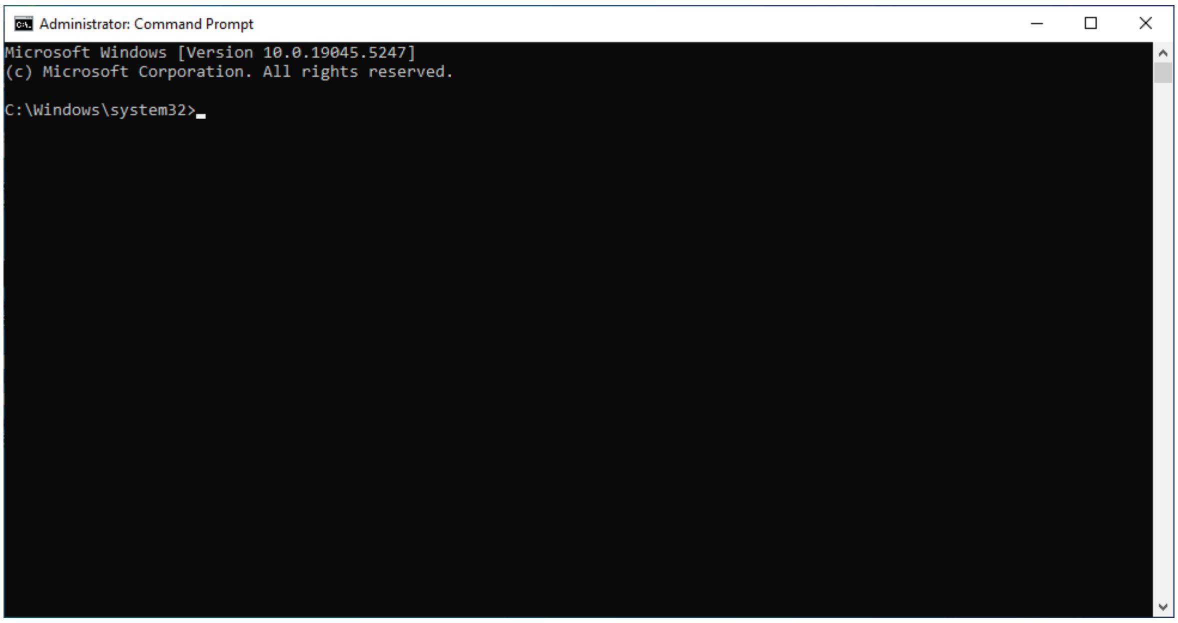
4.1.3 Environment Setup

To establish the development environment, IntelliJ IDEA, Java Development Kit, Ktor, and Docker were included through the following steps:

4.1.3.1 Install Git to download LiSD

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency [26]. We will use Git to download LiSD.

1. Open terminal as administrator (Figure 6)

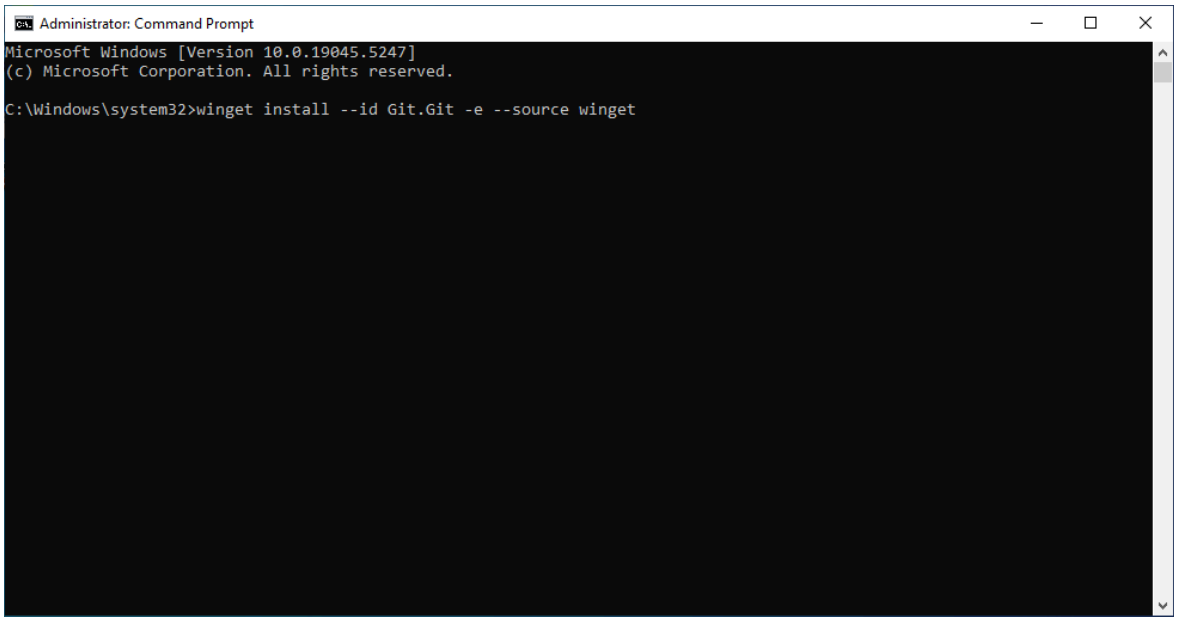


```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.19045.5247]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```

Figure 6: Open Terminal as Administrator

2. Run the following command (Figure 7)

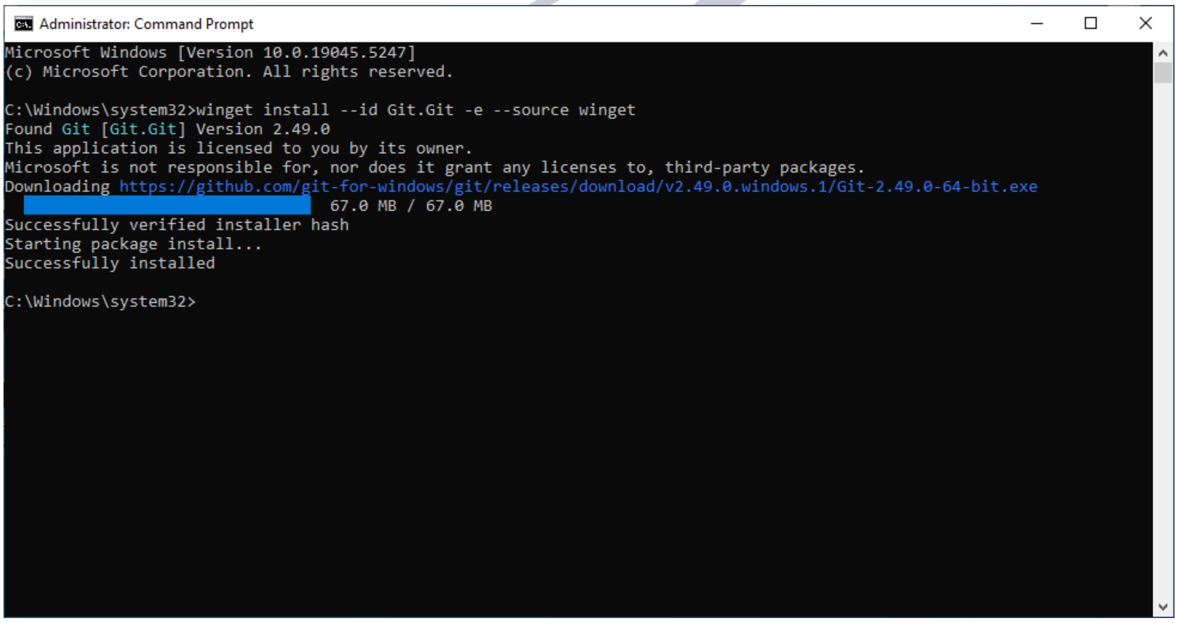
```
winget install --id Git.Git -e --source winget
```


Administrator: Command Prompt

```
Microsoft Windows [Version 10.0.19045.5247]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>winget install --id Git.Git -e --source winget
```

Figure 7: Run Git Installation Command

3. Waiting for installing finished (Figure 8)


Administrator: Command Prompt

```
Microsoft Windows [Version 10.0.19045.5247]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>winget install --id Git.Git -e --source winget
Found Git [Git.Git] Version 2.49.0
This application is licensed to you by its owner.
Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.
Downloading https://github.com/git-for-windows/git/releases/download/v2.49.0.windows.1/Git-2.49.0-64-bit.exe
                                67.0 MB / 67.0 MB
Successfully verified installer hash
Starting package install...
Successfully installed

C:\Windows\system32>
```

Figure 8: Install Git Finished

4.1.3.2 Install Docker as the runtime environment for the software, supporting system deployment and testing

Docker is a containerization platform that simplifies application deployment by packaging software and its dependencies into lightweight, portable containers. To install Docker, visit the Docker website, download the appropriate version for your operating system, and follow the installation instructions. Once installed, Docker provides a consistent runtime environment that supports efficient system deployment, testing, and scalability across various environments.

1. Go to the Docker official website (Figure 9)

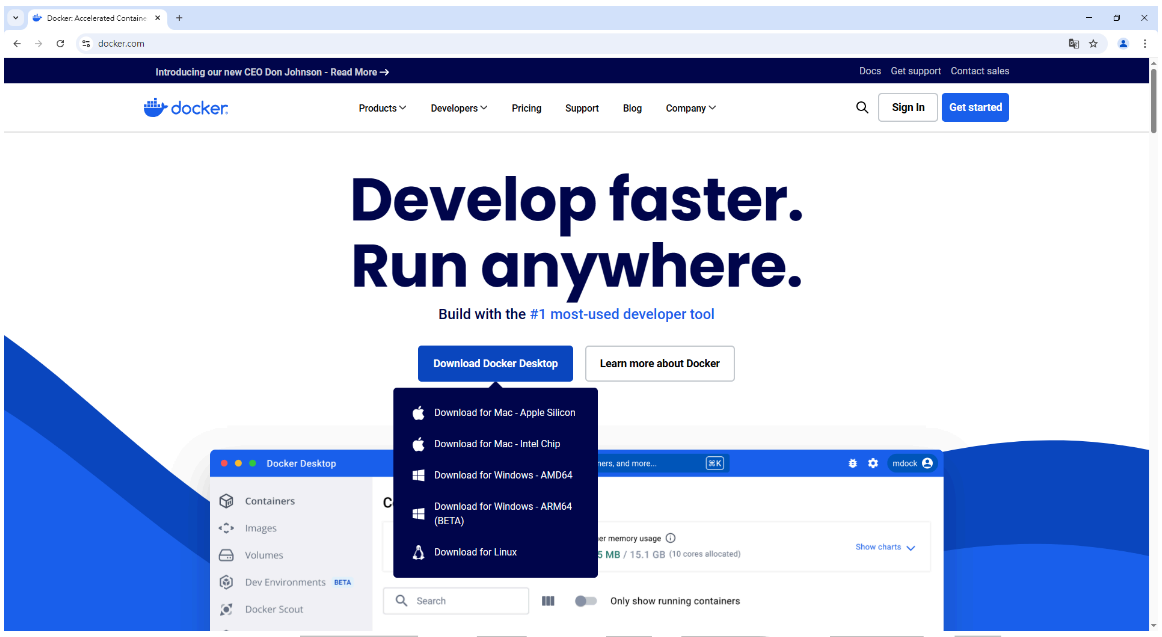


Figure 9: Docker Official Website

2. Download the Docker installer for Windows (Figure 10)

TAIPEI

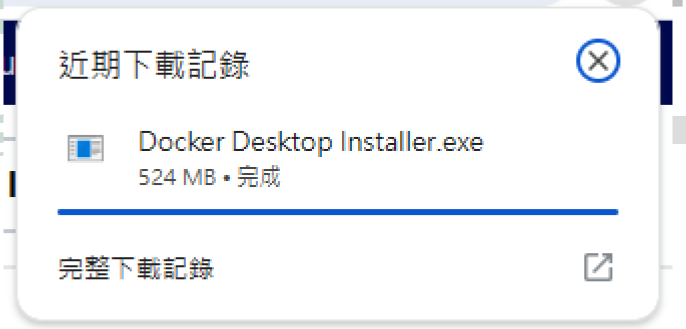
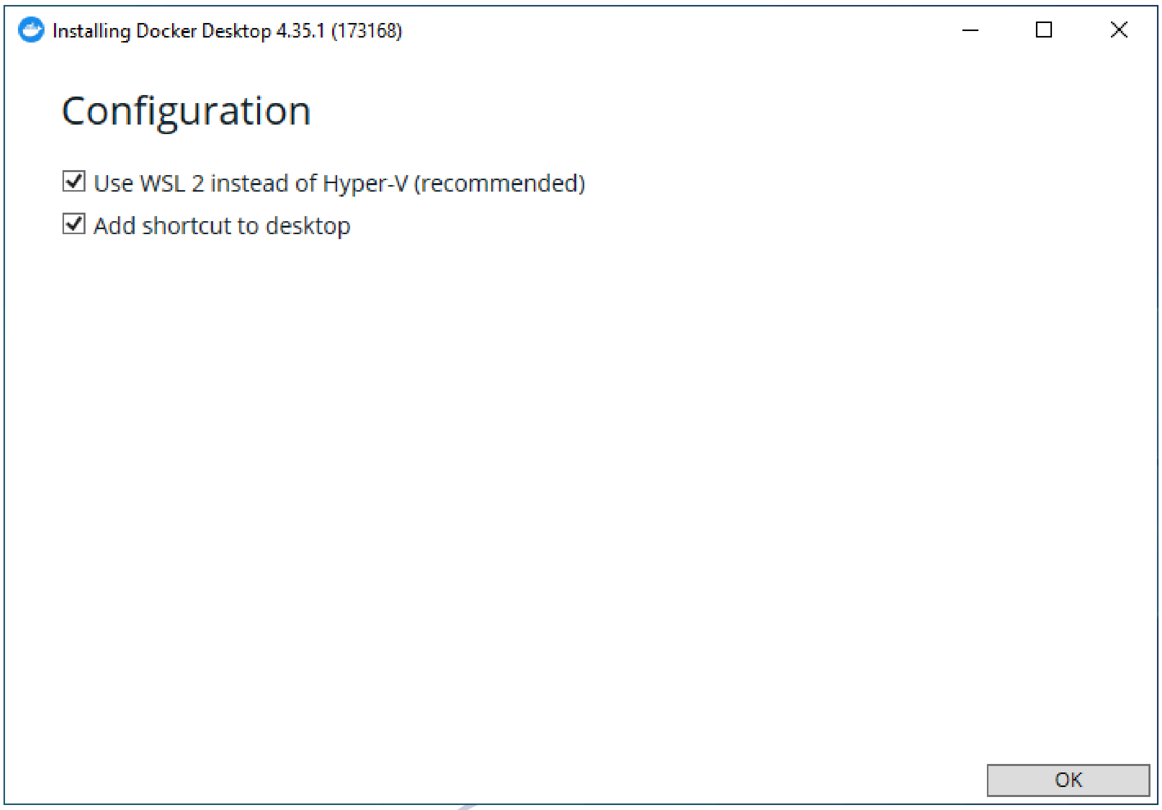


Figure 10: Download Docker Installer

3. Run the Docker installer and click OK (Figure 11)



Installing Docker Desktop 4.35.1 (173168)

Configuration

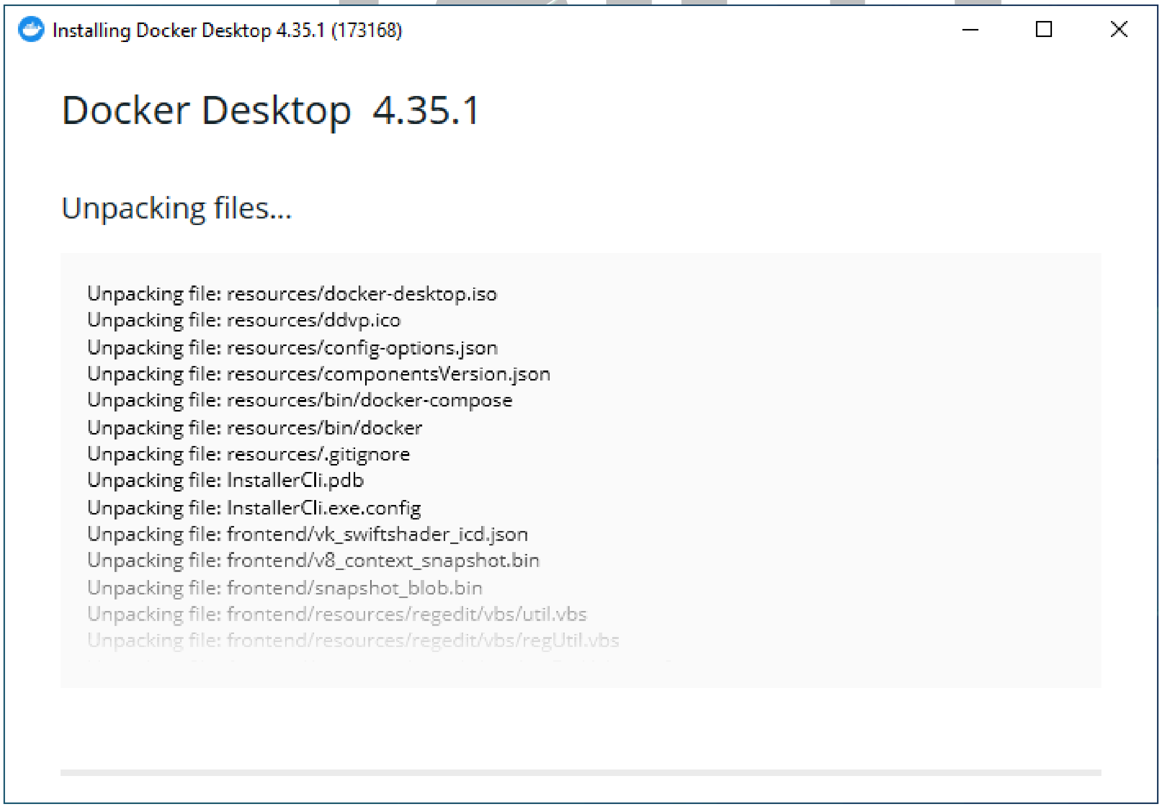
☑ Use WSL 2 instead of Hyper-V (recommended)

☑ Add shortcut to desktop

Figure 11: Docker Installer

TAIPEI

4. Wait for the installation progress to finish (Figure 12)



Installing Docker Desktop 4.35.1 (173168)

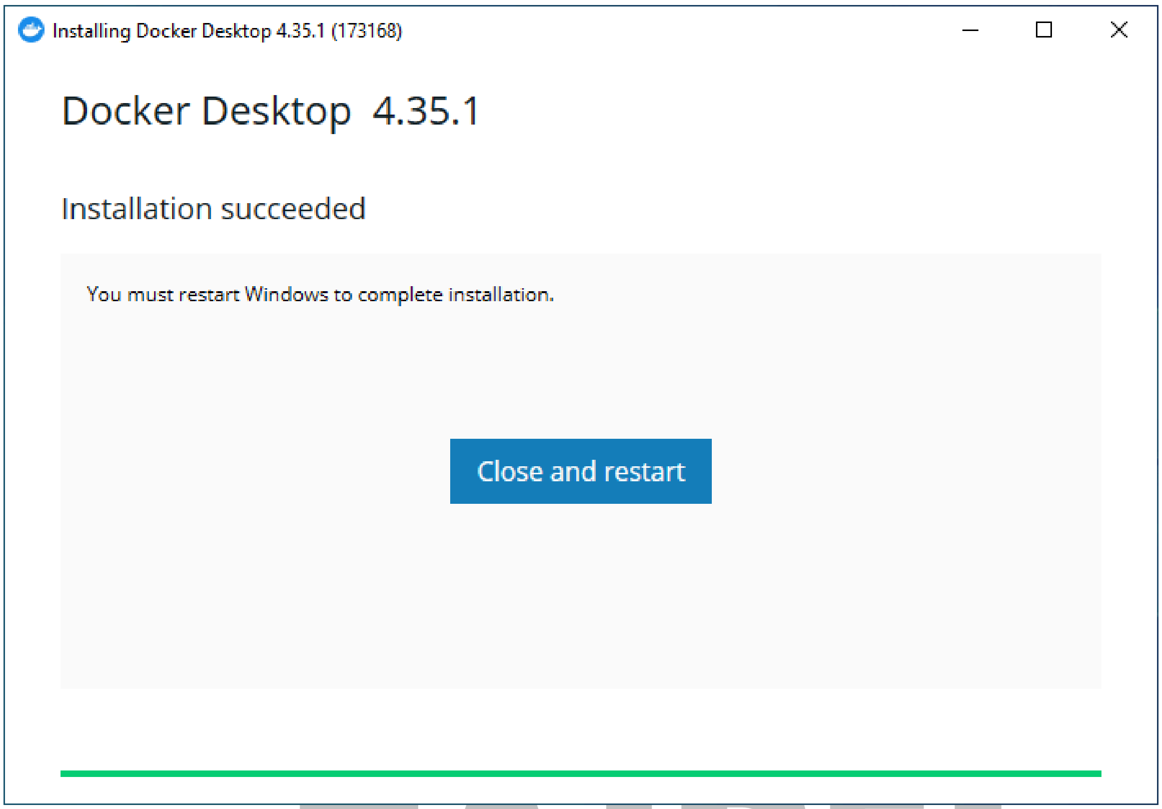
Docker Desktop 4.35.1

Unpacking files...

```
Unpacking file: resources/docker-desktop.iso
Unpacking file: resources/ddvp.ico
Unpacking file: resources/config-options.json
Unpacking file: resources/componentsVersion.json
Unpacking file: resources/bin/docker-compose
Unpacking file: resources/bin/docker
Unpacking file: resources/.gitignore
Unpacking file: InstallerCli.pdb
Unpacking file: InstallerCli.exe.config
Unpacking file: frontend/vk_swiftshader_icd.json
Unpacking file: frontend/v8_context_snapshot.bin
Unpacking file: frontend/snapshot_blob.bin
Unpacking file: frontend/resources/regedit/vbs/util.vbs
Unpacking file: frontend/resources/regedit/vbs/regUtil.vbs
```

Figure 12: Installing Docker

5. Click Close and restart when the installation is done (Figure 13)


Installing Docker Desktop 4.35.1 (173168)

Docker Desktop 4.35.1

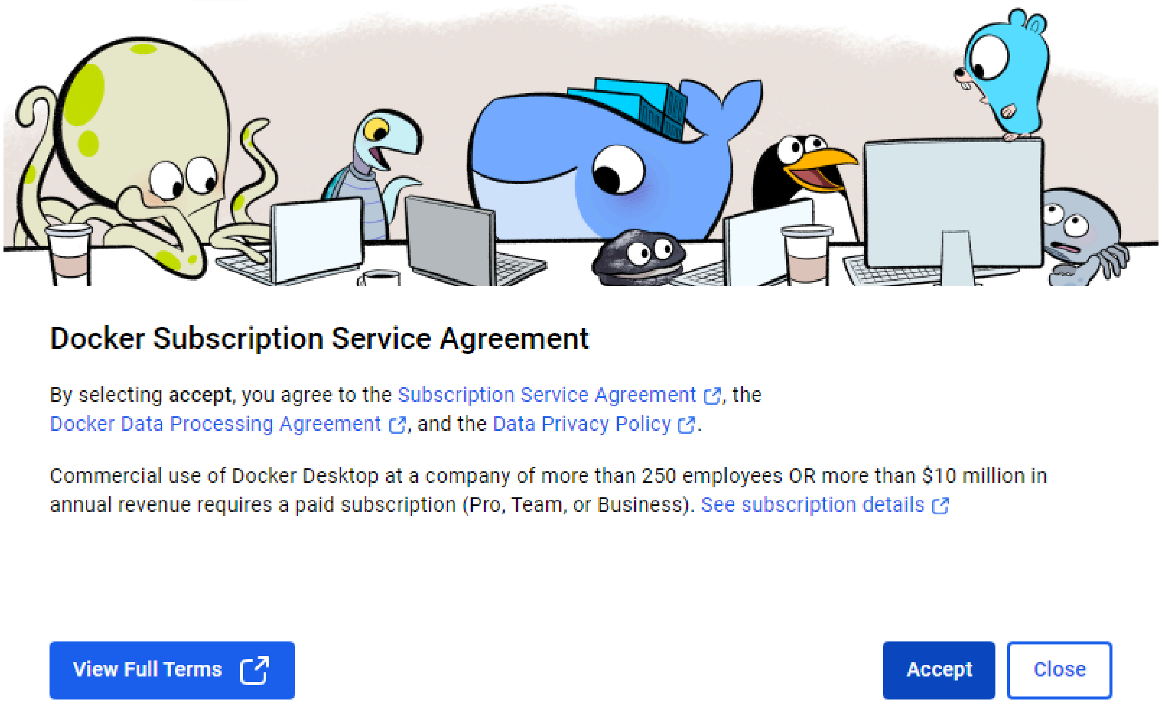
Installation succeeded

You must restart Windows to complete installation.

Figure 13: Installing Docker Finished

TAIPEI

6. Docker will start after Windows restart, we click Accept to accept the agreement (Figure 14)

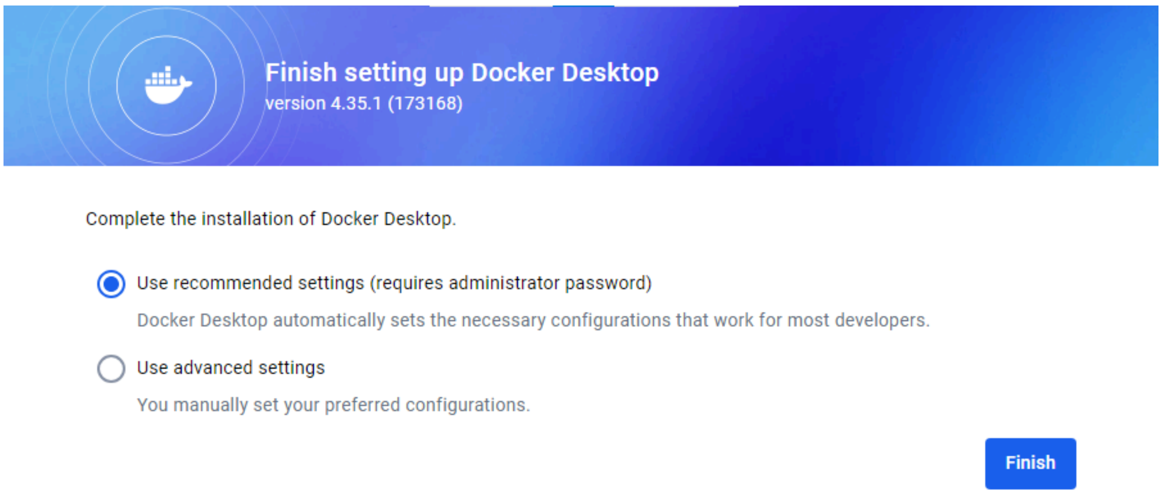

Docker Subscription Service Agreement

By selecting **accept**, you agree to the Subscription Service Agreement, the Docker Data Processing Agreement, and the Data Privacy Policy.

Commercial use of Docker Desktop at a company of more than 250 employees OR more than $10 million in annual revenue requires a paid subscription (Pro, Team, or Business). See subscription details

Figure 14: Accept Docker Agreement

7. Use recommended settings and click Finish (Figure 15)


Finish setting up Docker Desktop
version 4.35.1 (173168)

Complete the installation of Docker Desktop.

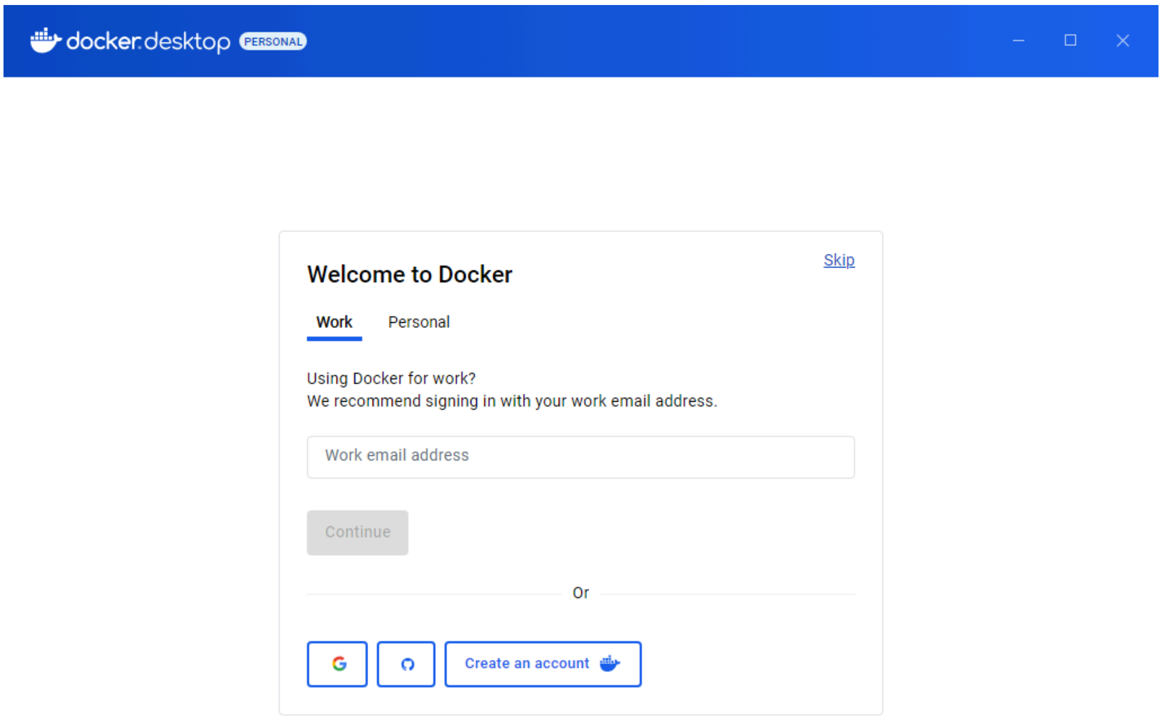
[✓ Use recommended settings (requires administrator password)]
Docker Desktop automatically sets the necessary configurations that work for most developers.

Use advanced settings
You manually set your preferred configurations.

Figure 15: Docker Startup Settings

8. Skip login to Docker (Figure 16)

TECH



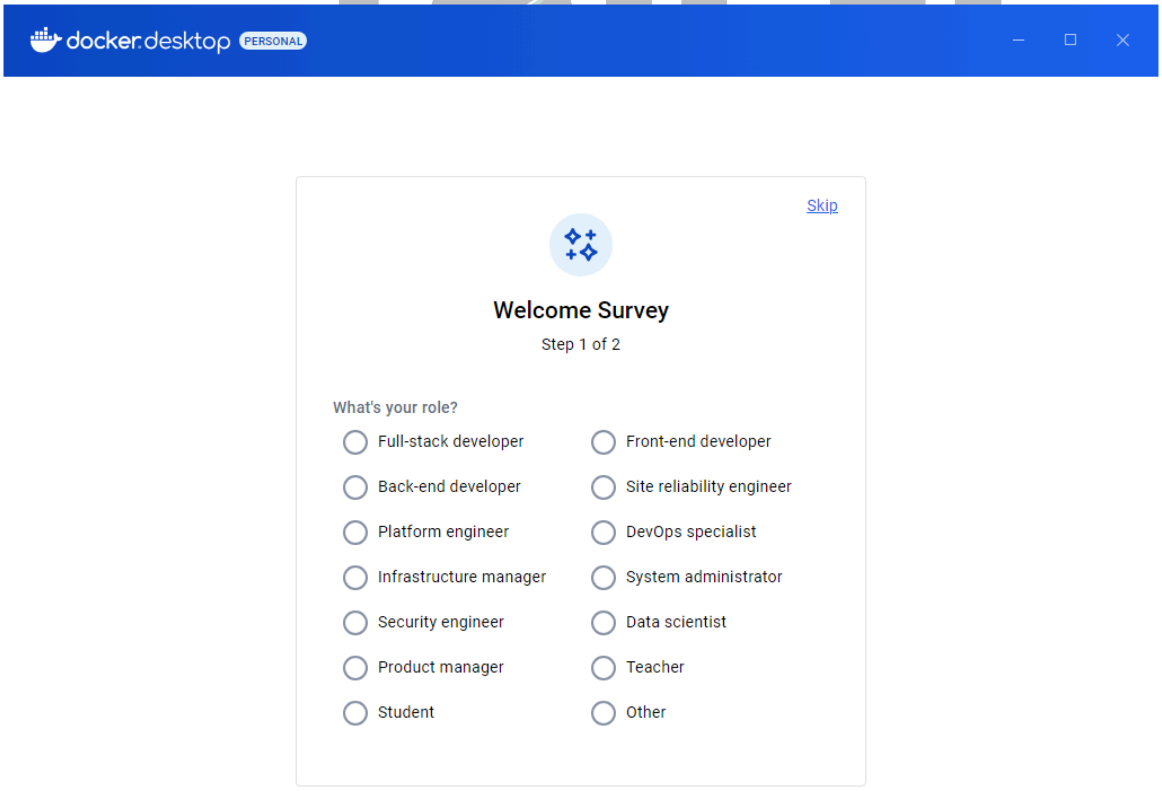
Welcome to Docker

Using Docker for work?
We recommend signing in with your work email address.

Figure 16: Docker Login

TAIPEI

9. Skip the Docker survey (Figure 17)



Welcome Survey

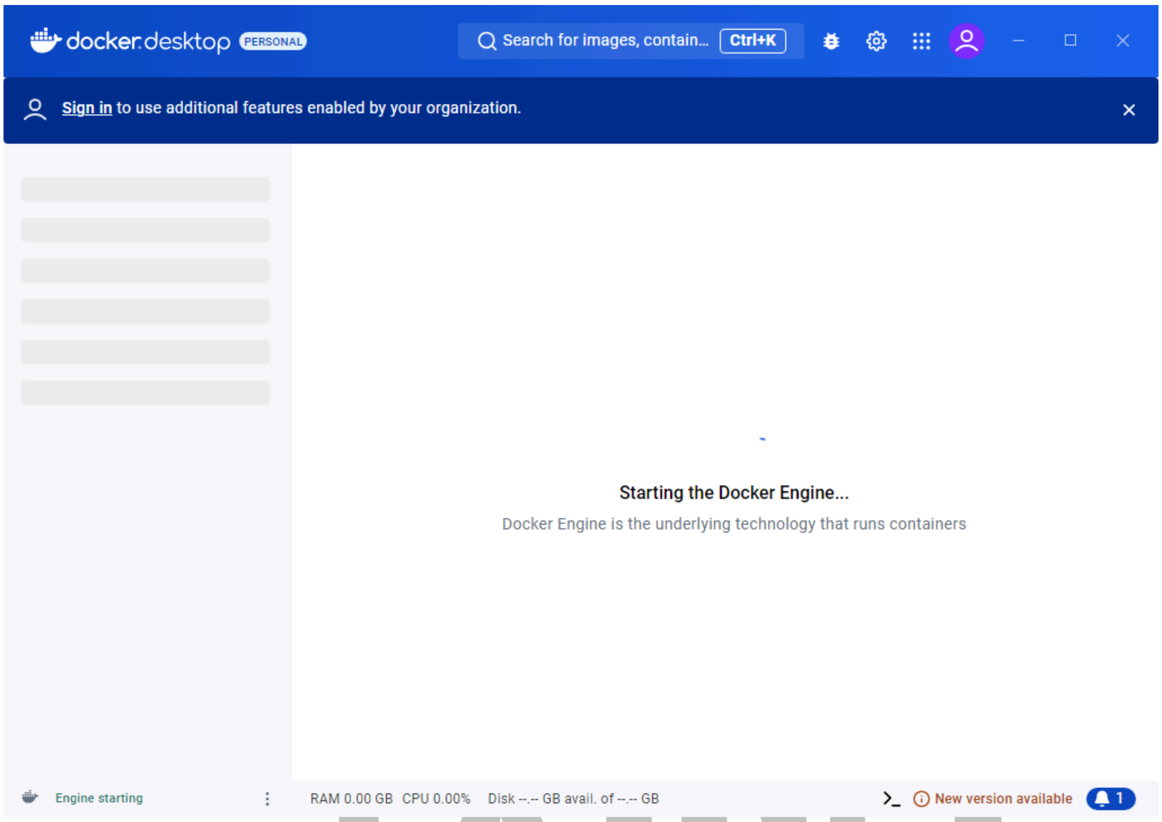
Step 1 of 2

What's your role?

○ Full-stack developer	○ Front-end developer
○ Back-end developer	○ Site reliability engineer
○ Platform engineer	○ DevOps specialist
○ Infrastructure manager	○ System administrator
○ Security engineer	○ Data scientist
○ Product manager	○ Teacher
○ Student	○ Other

Figure 17: Docker Survey

10. Wait for Docker engine to start (Figure 18)


docker desktop PERSONAL
Search for images, contain... Ctrl+K
Sign in to use additional features enabled by your organization.
Engine starting
RAM 0.00 GB CPU 0.00% Disk --.-- GB avail. of --.-- GB
New version available
1

Starting the Docker Engine...

Docker Engine is the underlying technology that runs containers

Figure 18: Docker Desktop

TAIPEI TECH

4.2 Implementation

In implementation chapter, we will introduce where the source code and APK file come from, and introduce detail steps and part of important code of Static Analysis

4.2.1 Data Collection

The test data for this study is sourced from two main categories: source code and APK files.

- Source Code:

  To obtain representative Android application source code, this study selects the top 5 open-source Android projects on GitHub [5] based on star count. As the world's largest open-source code hosting platform [27], GitHub's star count reflects a project's popularity and influence. Therefore, selecting the top 5 projects with the highest star count ensures that the test data is both representative and practical.

- APK files:

In order to include the mainstream Android applications, this study will use the 10 of the most installed applications in the Google Play Store [4] collected by Android Rank [28] as the test subject. Since Google Play Store is the official distribution platform for Android applications, the number of downloads indicates the popularity and widespread use of the applications. Selecting the most downloaded 10 apps helps to evaluate the effectiveness of the proposed static analysis method in real-world applications.

4.2.2 LiSD Establishment

LiSD can be establish by the following step:

1. Clone repository from Github

   Open a terminal and run the command:

   ```
   git clone https://github.com/littlexfish/LiSD.git
   ```

2. After Git clone finished, go to directory `LiSD` and run docker compose

   Open a terminal and execute the following command:

   ```
   docker compose -f docker-compose.yaml up
   ```

3. After the server is prepared, access the website

   If the server is prepared, it will show the log like `INFO Application - Responding at http://127.0.0.1:8080`. Then access the website url `https://localhost:8080` or `https://127.0.0.1:8080` and you will see the page like Figure 19.

TAIPEI TECH

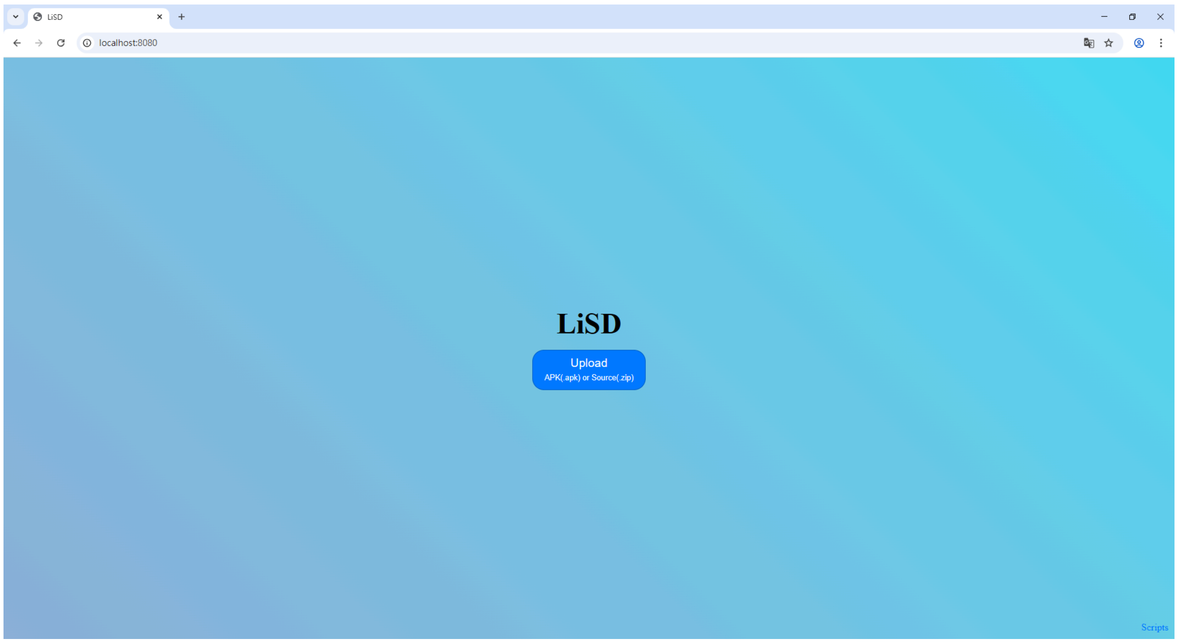


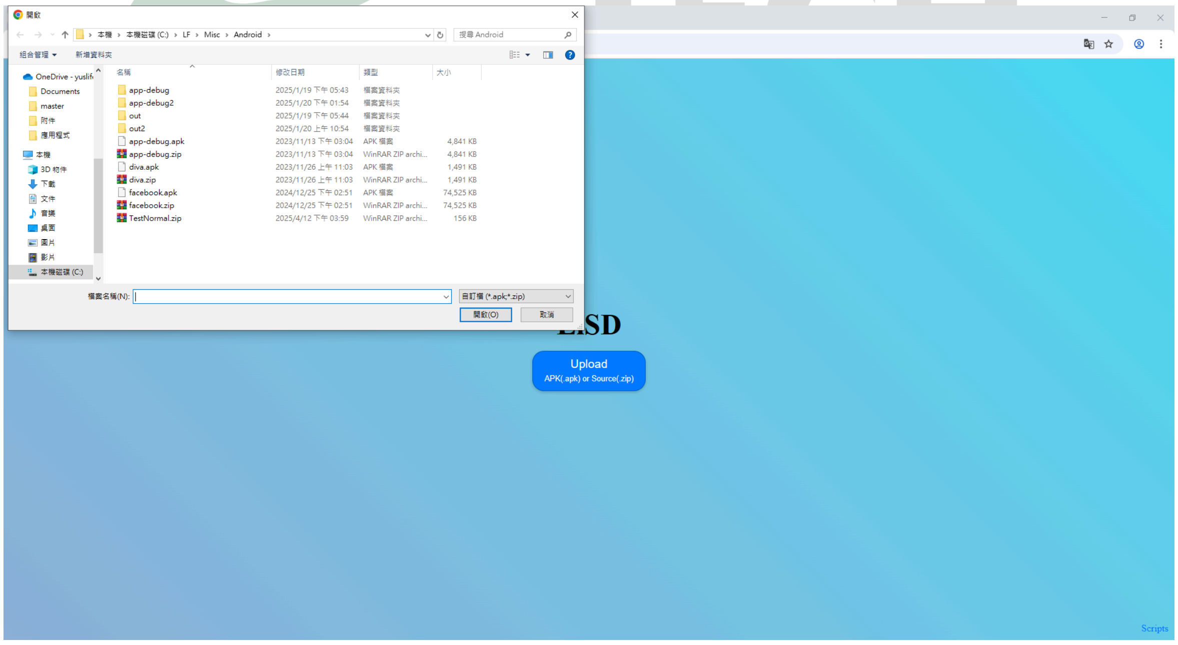
Figure 19: LiSD Home Page

4.2.3 Practical

The environment is set up as Section 4.1.3, we collect 10 of APK files and 5 of the source projects. Our analysis step is as follows:

1. Upload file (Figure 20).

   Use the Upload button to submit the source project as a zipped file or an APK file.



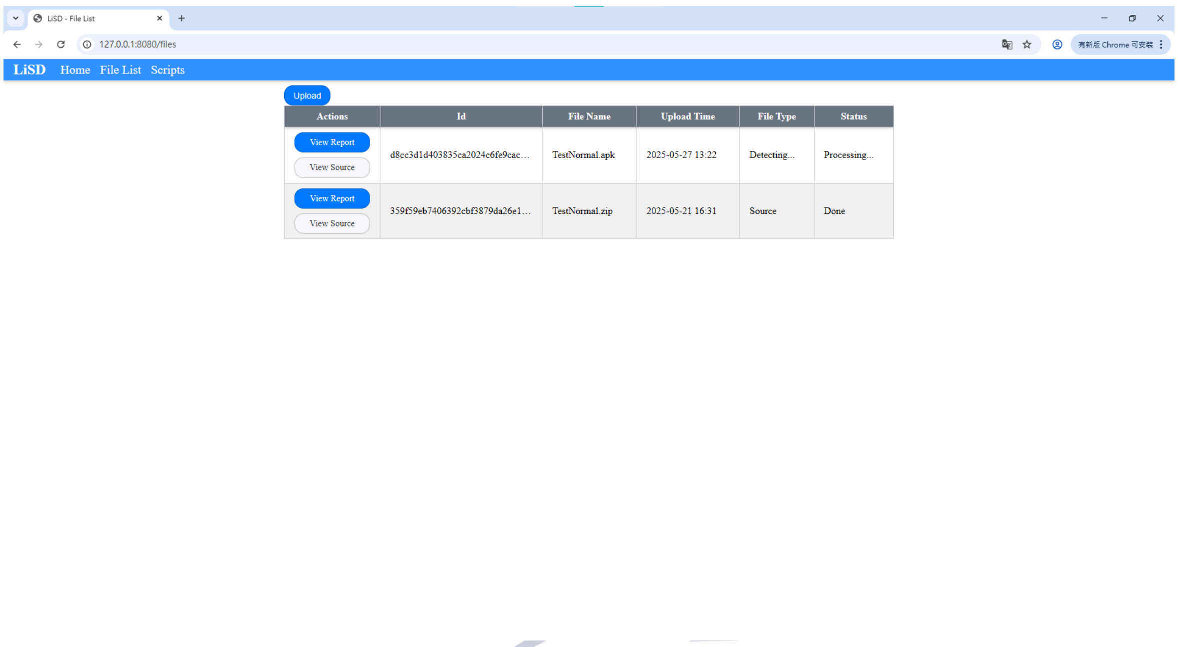
名稱	修改日期	類型	大小
app-debug	2025/1/19 下午 05:43	檔案資料夾	
app-debug2	2025/1/20 下午 01:54	檔案資料夾	
out	2025/1/19 下午 05:44	檔案資料夾	
out2	2025/1/20 上午 10:54	檔案資料夾	
app-debug.apk	2023/11/13 下午 03:04	APK 檔案	4,841 KB
app-debug.zip	2023/11/13 下午 03:04	WinRAR ZIP archi...	4,841 KB
diva.apk	2023/11/26 上午 11:03	APK 檔案	1,491 KB
diva.zip	2023/11/26 上午 11:03	WinRAR ZIP archi...	1,491 KB
facebook.apk	2024/12/25 下午 02:51	APK 檔案	74,525 KB
facebook.zip	2024/12/25 下午 02:51	WinRAR ZIP archi...	74,525 KB
TestNormal.zip	2025/4/12 下午 03:59	WinRAR ZIP archi...	156 KB

Figure 20: Upload File

TAIPEI TECH

2. Check the file type is correct (Figure 21).

   LiSD will identify whether the uploaded file is a source project or an APK file.

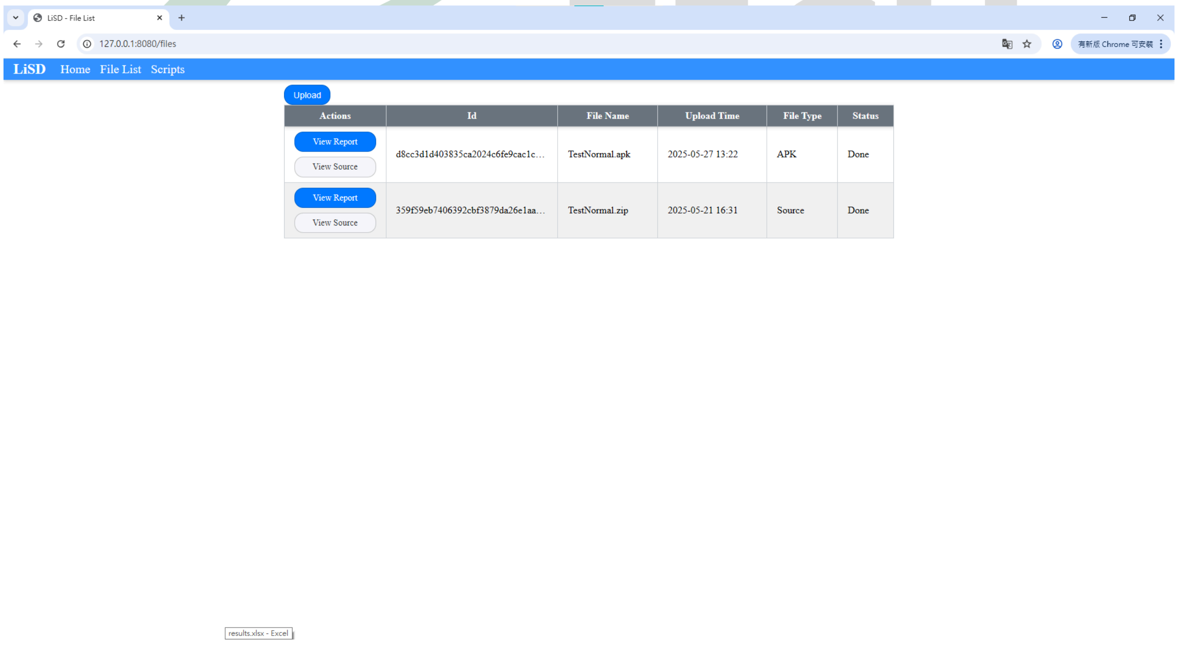


Actions	Id	File Name	Upload Time	File Type	Status
View Report View Source	d8cc3d1d403835ca2024c6fe9cac…	TestNormal.apk	2025-05-27 13:22	Detecting...	Processing...
View Report View Source	359f59eb7406392cbf3879da26e1…	TestNormal.zip	2025-05-21 16:31	Source	Done

Figure 21: Detect File Type

3. Check the analysis is done (Figure 22).

   The webpage will automatically refresh until all analyses are complete or an error occurs.



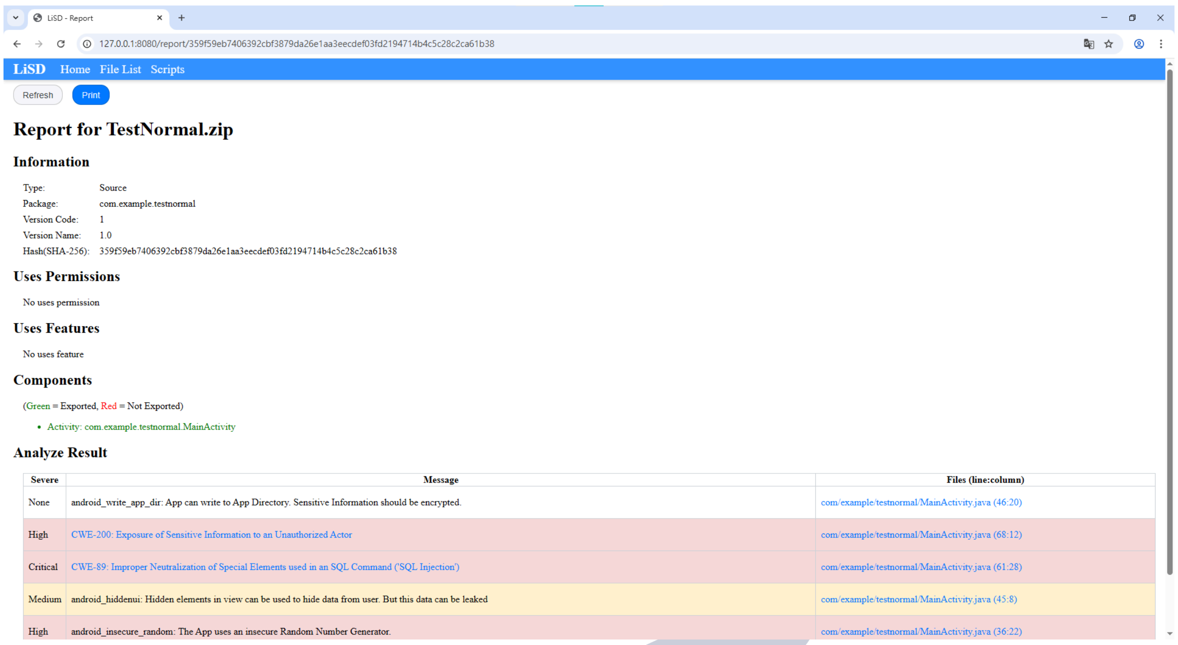
Actions	Id	File Name	Upload Time	File Type	Status
View Report View Source	d8cc3d1d403835ca2024c6fe9cac1c…	TestNormal.apk	2025-05-27 13:22	APK	Done
View Report View Source	359f59eb7406392cbf3879da26e1aa…	TestNormal.zip	2025-05-21 16:31	Source	Done

Figure 22: Analyze Finished

4. See what vulnerability was discovered and record the accuracy.

TAIPEI TECH

The report will list the information and vulnerabilities found in the file (Figure 23).



127.0.0.1:8080/report/359f59eb7406392cbf3879da26e1aa3eecdef03fd2194714b4c5c28c2ca61b38

Report for TestNormal.zip

Information

Type:	Source
Package:	com.example.testnormal
Version Code:	1
Version Name:	1.0
Hash(SHA-256):	359f59eb7406392cbf3879da26e1aa3eecdef03fd2194714b4c5c28c2ca61b38

Uses Permissions

No uses permission

Uses Features

No uses feature

Components

(Green = Exported, Red = Not Exported)

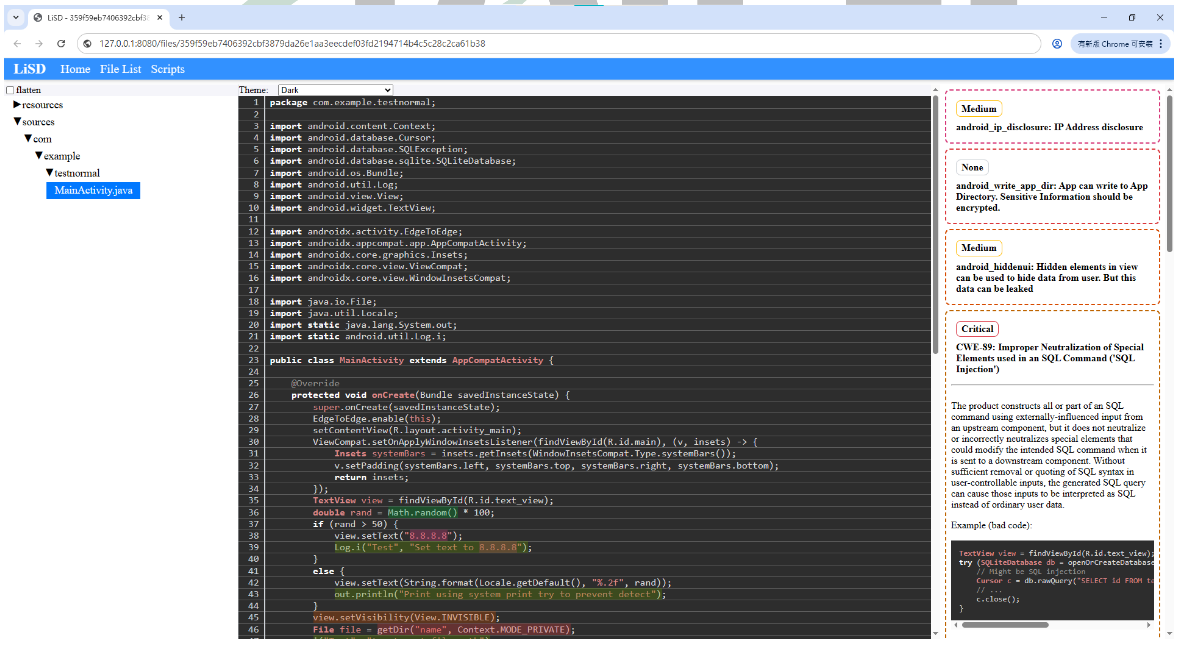
- Activity: com.example.testnormal.MainActivity

Analyze Result

Severe	Message	Files (line:column)
None	android_write_app_dir: App can write to App Directory. Sensitive Information should be encrypted.	com/example/testnormal/MainActivity.java (46:20)
High	CWE-200: Exposure of Sensitive Information to an Unauthorized Actor	com/example/testnormal/MainActivity.java (68:12)
Critical	CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	com/example/testnormal/MainActivity.java (61:28)
Medium	android_hiddenui: Hidden elements in view can be used to hide data from user. But this data can be leaked	com/example/testnormal/MainActivity.java (45:8)
High	android_insecure_random: The App uses an insecure Random Number Generator.	com/example/testnormal/MainActivity.java (36:22)

Figure 23: Report

Also, the discovered vulnerabilities are highlighted in the source code viewer in each file (Figure 24).



127.0.0.1:8080/files/359f59eb7406392cbf3879da26e1aa3eecdef03fd2194714b4c5c28c2ca61b38

Theme: Dark

```
package com.example.testnormal;

import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.TextView;

import androidx.activity.EdgeToEdge;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;

import java.io.File;
import java.util.Locale;
import static java.lang.System.out;
import static android.util.Log.i;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
            Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
            return insets;
        });
        TextView view = findViewById(R.id.text_view);
        double rand = Math.random() * 100;
        if (rand > 50) {
            view.setText("8.8.8.8");
            Log.i("Test", "Set text to 8.8.8.8");
        }
        else {
            view.setText(String.format(Locale.getDefault(), "%.2f", rand));
            out.println("Print using system print try to prevent detect");
        }
        view.setVisibility(View.INVISIBLE);
        File file = getDir("name", Context.MODE_PRIVATE);
```

Medium

android_ip_disclosure: IP Address disclosure

None

android_write_app_dir: App can write to App Directory. Sensitive Information should be encrypted.

Medium

android_hiddenui: Hidden elements in view can be used to hide data from user. But this data can be leaked

Critical

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

The product constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component. Without sufficient removal or quoting of SQL syntax in user-controllable inputs, the generated SQL query can cause those inputs to be interpreted as SQL instead of ordinary user data.

Example (bad code):

```
TextView view = findViewById(R.id.text_view);
try (SQLiteDatabase db = openOrCreateDatabase
    // Might be SQL injection
    Cursor c = db.rawQuery("SELECT id FROM te
    // ...
    c.close();
}
```

Figure 24: Visualize Vulnerability

Chapter 5 Result & Analysis

This chapter will collect the results of Section 4.2.3, than analyze and compare them with other tools according to their speeding, memeory usage and the support type.

5.1 Results

This section provide information of tools' activity which public on Github and their last commit date (see Table 5), and the apps and open source projects use in our implementation (see Table 6, Table 7)

Table 5: Other Tools Active

Name	Last commit date on Github
MobSF [29]	2025-05-05
Trueseeing [30]	2025-03-07
APKHunt [31]	2025-01-17
AUSERA [32]	2024-10-29
SPECK [33]	2024-10-25
DroidStatx [34]	2021-02-09
QARK [35]	2019-04-05
SUPER [36]	2018-12-10
Marvin Static Analyzer [37]	2018-11-23
JAADAS [38]	2017-04-12
AndroBugs [39]	2015-11-12

Table 6: APKs Information

Name	Version
WhatsApp Messenger	2.25.12.73
Facebook	508.0.0.74.47
YouTube	20.14.43
Google Photos	7.25.0.747549628
Google Chrome	135.0.7049.100
Google Play services	25.14.62
Google	16.13.45
Google Maps	25.16.06.747108139
Gboard - the Google Keyboard	15.2.08.736047990
Gmail	2025.04.06.748760211.Release

Table 7: Open Source Projects Information

Name	License	Version
Termux [40]	GPL-3.0 license	0.118.2
NewPipe [41]	GPL-3.0 license	0.27.7
LSPosed [42]	GPL-3.0 license	1.9.2
VirtualXposed [43]	GPL-3.0 license	0.22.0
xManager [44]	GPL-3.0 license	5.8

TAIPEI TECH

5.2 Analysis according to result

In this section, MobSF, Trueseeing, SPECK, and AndroBugs to be deployed as the comparison tools to perform the evaluation (Table 8).

Table 8: Other Tools Information

Name	Licence	Version
MobSF	GPL-3.0 license	4.3.2
Trueseeing	GPL-3.0 license	2.2.7
SPECK	GPL-3.0 license	Commit e40e862
AndroBugs	GPL-3.0 license	1.0.0

Table 9 shows that LiSD makes a better performance on the time process compared to MobSF, Trueseeing, SPECK, and AndroBigs. It could save 10%-90% in Facebook, Youtube, Google Chrome, Google, Google Maps, Gboard, and Gmail. There are two formats (APK and source code) will be utilized on the mobile applications, and both of them are very critical on security perspective. However, Trueseeing, SPECK and AndroBugs do not provide the ability on source code scanning.

Table 9: Processing Time

Name	LiSD	MobSF	Trueseeing	SPECK	AndroBugs
WhatsApp Messenger	59:43	11:49	07:09	41:13	00:13
Facebook	00:32	01:08	00:33	12:04	00:30
YouTube	01:12	05:58	05:20	47:12	01:14
Google Photos	14:49	09:20[1]	06:00	01:28:10	01:17
Google Chrome	00:25	06:51	02:32	44:07	00:34
Google Play services	01:16:07	27:05	16:44	15:04:14	01:19
Google	11:48	26:54	15:10	06:51:48	01:30
Google Maps	01:27	11:55	08:44	01:32:10	01:19
Gboard - the Google Keyboard	00:35	02:13	02:28	19:20	01:03
Gmail	07:26	09:03	07:09	01:50:08	01:13
Termux	03:39	00:12	-	-	-
NewPipe	05:46	00:18	-	-	-
LSPosed	03:09	00:14	-	-	-
VirtualXposed	02:09	00:12	-	-	-
xManager	01:31	00:12	-	-	-

[1] Sometimes it gets stuck at the “Android Behavior Analysis Started” step.

From Table 10, we observe that the memory usage on LiSD and MobSF appears to be comparable. In constrast, Trueseeing and SPECK demonstrate greater memory efficiency; however, they didn’t support source code analysis. Although MobSF plays a leading role among mobile detection tools, it may encounter limitations in certain aspects, such as “Android Behavior Analysis Started”.

TAIPEI TECH

Table 10: Maximum Memory Usage

Name	LiSD	MobSF	Trueseeing	SPECK	AndroBugs
WhatsApp Messenger	9.56GB	12.25GB	1.19GB	8.01GB	522MB
Facebook	3.55GB	1.73GB	560.35MB	1.59GB	735.3MB
YouTube	9.8GB	10.63GB	1.2GB	6.86GB	1.52GB
Google Photos	10.04GB	10.72GB	1.15GB	7.06GB	1.44GB
Google Chrome	6.07GB	5.67GB	1.08GB	3.32GB	1003MB
Google Play services	14.3GB	13.63GB	1.62GB	12.78GB	1.74GB
Google	16.46GB	18.39GB	1.97GB	13.71GB	1.72GB
Google Maps	14.19GB	14.31GB	1.36GB	9.27GB	1.62GB
Gboard - the Google Keyboard	6.61GB	4.79GB	629.41 MB	3.21GB	1.57GB
Gmail	8.65GB	11.06GB	1.38GB	8.37GB	1.58GB
Termux	2.59GB	206.8MB	-	-	-
NewPipe	3.36GB	180.9MB	-	-	-
LSPosed	2.96GB	473MB	-	-	-
VirtualXposed	2.74GB	149.5MB	-	-	-
xManager	6.99GB	135.2MB	-	-	-

5.3 Comparation with other tools

This section will compare our tool with other tools, including operating systems, programming languages, report viewing methods, interactivity, script scanning support, processing time, and memory usage.

Table 11 shows that LiSD, MobSF, and Trueseeing support all operating systems using Docker. However, SPECK only supports Linux, and AndroBugs supports Windows, Unix, and Linux.

Table 11: Functionality and Readability Comparison

	LiSD	MobSF	Trueseeing	SPECK	AndroBugs
OS	All (Docker)	All (Docker)	All (Docker)	Linux	Windows/Unix/ Linux
Language	Kotlin	Python	Python	Python	Python
Report Type	Web	Web	Web/Text	Text	Text
Interactivity	✓	✓	✗	✗	✗
Script Scanning	✓	✗	✗	✗	✗

TAIPEI TECH

Figure 25 show that LiSD has leading role in Facebook, Youtube, Google Chrome, Google, Google Maps, and Gboard. SPECK and AndroBugs have shorter processing times than LiSD, however they only support APK scanning.

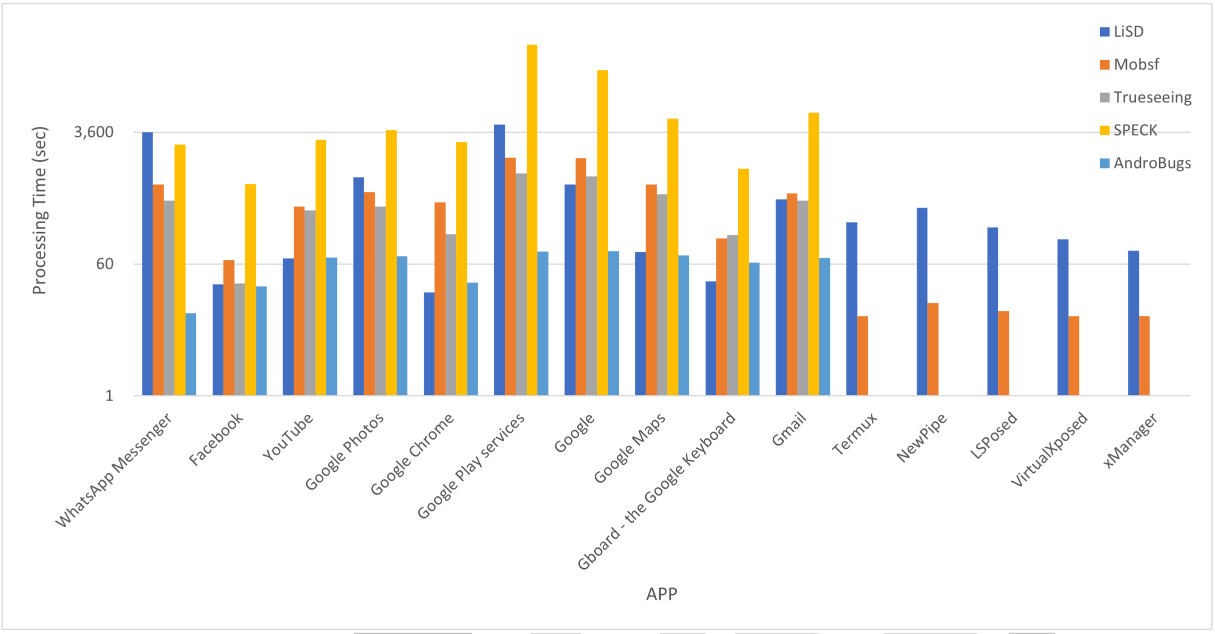


Figure 25: Processing Time in Second

TAIPEI TECH

As shown in Figure 26, LiSD and MobSF use more memory than other tools due to they use jadx to decompile. However, both tools support scanning APKs and source code.

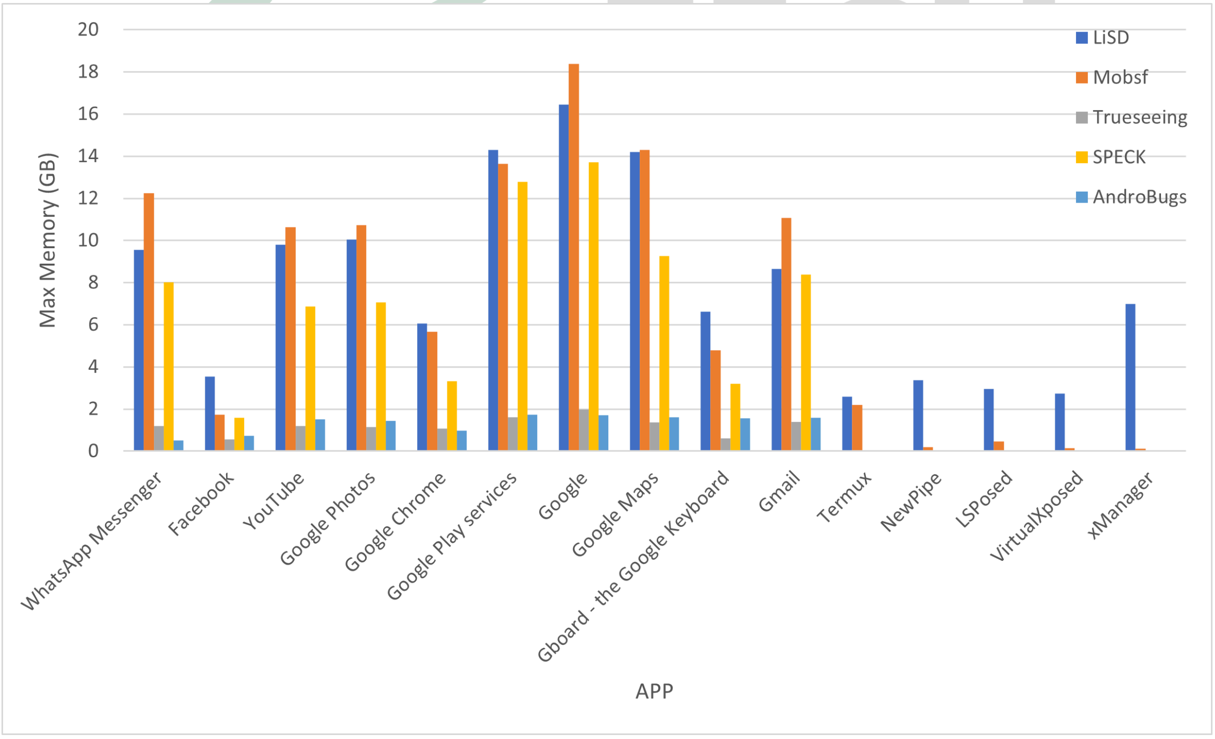


Figure 26: Memory Usage in Giga-byte

In summary, the above observations are as follows:

Although LiSD compared to other tools may exhibit slower performance when processing a single file of excessive size or a large number of files simultaneously, it nonetheless demonstrates several significant advantages. Firstly, it features a fast startup time, making it well-suited for real-time usage scenarios. Secondly, LiSD supports scanning of both APK files and source code, enhancing its applicability across diverse development environments. Furthermore, the tool allows for customizable scanning scripts, enabling users to create tailored scanning procedures based on specific use cases, thereby improving flexibility and extensibility. LiSD also provides a Dockerfile, facilitating rapid deployment within containerized environments and supporting integration into modern development or testing pipelines. Lastly, LiSD leverages Abstract Syntax Tree (AST)-based detection. This advantage minimizes false positives commonly found in traditional tools, thereby increasing detection accuracy and practical value.

To rigorously validate the accuracy of the proposed LiSD, we prepared a series of testing scripts and conducted a comparative evaluation against MobSF, a well-established open-source framework for mobile application security analysis. Based on the evaluation results, our tool exhibits superior detection accuracy in identifying potential issues related to Android-specific features—such as the incorrect use of certain import statements within the application code. Under equivalent test conditions, MobSF failed to detect these issues, revealing a limitation in its static analysis capabilities. This evaluation serves as an initial feasibility validation of our approach. To comprehensively assess its effectiveness, further testing with a broader range of sample applications will be required in future work.

Both MobSF and LiSD support static analysis of source code and APK files, making them valuable tools in the domain of mobile application security assessment. However, they differ fundamentally in their detection methodologies. MobSF primarily relies on regular expressions (RegEx) for identifying potential security risks. This approach offers the advantage of high-speed processing by quickly matching predefined patterns within the codebase. Nevertheless,

TAIPEI TECH

RegEx-based detection is inherently limited in its ability to interpret code semantics and contextual nuances. As a result, MobSF is prone to false positives, particularly when code structures vary in context. In contrast, LiSD employs AST-based analysis. While this method may be relatively slower when handling large file volumes, it provides deeper semantic understanding and contextual awareness, thereby significantly reducing misjudgments and enhancing the accuracy and reliability of detection outcomes.

Trueseeing is a lightweight static analysis tool. While it offers fast execution speed and low resource consumption, aligning with the needs of rapid assessments, it presents some limitations. Trueseeing also provides a Dockerfile, which facilitates easier deployment in containerized environments. However, its primary drawback is its restricted scope, as it only scans for vulnerabilities listed in the 2016 OWASP Mobile Top 10. Furthermore, Trueseeing does not support direct source code scanning, limiting its utility in early development stages where source code analysis is often crucial.

SPECK is a lightweight static analysis tool known for its minimal resource consumption, making it particularly suitable for constrained development or testing environments. One of its notable strengths lies in its ability to perform scans targeting specific rules, offering considerable flexibility for focused analysis. However, SPECK also presents several limitations. Most notably, it requires users to manually prepare and integrate the jadx decompiler, which raises the barrier to entry. Additionally, SPECK is limited to analyzing decompiled APK files and does not support direct scanning of source code, reducing its applicability during early-stage development. In summary, while SPECK is effective for rapid, targeted assessments, its overall completeness and integration capabilities remain limited compared to more comprehensive solutions.

AndroBugs is another static analysis tool worth considering. It characterized by low resource consumption and high execution speed, making it suitable for rapid initial security assessments of APK applications. However, the tool presents notable limitations. Firstly, AndroBugs is compatible only with Python 2.X and does not support Python 3.X, posing significant

TAIPEI TECH

challenges for integration and maintenance in modern development environments. Secondly, the tool is restricted to scanning decompiled APK files and lacks support for direct source code analysis, thereby limiting its applicability during early development stages. In summary, while AndroBugs is advantageous for quick assessments in resource-constrained settings, its maintainability and extensibility are limited.

TAIPEI TECH

Chapter 6 Conclusion & Future work

In summary, while LiSD presents a steeper learning curve compared to other tools, its high performance, reliability, and support for source code analysis make it a flexible and reliable solution for mobile application security. Notably, its ability to integrate with the Secure Software Development Life Cycle (SSDLC) [45] allows for the incorporation of security mechanisms at the early stages of design and development, aligning with the "design-in-security" principle. Given the increasing complexity of modern mobile threats, LiSD holds strong potential not only as a technical assessment tool but also as a valuable enabler in promoting secure software development practices.

For future development, LiSD can be enhanced by integrating Large Language Models (LLMs) [46] to improve its ability to identify and reason about emerging and variant vulnerabilities, thereby advancing its intelligent analysis capabilities. In addition, the issue of high memory usage observed in certain scenarios warrants deeper investigation, with a focus on performance profiling and root cause analysis to improve the tool's stability and scalability. At the same time, this study presents a preliminary feasibility validation of our proposed tool by comparing its detection capabilities against MobSF. While the results demonstrate the effectiveness of our tool in identifying certain Android-specific issues, such as incorrect usage of import statements, the current evaluation is limited in scope. To provide a more comprehensive assessment, future work will involve expanding the test set to include a wider variety of Android applications with diverse structures, libraries, and code patterns. This will allow us to further evaluate the tool's precision, recall, and overall robustness in real-world scenarios. In addition, we plan to explore the integration of dynamic analysis components and extend the tool's compatibility with Kotlin-specific language features and obfuscation patterns. Finally, integrating LiSD with DevSecOps [47] workflows could enable a more automated and intelligent detection process, supporting continuous integration, delivery, and security (CI/CD/CS) [48,49], and further solidifying its value in modern secure software development environments.

TAIPEI TECH

References

[1] N. J. Palatty, “30+ Malware Statistics You Need To Know In 2024.” Accessed: Mar. 04, 2025. [Online]. Available: https://www.getastra.com/blog/security-audit/malware-statistics/

[2] “Kaspersky predicts quantum-proof ransomware and advancements in mobile financial cyberthreats in 2025,” Dec. 2024. Accessed: Jun. 10, 2025. [Online]. Available: https://www.kaspersky.com/about/press-releases/kaspersky-predicts-quantum-proof-ransomware-and-advancements-in-mobile-financial-cyberthreats-in-2025

[3] Github, “What is SAST?,” Jul. 2024, Accessed: Mar. 04, 2025. [Online]. Available: https://github.com/resources/articles/security/what-is-sast

[4] “Google Play.” Accessed: Mar. 04, 2025. [Online]. Available: https://play.google.com/store/apps

[5] “Github.” Accessed: Mar. 04, 2025. [Online]. Available: https://github.com/

[6] M. S. Thakur et al., “OWASP Mobile Top 10.” Accessed: Mar. 04, 2025. [Online]. Available: https://owasp.org/www-project-mobile-top-10/

[7] X. Xu et al., “Symbol Preference Aware Generative Models for Recovering Variable Names from Stripped Binary.” [Online]. Available: https://arxiv.org/abs/2306.02546

[8] H. Rathore, S. Chari, N. Verma, S. K. Sahay, and M. Sewak, “Android Malware Detection Based on Static Analysis and Data Mining Techniques: A Systematic Literature Review,” in Broadband Communications, Networks, and Systems, W. Wang and J. Wu, Eds., Cham: Springer Nature Switzerland, 2023, pp. 51–71.

[9] N. Mauthe, U. Kargén, and N. Shahmehri, “A Large-Scale Empirical Study of Android App Decompilation,” in 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 2021, pp. 400–410. doi: 10.1109/SANER50967.2021.00044.

TAIPEI TECH

[10] J.-M. Mineau and J.-F. Lalande, “Evaluating the Reusability of Android Static Analysis Tools,” in International Conference on Software and Software Reuse, 2024, pp. 153–170.

[11] N. Abolhassani and W. G. Halfond, “A Component-Sensitive Static Analysis Based Approach for Modeling Intents in Android Apps,” in 2023 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2023, pp. 97–109.

[12] Rahul, “A hands-on introduction to static code analysis,” deepsource, Apr. 2020, Accessed: Mar. 23, 2025. [Online]. Available: https://deepsource.com/blog/introduction-static-code-analysis

[13] J. Zhu, K. Li, S. Chen, L. Fan, J. Wang, and X. Xie, “A Comprehensive Study on Static Application Security Testing (SAST) Tools for Android,” IEEE Transactions on Software Engineering, vol. 50, no. 12, pp. 3385–3402, 2024, doi: 10.1109/TSE.2024.3488041.

[14] “ANTLR.” Accessed: Mar. 04, 2025. [Online]. Available: https://www.antlr.org/

[15] N. Team, “APK,” Handbook for CTFers. Springer, pp. 269–293, 2022.

[16] R. S. Menshawy, A. H. Yousef, and A. Salem, “Code smells and detection techniques: a survey,” in 2021 international mobile, intelligent, and ubiquitous computing conference (MIUCC), 2021, pp. 78–83.

[17] “Ktor.” Accessed: Mar. 04, 2025. [Online]. Available: https://ktor.io/

[18] “PostgreSQL.” Accessed: Mar. 04, 2025. [Online]. Available: https://www.postgresql.org/

[19] “Exposed.” Accessed: Apr. 22, 2025. [Online]. Available: https://www.jetbrains.com/exposed/

[20] “H2.” Accessed: Apr. 22, 2025. [Online]. Available: https://www.h2database.com/

[21] “Jackson.” Accessed: Mar. 04, 2025. [Online]. Available: https://github.com/FasterXML/jackson

[22] “Docker.” Accessed: Apr. 22, 2025. [Online]. Available: https://www.docker.com/

[23] “Corretto/Corretto-21.” Accessed: May 09, 2025. [Online]. Available: https://github.com/corretto/corretto-21

TAIPEI TECH

[24] "Kotlin Script Support." Accessed: Apr. 22, 2025. [Online]. Available: https://github.com/Kotlin/KEEP/blob/master/proposals/scripting-support.md

[25] "JADX." Accessed: Mar. 04, 2025. [Online]. Available: https://github.com/skylot/jadx

[26] "Git." Accessed: Apr. 08, 2025. [Online]. Available: https://git-scm.com/

[27] W. contributors, "GitHub — Wikipedia, The Free Encyclopedia." Accessed: Feb. 24, 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=GitHub&oldid=1277160746

[28] "Android Rank." Accessed: Apr. 21, 2025. [Online]. Available: https://androidrank.org/android-most-popular-google-play-apps?category=all&sort=4&price=all

[29] "MobSF/Mobile-Security-Framework-MobSF." Accessed: May 20, 2025. [Online]. Available: https://github.com/MobSF/Mobile-Security-Framework-MobSF

[30] "alterakey/trueseeing." Accessed: May 20, 2025. [Online]. Available: https://github.com/alterakey/trueseeing

[31] "Cyber-Buddy/APKHunt." Accessed: May 20, 2025. [Online]. Available: https://github.com/Cyber-Buddy/APKHunt

[32] "tjusenchen/AUSERA." Accessed: May 20, 2025. [Online]. Available: https://github.com/tjusenchen/AUSERA

[33] "SPRITZ-Research-Group/SPECK." Accessed: May 20, 2025. [Online]. Available: https://github.com/SPRITZ-Research-Group/SPECK

[34] "clviper/droidstatx." Accessed: May 20, 2025. [Online]. Available: https://github.com/clviper/droidstatx

[35] "linkedin/qark." Accessed: May 20, 2025. [Online]. Available: https://github.com/linkedin/qark

[36] "SUPERAndroidAnalyzer/super." Accessed: May 20, 2025. [Online]. Available: https://github.com/SUPERAndroidAnalyzer/super

[37] "programa-stic/Marvin-static-Analyzer." Accessed: May 20, 2025. [Online]. Available: https://github.com/programa-stic/Marvin-static-Analyzer

TAIPEI TECH

[38] "flankerhqd/JAADAS." Accessed: May 20, 2025. [Online]. Available: https://github.com/flankerhqd/JAADAS

[39] "AndroBugs/AndroBugs_Framework." Accessed: May 20, 2025. [Online]. Available: https://github.com/AndroBugs/AndroBugs_Framework

[40] "termux/termux-app." Accessed: May 20, 2025. [Online]. Available: https://github.com/termux/termux-app

[41] "TeamNewPipe/NewPipe." Accessed: May 20, 2025. [Online]. Available: https://github.com/TeamNewPipe/NewPipe

[42] "LSPosed/LSPosed." Accessed: May 20, 2025. [Online]. Available: https://github.com/LSPosed/LSPosed

[43] "android-hacker/VirtualXposed." Accessed: May 20, 2025. [Online]. Available: https://github.com/android-hacker/VirtualXposed

[44] "Team-xManager/xManager." Accessed: May 20, 2025. [Online]. Available: https://github.com/Team-xManager/xManager

[45] vishaldhaygude01, "What is Secure Software Development Life Cycle (SSDLC)?," GeeksforGeeks, Jan. 2024, Accessed: Jun. 10, 2025. [Online]. Available: https://www.geeksforgeeks.org/what-is-secure-software-development-life-cycle-ssdlc/

[46] M. Shanahan, "Talking about Large Language Models," Commun. ACM, vol. 67, no. 2, pp. 68–79, Jan. 2024, doi: 10.1145/3624724.

[47] H. Myrbakken and R. Colomo-Palacios, "DevSecOps: A Multivocal Literature Review," in Software Process Improvement and Capability Determination, A. Mas, A. Mesquida, R. V. O'Connor, T. Rout, and A. Dorling, Eds., Cham: Springer International Publishing, 2017, pp. 17–29.

[48] baeldung, "Continuous Integration vs. Continuous Deployment vs. Continuous Delivery," Baeldung, Jun. 2024, Accessed: Jun. 10, 2025. [Online]. Available: https://www.baeldung.com/cs/continuous-integration-deployment-delivery

TAIPEI TECH

[49] P. Labs, “What Is Continuous Security Validation?,” Picus, Jun. 2025, Accessed: Jun. 10, 2025. [Online]. Available: https://www.picussecurity.com/resource/glossary/what-is-continuous-security-validation

TAIPEI TECH

Appendix

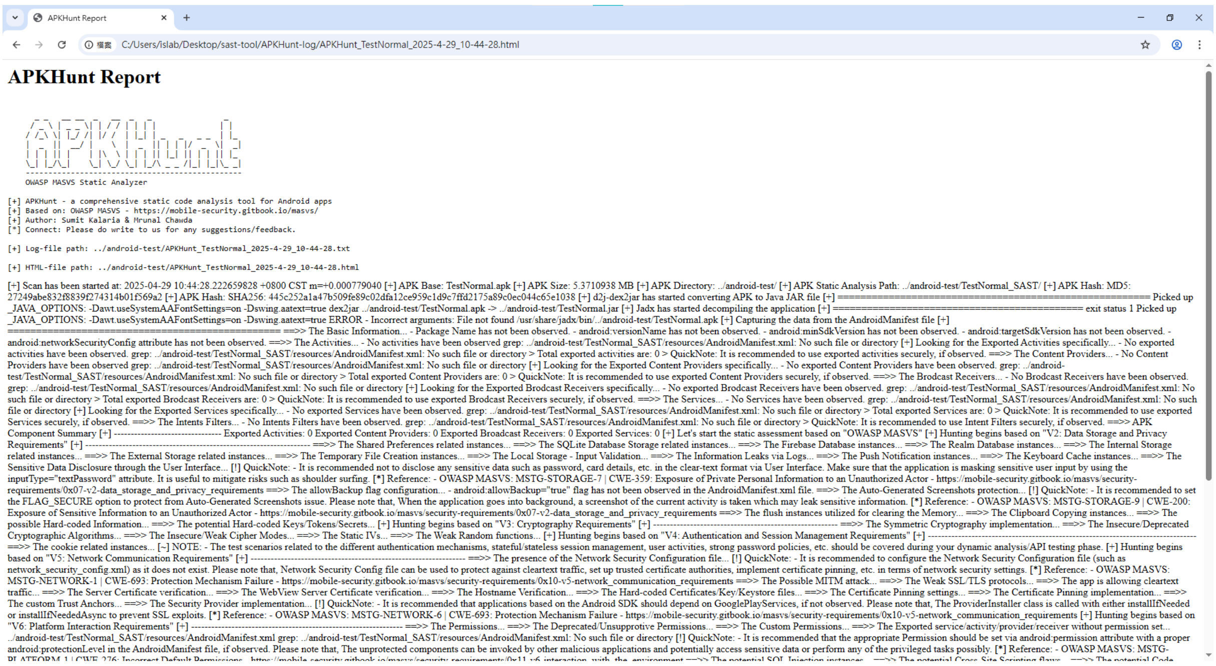
A. Other Tools

A.I. Build

This section will show what I get (error/report) when I use/build these tool.

A.I.a. APKHunt

APKHunt successfully run, but it shows an error `File not found /usr/share/jadx/bin/../android-test/TestNormal.apk` in the report


APKHunt Report
C:/Users/islab/Desktop/sast-tool/APKHunt-log/APKHunt_TestNormal_2025-4-29_10-44-28.html

APKHunt Report

```
    ---------------------------------------------
    OWASP MASVS Static Analyzer

[+] APKHunt - a comprehensive static code analysis tool for Android apps
[+] Based on: OWASP MASVS - https://mobile-security.gitbook.io/masvs/
[+] Author: Sumit Kalaria & Mrunal Chawda
[*] Connect: Please do write to us for any suggestions/feedback.

[+] Log-file path: ../android-test/APKHunt_TestNormal_2025-4-29_10-44-28.txt

[+] HTML-file path: ../android-test/APKHunt_TestNormal_2025-4-29_10-44-28.html
```

[+] Scan has been started at: 2025-04-29 10:44:28.222659828 +0800 CST m=+0.000779040 [+] APK Base: TestNormal.apk [+] APK Size: 5.3710938 MB [+] APK Directory: ../android-test/ [+] APK Static Analysis Path: ../android-test/TestNormal_SAST/ [+] APK Hash: MD5: 27249abe832f8839f274314b01f569a2 [+] APK Hash: SHA256: 445c252a1a47b509fe89c02dfa12ce959c1d9c7ffd2175a89c0ec044c65e1038 [+] d2j-dex2jar has started converting APK to Java JAR file [+] == Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true dex2jar ../android-test/TestNormal.apk -> ../android-test/TestNormal.jar [+] Jadx has started decompiling the application [+] == exit status 1 Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true ERROR - Incorrect arguments: File not found /usr/share/jadx/bin/../android-test/TestNormal.apk [+] Capturing the data from the AndroidManifest file [+] == ==>> The Basic Information... - Package Name has not been observed. - android:versionName has not been observed. - android:minSdkVersion has not been observed. - android:targetSdkVersion has not been observed. - android:networkSecurityConfig attribute has not been observed. ==>> The Activities... - No activities have been observed grep: ../android-test/TestNormal_SAST/resources/AndroidManifest.xml: No such file or directory [+] Looking for the Exported Activities specifically... - No exported activities have been observed. grep: ../android-test/TestNormal_SAST/resources/AndroidManifest.xml: No such file or directory > Total exported activities are: 0 > QuickNote: It is recommended to use exported activities securely, if observed. ==>> The Content Providers... - No Content Providers have been observed grep: ../android-test/TestNormal_SAST/resources/AndroidManifest.xml: No such file or directory [+] Looking for the Exported Content Providers specifically... - No exported Content Providers have been observed. grep: ../android-test/TestNormal_SAST/resources/AndroidManifest.xml: No such file or directory > Total exported Content Providers are: 0 > QuickNote: It is recommended to use exported Content Providers securely, if observed. ==>> The Brodcast Receivers... - No Brodcast Receivers have been observed. grep: ../android-test/TestNormal_SAST/resources/AndroidManifest.xml: No such file or directory [+] Looking for the Exported Brodcast Receivers specifically... - No exported Brodcast Receivers have been observed. grep: ../android-test/TestNormal_SAST/resources/AndroidManifest.xml: No such file or directory > Total exported Brodcast Receivers are: 0 > QuickNote: It is recommended to use exported Brodcast Receivers securely, if observed. ==>> The Services... - No Services have been observed. grep: ../android-test/TestNormal_SAST/resources/AndroidManifest.xml: No such file or directory [+] Looking for the Exported Services specifically... - No exported Services have been observed. grep: ../android-test/TestNormal_SAST/resources/AndroidManifest.xml: No such file or directory > Total exported Services are: 0 > QuickNote: It is recommended to use exported Services securely, if observed. ==>> The Intents Filters... - No Intents Filters have been observed. grep: ../android-test/TestNormal_SAST/resources/AndroidManifest.xml: No such file or directory > QuickNote: It is recommended to use Intent Filters securely, if observed. ==>> APK Component Summary [+] ------------------------------ Exported Activities: 0 Exported Content Providers: 0 Exported Broadcast Receivers: 0 Exported Services: 0 [+] Let's start the static assessment based on "OWASP MASVS" [+] Hunting begins based on "V2: Data Storage and Privacy Requirements" [+] -- ==>> The Shared Preferences related instances... ==>> The SQLite Database Storage related instances... ==>> The Firebase Database instances... ==>> The Realm Database instances... ==>> The Internal Storage related instances... ==>> The External Storage related instances... ==>> The Temporary File Creation instances... ==>> The Local Storage - Input Validation... ==>> The Information Leaks via Logs... ==>> The Push Notification instances... ==>> The Keyboard Cache instances... ==>> The Sensitive Data Disclosure through the User Interface... [!] QuickNote: - It is recommended not to disclose any sensitive data such as password, card details, etc. in the clear-text format via User Interface. Make sure that the application is masking sensitive user input by using the inputType="textPassword" attribute. It is useful to mitigate risks such as shoulder surfing. [*] Reference: - OWASP MASVS: MSTG-STORAGE-7 | CWE-359: Exposure of Private Personal Information to an Unauthorized Actor - https://mobile-security.gitbook.io/masvs/security-requirements/0x07-v2-data_storage_and_privacy_requirements ==>> The allowBackup flag configuration... - android:allowBackup="true" flag has not been observed in the AndroidManifest.xml file. ==>> The Auto-Generated Screenshots protection... [!] QuickNote: - It is recommended to set the FLAG_SECURE option to protect from Auto-Generated Screenshots issue. Please note that, When the application goes into background, a screenshot of the current activity is taken which may leak sensitive information. [*] Reference: - OWASP MASVS: MSTG-STORAGE-9 | CWE-200: Exposure of Sensitive Information to an Unauthorized Actor - https://mobile-security.gitbook.io/masvs/security-requirements/0x07-v2-data_storage_and_privacy_requirements ==>> The flush instances utilized for clearing the Memory... ==>> The Clipboard Copying instances... ==>> The possible Hard-coded Information... ==>> The potential Hard-coded Keys/Tokens/Secrets... [+] Hunting begins based on "V3: Cryptography Requirements" [+] -- ==>> The Symmetric Cryptography implementation... ==>> The Insecure/Deprecated Cryptographic Algorithms... ==>> The Insecure/Weak Cipher Modes... ==>> The Static IVs... ==>> The Weak Random functions... [+] Hunting begins based on "V4: Authentication and Session Management Requirements" [+] -- ==>> The cookie related instances... [~] NOTE: - The test scenarios related to the different authentication mechanisms, stateful/stateless session management, user activities, strong password policies, etc. should be covered during your dynamic analysis/API testing phase. [+] Hunting begins based on "V5: Network Communication Requirements" [+] -- ==>> The presence of the Network Security Configuration file... [!] QuickNote: - It is recommended to configure the Network Security Configuration file (such as network_security_config.xml) as it does not exist. Please note that, Network Security Config file can be used to protect against cleartext traffic, set up trusted certificate authorities, implement certificate pinning, etc. in terms of network security settings. [*] Reference: - OWASP MASVS: MSTG-NETWORK-1 | CWE-693: Protection Mechanism Failure - https://mobile-security.gitbook.io/masvs/security-requirements/0x10-v5-network_communication_requirements ==>> The Possible MITM attack... ==>> The Weak SSL/TLS protocols... ==>> The app is allowing cleartext traffic... ==>> The Server Certificate verification... ==>> The WebView Server Certificate verification... ==>> The Hostname Verification... ==>> The Hard-coded Certificates/Key/Keystore files... ==>> The Certificate Pinning settings... ==>> The Certificate Pinning implementation... ==>> The custom Trust Anchors... ==>> The Security Provider implementation... [!] QuickNote: - It is recommended that applications based on the Android SDK should depend on GooglePlayServices, if not observed. Please note that, The ProviderInstaller class is called with either installIfNeeded or installIfNeededAsync to prevent SSL exploits. [*] Reference: - OWASP MASVS: MSTG-NETWORK-6 | CWE-693: Protection Mechanism Failure - https://mobile-security.gitbook.io/masvs/security-requirements/0x10-v5-network_communication_requirements [+] Hunting begins based on "V6: Platform Interaction Requirements" [+] -- ==>> The Permissions... ==>> The Deprecated/Unsupprotive Permissions... ==>> The Custom Permissions... ==>> The Exported service/activity/provider/receiver without permission set... ../android-test/TestNormal_SAST/resources/AndroidManifest.xml grep: ../android-test/TestNormal_SAST/resources/AndroidManifest.xml: No such file or directory [!] QuickNote: - It is recommended that the appropriate Permission should be set via android:permission attribute with a proper android:protectionLevel in the AndroidManifest file, if observed. Please note that, The unprotected components can be invoked by other malicious applications and potentially access sensitive data or perform any of the privileged tasks possibly. [*] Reference: - OWASP MASVS: MSTG-PLATFORM-1 | CWE-276: Incorrect Default Permissions - https://mobile-security.gitbook.io/masvs/security-requirements/0x11-v6-interaction_with_the_environment ==>> The potential SQL Injection instances... ==>> The potential Cross-Site Scripting flaws... ==>> The potential Code

A.I.b. AndroBugs

The report will look like this:

```
*************************************************************************
**   AndroBugs Framework - Android App Security Vulnerability Scanner  **
**                            version: 1.0.0                           **
**     author: Yu-Cheng Lin (@AndroBugs, http://www.AndroBugs.com)     **
**               contact: androbugs.framework@gmail.com                **
*************************************************************************
```

```
Platform: Android

Package Name: com.example.testnormal

Package Version Name: 1.0

Package Version Code: 1

Min Sdk: 24

Target Sdk: 34

MD5   : 27249abe832f8839f274314b01f569a2

SHA1  : 47b0142d7e592afd81998933f4f03102d6e89d79

SHA256: 445c252a1a47b509fe89c02dfa12ce959c1d9c7ffd2175a89c0ec044c65e1038

SHA512:

a96dc8aaaafc2b98acedea584b98d6ad6c9d4885e2b179708b5e0a2be6f803108bb4feaf07c225645d7acbc0b13

Analyze Signature:

71a0b2abd2d45412594af4a4e065e2df036ac2c2cdc21c10b7cf3743d6c1a83128770145541cbc9f79b6b0ece20

-------------------------------------------------------------------------------------------

(Details ...)

------------------------------------------------------------

AndroBugs analyzing time: 5.684021 secs

Total elapsed time: 26.435392 secs
```

TAIPEI TECH

A.I.c. QARK

We didn't succeed when we tried to install QARK. Here are the steps for installing it from GitHub:

```
$ pip install --user qark  # --user is only needed if not using a virtualenv

$ qark --help
```

After we finish the installation, we get the error below:

```
$ qark --help

Command 'qark' not found, did you mean:

  command 'quark' from deb quark-engine

  command 'mark' from deb nmh

  command 'mark' from deb mailutils-mh
  command 'ark' from deb ark
Try: sudo apt install <deb name>


$ python qark/qark.py --apk ~/android-test/TestNormal.apk
Traceback (most recent call last):
  File "/home/user/qark/qark/qark.py", line 11, in <module>
    from qark.apk_builder import APKBuilder
  File "/home/user/qark/qark/qark.py", line 11, in <module>
    from qark.apk_builder import APKBuilder
ModuleNotFoundError: No module named 'qark.apk_builder'; 'qark' is not a
package
```

A.I.d. JAADAS

JAADAS will always throw NPE. We found that some properties are missing.

```
PS C:\Users\islab\Desktop\jaadas-0.1> java -jar .\jade-0.1.jar vulnanalysis --
fastanalysis -f ..\apk\Facebook.apk -p C:
\Users\islab\AppData\Local\Android\Sdk\platforms --configDirPath config/
********************************************
enabled plugins: implements custom verifier that always return true
Webview js file access misconfigurations
Webview ssl handler impl onReceivedSslError, lead to SSL vulnerability
X509TrustManager empty impl, lead to SSL vulnerability
FAKEID reloaded vulnerability
Check webview save password disabled or not
Scan for ZipEntry vulnerable to unzip directory traversal vulnerability
********************************************
********************************************
enabled modules: constapicheck, crash analysis
********************************************
Using 'C:
\Users\islab\AppData\Local\Android\Sdk\platforms\android-35\android.jar' as
android.jar
```

TAIPEI TECH

```
Exception in thread "main" java.lang.NullPointerException: Cannot invoke
"String.split(String)" because the return value of
"java.lang.System.getProperty(String)" is null
        at soot.JastAddJ.Program.initPaths(Program.java:350)
        at soot.SootResolver.<init>(SootResolver.java:87)
        at soot.Singletons.soot_SootResolver(Singletons.java:802)
        at soot.SootResolver.v(SootResolver.java:91)
        at soot.Scene.tryLoadClass(Scene.java:670)
        at soot.Scene.loadBasicClasses(Scene.java:1268)
        at soot.Scene.loadNecessaryClasses(Scene.java:1347)
        at
org.k33nteam.jade.drivers.CheckDriver.prepareMethodTraversal(CheckDriver.scala:77)
        at
org.k33nteam.jade.drivers.CheckDriver.fastentry(CheckDriver.scala:98)
        at main$.main(main.scala:75)
        at main.main(main.scala)
```

A.I.e. Marvin Static Analyzer

A `NoClassDefFoundError` occurred during scanning. Here is the running log:

```
$ python2.7 MarvinStaticAnalyzer.py apk/
===================================APKS===================================
apk/TestNormal.apk
===================================APKS===================================
Permission not defined by manifest: android.permission.DUMP
cd SAAF-MODULE; java -jar SAAF.jar -nq -nodb -hl "/home/user/Marvin-static-
Analyzer/apk/TestNormal.apk"
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/commons/
cli/UnrecognizedOptionException
        at java.base/java.lang.Class.forName0(Native Method)
        at java.base/java.lang.Class.forName(Class.java:467)
        at
```

```
org.eclipse.jdt.internal.jarinjarloader.JarRsrcLoader.main(JarRsrcLoader.java:56)
Caused by: java.lang.ClassNotFoundException:
org.apache.commons.cli.UnrecognizedOptionException
        at java.base/
java.net.URLClassLoader.findClass(URLClassLoader.java:445)
        at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:592)
        at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:525)
        ... 3 more
There was an error with the analysis
Traceback (most recent call last):
  File "MarvinStaticAnalyzer.py", line 102, in worker
    print analyze_vulnerabilities_from(filename)
  File "MarvinStaticAnalyzer.py", line 48, in analyze_vulnerabilities_from
    return analyze_vulnerabilities(apk_file,_apk,vm,dx)
  File "MarvinStaticAnalyzer.py", line 92, in analyze_vulnerabilities
    final_report.update(SAAFModuleAnalyzer(apk_file).get_clean_report())
  File "/home/user/Marvin-static-Analyzer/SAAFAnalyzer.py", line 61, in
get_clean_report
    report = self.open_xml_report()
  File "/home/user/Marvin-static-Analyzer/SAAFAnalyzer.py", line 55, in
open_xml_report
    xml = max(glob.glob('SAAF-MODULE/reports/Report-' + self.file + '*'),
key=os.path.getctime)
ValueError: max() arg is an empty sequence
```

A.I.f. SUPER

We got a compile error. Here is the running log:

```
$ ./debian_build.sh
...
warning: `super-analyzer` (lib) generated 17 warnings
error: could not compile `super-analyzer` (lib) due to 1 previous error; 17
warnings emitted
```

TAIPEI TECH

```
Caused by:
  process didn't exit successfully: `/root/.rustup/toolchains/stable-x86_64-
unknown-linux-gnu/bin/rustc ...` (exit status: 1)
cargo-deb: Build failed
```

A.I.g. SPECK

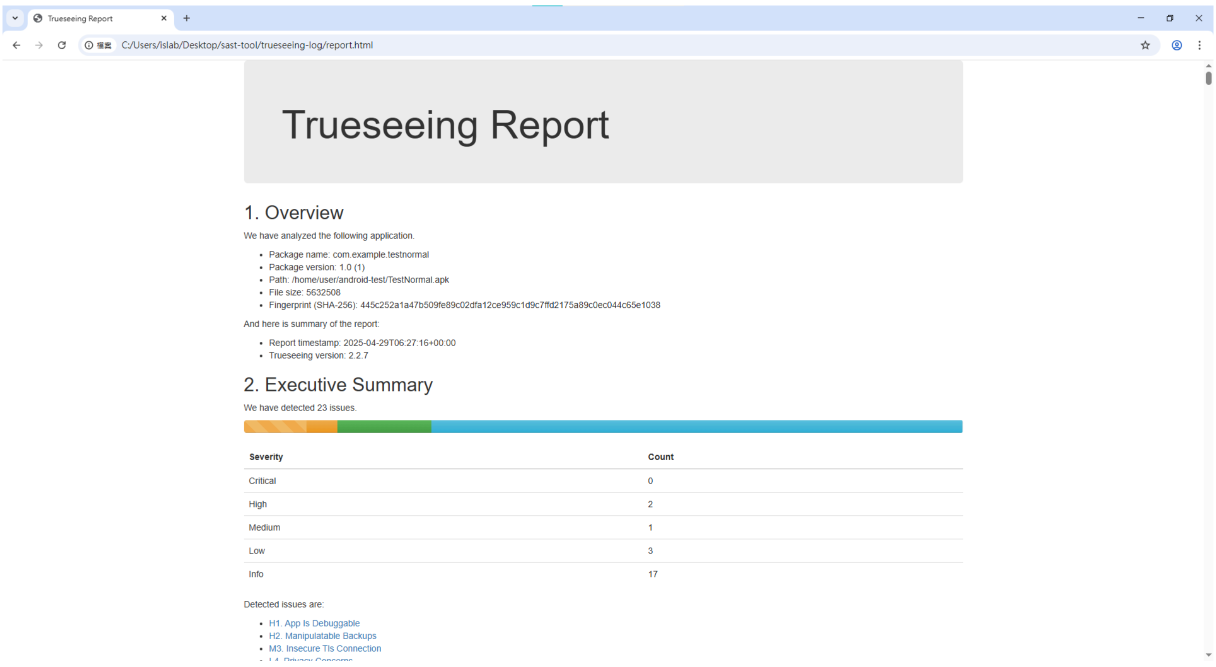
The report only shows on terminal:

```
[+] Extracting TestNormal.apk
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
INFO  - loading ...
INFO  - processing ...
ERROR - finished with errors, count: 19
[+] TestNormal.apk extracted


[!] Analysing TestNormal


[§] RULE: 1
...
```

A.I.h. Trueseeing

The report is detailed.

TAIPEI TECH


Trueseeing Report

C:/Users/islab/Desktop/sast-tool/trueseeing-log/report.html

Trueseeing Report

1. Overview

We have analyzed the following application.

- Package name: com.example.testnormal
- Package version: 1.0 (1)
- Path: /home/user/android-test/TestNormal.apk
- File size: 5632508
- Fingerprint (SHA-256): 445c252a1a47b509fe89c02dfa12ce959c1d9c7ffd2175a89c0ec044c65e1038

And here is summary of the report:

- Report timestamp: 2025-04-29T06:27:16+00:00
- Trueseeing version: 2.2.7

2. Executive Summary

We have detected 23 issues.

Severity	Count
Critical	0
High	2
Medium	1
Low	3
Info	17

Detected issues are:

- H1. App Is Debuggable
- H2. Manipulatable Backups
- M3. Insecure Tls Connection

A.I.i. DroidStatx

We received an error during setup. Here is the setup log:

```
$ python3 droidstatx.py --apk ../android-test/TestNormal.zip
/home/user/droidstatx/droidstatx.py:38: SyntaxWarning: "is not" with a
literal. Did you mean "!="?
  if location is not "":
Traceback (most recent call last):
  File "/home/user/droidstatx/droidstatx.py", line 5, in <module>
    from App import App
  File "/home/user/droidstatx/App.py", line 2, in <module>
    from androguard.misc import AnalyzeAPK
ModuleNotFoundError: No module named 'androguard.misc'
```

A.I.j. AUSERA

"No such file or directory" occurred.

```
$ python2.7 apk-engine.py ~/AUSERA/ausera-main $JAVA_HOME /usr/lib/android-sdk
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Traceback (most recent call last):
  File "apk-engine.py", line 103, in <module>
```

TAIPEI TECH

```
    main()
  File "apk-engine.py", line 58, in main
    os.mknod(first_file)
OSError: [Errno 2] No such file or directory
```

TAIPEI TECH