The Data Security Detection for Health Applications with an Advanced Pipeline Tool

Wenping Du (a1762784)
The University of Adelaide

***Abstract*—Due to the diversity of Android application sources (stores) [1], data security issues become uncontrollable. The data security issues of financial apps have attracted much attention of many researchers [2]. Although there is very sensitive user information involved in health apps, the current research is rare, therefore our team pays special attention. The purpose of this research is to combine the technologies of static and dynamic detection, then optimize several of the state-of-art detection tools (FlowDroid [3], Frida [4], MobSF [5], Monkey [6]), and assemble them into a pipeline tool, focusing on the detection and analysis of health applications. We built a detection system based on the client-server (CS) architecture. In this detection system, the client side displays the list of applications (APK files) to be detected, and the server side starts the pipeline tool to detect and analyze the APK transmitted by the client side. After the analysis completed, we can view the final analysis report on the client side. In this study, I aim to develop the Android client-side and focus on some optimization of the state-of-art detection tools. Besides, for the dynamic detection part, since the automated testing tool Monkey cannot fully meet our testing requirements, I wrote an automated testing tool that can traverse the component types in the application pages and then simulate clicks on it or analogue input.**

***Index Terms*—security detection, health application, pipeline tool, data leakage, automated testing**

I. Introduction

With the rapid development of smart phones and mobile Internet, the data security of mobile phone applications has attracted much attention in recent years. Especially mobile devices based on the Android operating system, which are different from iOS, the diversity of Android application stores [1] makes data security issues uncontrollable. As of the present, the data security issues of financial apps have attracted the attention of many researchers [2], but for health apps, although the user information involves is very sensitive, current research is rare. This attracted the attention of our team. This research will focus on the detection and analysis of Android health applications to explore possible data security issues.

The study started with static and dynamic detection, and investigated several of the most advanced detection tools. Some capabilities of these tools in identifying and detecting data leaks are limited. Thus, we hope to optimize and combine them, and develop a new tool that can better detect data leakage issues.

Identify applicable funding agency here. If none, delete this.

Since the detection process is time-consuming and resource-intensive, we built a detection system based on the client-server (CS) architecture. In the detection system, the client side displays the list of applications (APK files) to be detected, and clicks the detection button to transmit the relevant data of the applications that need to be detected to the server, and the server side starts the pipeline tool (the tool consists of three parts, Flowdroid, Frida and NLP) for detection and analysis. After completing the analysis, the client-side can display the analysis report.

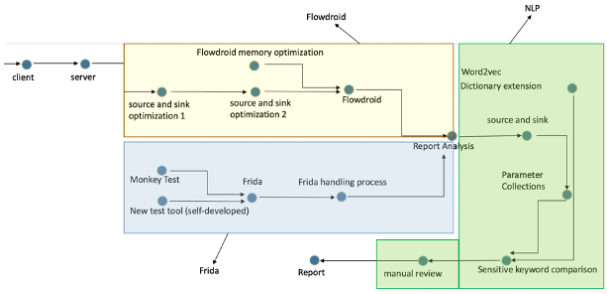

Fig. 1. Architecture of the Pipeline tool.

Fig.1 shows the framework of the pipeline tool.

In this research, I am mainly responsible for the development of Android client-side, the optimization of the detection tools Frida and Flowdroid (the configuration file: source and sink), and the writing of some script processing reports(logs) in the Flowdroid and NLP sections. Additionally, in the dynamic detection of Frida, we initially used Monkey Test for automated testing and used the dynamic detection tool Frida at the same time to collect user data in the automated testing process. However, because Monkey cannot fully meet our testing requirements, thus I wrote an automated testing tool to work with the Monkey to complete data collection with Frida.

II. Related Work

Commonly used dynamic/static detection tools usually judge whether there is data leakage in the application by analyzing whether there is data flow in the sources and sinks fucntions [7]. Figure 2 shows a basic data flow detection

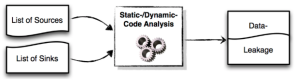

Fig. 2. Data Analysis with Sources and Sinks.

framework.

The Sources and Sinks determine the basic configuration of a dynamic/static detection tool. In the Android operating system, there are many sensitive sources and sinks [7]. The source method is responsible for reading user data, and the sink method is responsible for writing user data. The dynamic detection tool Frida can track the source data and hook it to determine whether there is data flow that has been maliciously written. The static detection tool FlowDroid can manually review the calls of the two methods to determine whether there is a data leakage.

Monkey is an automated process testing tool based on Android SDK [6]. The tool supports automatic installation by entering cmd commands in the terminate and automatic simulation of user clicks. It is a common testing tool that integrates installation and automatic testing.

AirTest is also an automated process testing tool. This tool is based on python scripts. It is a commercial testing software used to provide testing services for enterprise applications. The advantage of this tool is that it can completely simulate user clicks and automate all processes. The disadvantage is that it needs to write a corresponding test script for each application. Moreover, this tool needs to list all the clickable buttons in the script in advance to configure the testing.

As for the client development part, due to my past Android development experience, I won't introduce too much about this part of the work. The page is drawn using xml native coding, and the data flow is detailed in the HTTP network protocol and the Json packet protocol.

III. Method

A. Client-side Development

In the Client side, I use the native UI for page display, and no other third-party frameworks are applied. Fig.3 shows the homepage listing the application APKs that to be detected. Clicking the corresponding detect button will sent a HTTP request to transmit the relevant data to the server with a json packet format.

Fig.4 shows the HTTP connection flow of the CS architecture. The client first extracts the hashcode of the APK file to be tested. After it is sent to the server, the server will detect whether the hashcode exist in the database, if it exists, it will start the pipeline tool to detect and analyze the

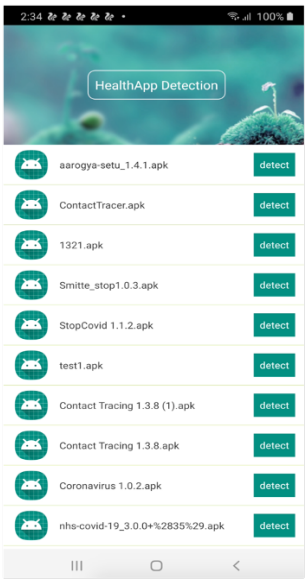

Fig. 3. Homepage of Android side of the Pipeline tool.

hashcode's corresponding file. Otherwise, it will notify the client-side to send the original APK file for detection. After the final detection is completed, the relevant report can be viewed through the client-side.

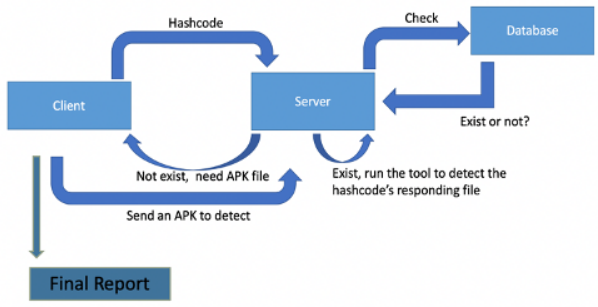

Fig. 4. CS Http-Connection Flow of the Pipeline tool.

B. Optimization of source and sink

Since I am familiar with the Android SDK, the first optimization of the source and sink is also done by me. Flowdroid provides an original configuration file. First of all, from the file I found that many source methods would not be called in Android, such as some packages like Java Servlet which is used in Java web development. Thus, these types of methods can be directly removed, otherwise these redundant methods will increase unnecessary detection costs. Secondly, some methods such as findViewbyId() are added. These methods read the user data of the component of the pages, which will definitely involve sensitive user information. Finally, I checked java open-source and integrated some methods. Such as a function A must call another function B, I only keep B in this configure file in order to reduce the

pressure of the detection tool.

C. Self-developed automated testing tool

In the dynamic testing of Frida part, initially we used Monkey for automated testing and Frida to monitor and track data flow information. However, in the usage process, I found that Monkey cannot participate in all events of the tested application. Each click of Monkey is a random coordinate point on the screen. Most of this coordinate point is not a clickable component, thus it has been simulated for thousands of times, with few effective clicks. At the same time, Monkey can not control the simulation clicks in the current application. In many cases, it will trigger the notification bar at the top of the screen, thereby clicking the system button, which also causes invalid clicks. Since Monkey cannot trigger all events, in most cases, it is impossible to detect whether the sink method is called, that is, whether user data is written out. From this situation, if an application is a definitely positive case but since the core functions cannot triggered so that the sensitive sink functions cannot be detection by the pipeline, we often regard it as a false negative case.

Considering these circumstances, I tried to find a more suitable tool for automated testing in Google, and accidentally found Airtest. Airtest is a commercial testing software used to provide testing services for enterprise applications. The advantage of this tool is that it can completely simulate user clicks and automate all processes. The disadvantage is that we need to write a corresponding test script for each application. In addition, the tool needs to list all the clickable buttons in the script in advance to complete the test. And due to the large number of applications we need to test, it is not realistic to write test scripts for each application. Based on this situation, I thought of an accessibility method in Android [8]. This function was originally designed for people with disabilities and is not often used for development. This function allows the program to traverse and find the components on the page. Thus, starting from this direction, I re-studied the API documentation of this function.

Fig. 5. Partial pages of an example application.

As shown in Fig.5, each page of the application can be regarded as an event node, then an application can be considered as an event graph. Fig.5 shows a partial pages of an example application. If I use DFS to traverse the graph, simulate clicks, and combine with the Android Intent function, we can regard a process as a single source path, and after we complete a process testing and jump to the homepage to directly test another process. This traversal method best preserves the integrity and continuity of the process, and we can ensure that all events will be triggered to obtain complete data flow information. In addition, this function is not limited to simulating click actions. What's more exciting is that we can identify the type of components and realize different operations for different components.

D. Scripts of information extraction

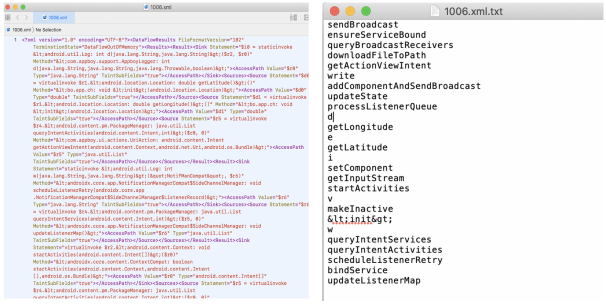

Fig. 6. Functions Extraction of FlowDroid.

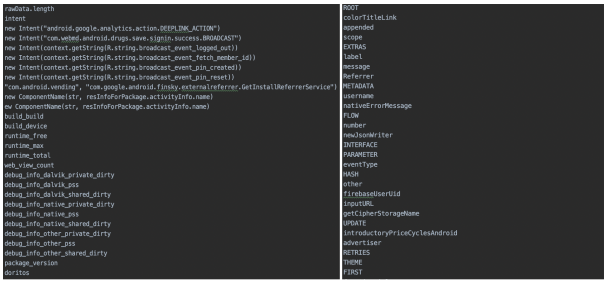

Fig. 7. Keywords Extraction of MobSF.

Regarding to the report results of Flowdroid and MobSF, I wrote a set of scripts to automatically extract key information. By observing the reports' features, I segmented the word in the report by symbol, and finally extracted the function list for the report generated for Flowdroid, and extracted the keyword list for the report generated for MobSF. The writing of this part of the script improves the efficiency of manual review. Since our pipeline tool tested a total of 72 APK files, I rewritten the script into a batch extraction tool. One-click startup script can extract relevant information for all reports and write the corresponding information into the corresponding file. Fig.6 and Fig.7 shows the extraction examples of Flowdroid and MobSF respectively.

IV. Experiments

Because of resource constraints, the CS architecture in this study currently only transmits data in the local area network. We place the client and server on the same network and use json packets to transmit data. But this part is not the purpose of our research. The aim of this study is on the result of comparing the optimized pipeline tool and the original tool to the data security detection and analysis of the application after the server receives the application data to be detected.

After optimizing Flowdroid's configuration file source and sink for the first time, we optimized the original 402 functions to 220, and then performed a comparative experiment on 72 APKs (before optimization and after optimization). Due to time constraints, we randomly selected 20 of the 72 reports for manual review and recorded the precision of the two experiments. Comparing the results, the precision of the original Flowdroid is 63%, while the precision of the FlowDroid after the first optimization is 72%, the results were shown in TABLE I.

TABLE I
Results of Flowdroid and first optimized Flowdroid

	Original	***First Optimization***
Source and Sink	402	220
Report	20	50
Precision	63%	72%

The automated tool I developed can complete automated testing of precise clicks and partial processes. For this section of the research, I have shown an automated test demo in my presentation on the date of 11.6 this semester, in which all clicks are automatically simulated, and the data in the edit box is automatically input. Regarding the inputting data, I use a predefined strategy.

Currently the automation tool has not been fully completed and optimized, so in the Frida dynamic detection section, our experiment still needs to be combined with Monkey for automatic testing. However, such a strategy currently does not completely trigger all events in the application, which also limits the final results of the current experiment.

According to the final experimental results, the pipeline tool we have developed has a performance of 85% for application data leakage detection, which detects some sensitive information, like GPS, Tracker, device Info and some private data respectively, the results were shown in TABLE II. Comparing the results with the performance of the detection tool Flowdroid which is 63%, it achieved about a 22% precision improvement.

TABLE II
Results of Flowdroid and the Pipeline tool

Sensitive information	***Frequency***
GPS(Location)	35%
Tracker	15%
Device Info(WIFI, IP, Country, etc.)	15%
Other Data Leakage(user id, Calendar)	5%

V. Conclusion and Future Work

In this research, our team proposed a pipeline detection tool to analyze data security issues for Android Health applications. During this research period, I am committed to the development of android client and the configuration optimization of some detection tools. In addition, I proposed a new automatic process testing tool, which can currently complete the testing of some process. In the future, I hope to continuously improve the automatic process testing tool from the following aspects:

- After completing a process test, store the visited event nodes and record them in a file to prevent the program from being abnormally crashed during the automatic test. Then we can use the file to recover the test process instantly.
- Automatically fill in the input box data. At present, in the data filling part, I experimentally define the scheme in advance. I hope to use machine learning to classify edit box names, such as 'name'/'account' corresponds to string, 'number'/'amount' corresponds to digit, etc.
- Identify more control types and perform corresponding simulation behaviors. For example, extended click to ListView, Gallery, processing of pictures and maps and other components can simulate zooming, long press and so on.

References

[1] buildfire, Ultimate Mobile App Stores List (2019), viewed 5 Aug 2020, https://buildfire.com/mobile-app-stores-list/.

[2] Chen S, Fang L, Meng G, Su T, Xue M, Xue Y, Liu Y, & Xu L 2020, 'An Empirical Assessment of Security Risks of Global Android Banking Apps', Association for Computing Machinery.

[3] StevenArzt, FlowDroid 2020, viewed by 8 Aug 2020, https://github.com/secure-software-engineering/FlowDroid.

[4] oleavr, Frida 2020, viewed by 8 Aug 2020, https://github.com/frida/frida.

[5] ajinabraham, Mobile-Security-Framework-MobSF 2020, viewed by 9 Aug 2020, https://github.com/MobSF/Mobile- Security- Framework-MobSF.

[6] developers, monkeyrunner, viewed 6 Aug 2020, https://developer.android.com/studio/test/monkeyrunner.

[7] Secure Software Engineering , SuSi – Sources and Sinks 2013, viewed 10 Aug 2020, https://blogs.uni-paderborn.de/sse/tools/susi/.

[8] developers, android.view.accessibility, viewed 15 Sep 2020, https://developer.android.com/reference/android/view/accessibility/package-summary.

[9] Pooryousef S & Amini M, 2017, 'Enhancing Accuracy of Android Malware Detection using Intent Instrumentation', In Proceedings of the 3rd International Conference on Information Systems Security and Privacy, pages 380-388.

[10] Hussain M, Al-Haiqic A, Zaidan A.A, Kiah M., Iqbal S, Iqbal S. & Abdulnabi M, 2018, A security framework for mHealth apps on Android platform, Computers & Security, Vol. 75, June 2018, Pages 191-217.

[11] Yang Z & Yang M, "LeakMiner: Detect Information Leakage on Android with Static Taint Analysis," 2012 Third World Congress on Software Engineering, Wuhan, 2012, pp. 101-104, doi: 10.1109/WCSE.2012.26.