ARQUS Cybersecurity Summer School 2021

Kushagra Singh BISEN[1,2]

[1] Université Jean Monnet, Saint Etienne
[2] MINES Saint Etienne, Saint Etienne
kushagra.bisen@etu.univ-st-etienne.fr

Abstract. The document presents the solutions to the challenges given during the ARQUS Cybersecurity summer school held during 6th to 10th September 2021. The event was hosted by universities located in Italy, France, Lithuania and Spain. The event promoted collaboration between universities and promoted research across Europe. There are 4 different sections detailing each section, concluding with the skills learnt by myself during the course of 4 days.

Keywords: ARQUS · Cybersecurity · Summer-School

1 Introduction

2 Security in Android Applications

2.1 Introduction

Android smartphones are the most popular versions of the modern smartphone. There are around 3 billion android smartphones active across the world. By using a popular software, the risk of being exploited by unwanted users is tremendously huge. A malware can be used to control/exploit millions of users even if the malware's reach was not big. Cybersecurity experts, responsible for the security of users plan to tackle the issue in two different ways, 1) Ensuring illegal code practices are not followed. 2) Using analysis tools to check if the application being used is safe. During the first day of our security in android application challenge, we were advised to use two different tools to analyze the coding practices as well as the security of the application being used. Subsequent sections will describe the tools being used and present results from analysis of the tools.

What is Android? Android can be considered an extention to the existing Java JVM library with tools to develop applications. Android follows a layered architecture 5 with application layer containing the applications for the users. A user never goes further than the application layer, whereas the developers can exploit the other layers based upon the requirement. Since Android 6.0, Android provides the option for the user to decide if a particular application has the **permission** to use specific resources in the device. Malicious applications try to extract informations by using permissions they do not need but use for data

mining applications. Android assigns a specific Linux User ID to every application on installation. There are 130 different types of permissions an application can ask from the user. They are divided into categories, namely, 1) *Normal*, basic permissions which are automatically granted by Android. 2) *Dangerous*, permissions required to access important core APIs of Android. 3) *Signature*, permissions which are granted by the developer of the application itself. 4) *SignatureOrSystem*, permissions which granted to the system apps automatically.

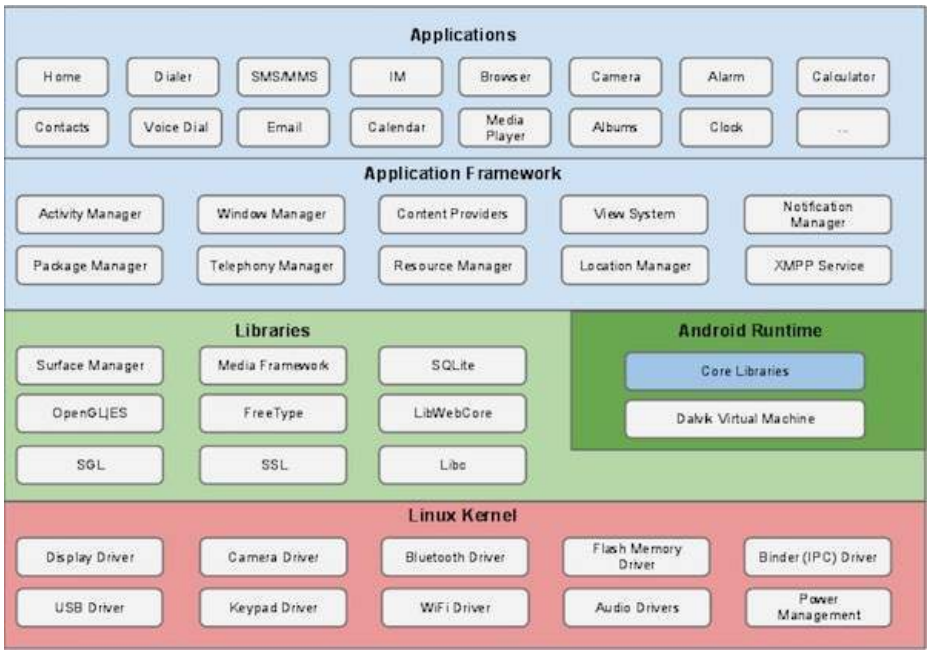

Fig. 1. The Android OS structure

2.2 RiskInDroid Analysis Tool

Analysis of the permissions are usually calculated empirically. An application is judged on a *risk index value* out of 100. Probabalistic risk index analysis approach for android application is usually a good way to analyze the safety of an application. RiskInDroid does quantitative risk analysis of android applications which are written in Java. It utilizes machine learning tools such as *scikit-learn* to generate a numeric risk index value between 0 and 100 for an application. Classifiers such as :

– Support Vector Machines (SVM)

– Multinomial Naive Bayes (MNB)
– Gradient Boosting (GB)
– Logistic Regression (LR)

are employed for classification.

RiskInDroid takes in consideration both the permissions declares into the applications manifest and the bytecode retreived from the application through reverse engineering. Static analysis is used to classify the permissions used in the application into four different ways :

– Declared permissions : which are extracted from the application manifest.
– Exploited permissions : declared and actually used in the bytecode.
– Ghost permissions : not declared but with usages in the bytecode.
– Useless permissions : they are declared but not used in the bytecode.

The risk value generated is considered to be accurate as it is trained upon 6000 malware samples and 112000 applicaitons of varying levels of security.

Analysis To execute the analysis, we are using :

– Ubuntu 20.04 LTS
– Docker
– RiskInDroid Application
– Xiaomi Home Application for Analyis

Instructions :

– git clone https://github.com/ClaudiuGeorgiu/RiskInDroid.git
– docker build -t riskindroid .
– docker run –rm -p 8080:80 riskindroid

The resulting RiskInDroid application is now running at localhost:8080. 2

Results The Xiaomi application had a score of 30.34. 3. The results related to permissions were :

– Declared - 58 permissions.
– Required and Used - 21 permissions.
– Required but Not Used - 37 permissions.
– Not required but Used - 4 permissions.

Developers can use this to analyze the permissions to optimize the use, and thus improving the risk index.

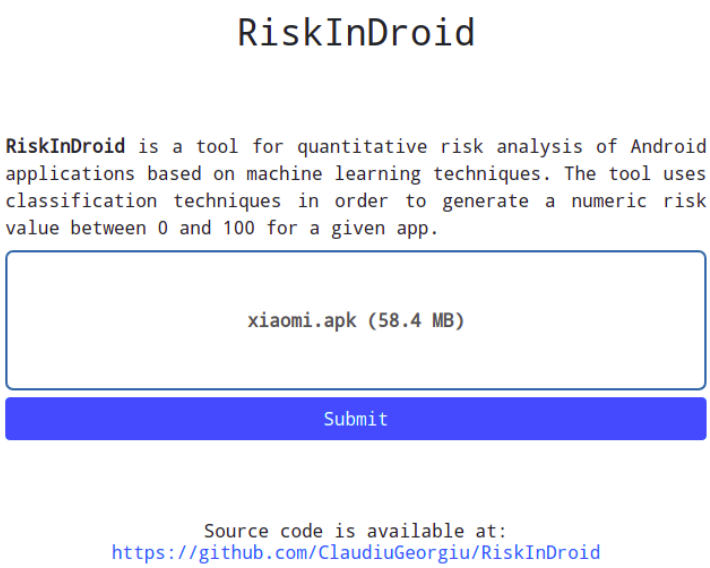

Fig. 2. RiskInDroid Application

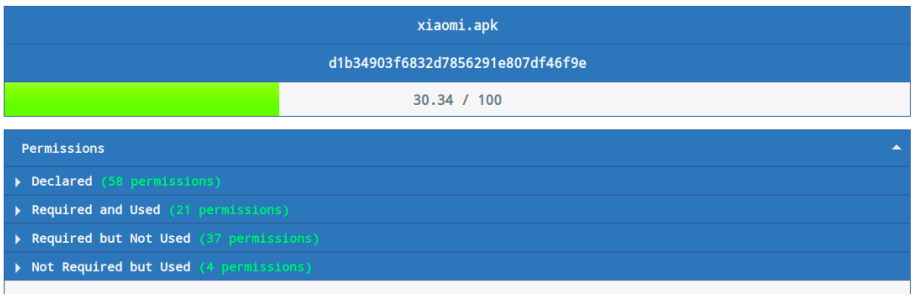

Fig. 3. RiskInDroid Application - Xiaomi Analysis

2.3 SPECK Tool

As we saw in the previous section, analysis of an application can be done with permissions being provided to the android application. One more way to analyze the application is by using the code written to develop the application. SPECK is a tool designed to search for bad/malicious coding practices in the android application. The software declares a specific set of rules to ensure the application's security.

The many rules divide the intensity of the flaw into three levels :

- **INFO** : if a good coding practice is detected.
- **WARNING** : if there is an security issue which can be dangerous.
- **CRITICAL** : informs that there is a confirmed security issue.

To analyze an android application, we use the tool in a developer mode. There are 32 different rules which will be used to analyze the coding practices of the android application. The tools are based upon, 1) Best Practices 2) Security Tips 3) SSL Security 4) Configuration Security 5) Crytography and 6) Direct Boot

The SPECK tool is downloaded and compiled. To analyze the Xiaomi application, we write *python3 Scan.py -s path-to-xiaomi-app* Manifest and the main java file in analyzed. The application analyzed **1719** files for the first rule. In the 4, you see the analysis with Rule 8,9,10 which are :

- Rule 8 : Store private data within internal storage
- Rule 9 : Share data securely across apps
- Rule 10 : Use scoped directory access

A developer can read the output according to the different rules to improve the security of the application.

3 Buffer Overflow in Cybersecurity

4 UNIX commands

5 Side Channel Analyis with Deep Learning

6 Conclusion

7 First Section

7.1 A Subsection Sample

Please note that the first paragraph of a section or subsection is not indented. The first paragraph that follows a table, figure, equation etc. does not need an indent, either.

Subsequent paragraphs, however, are indented.

Fig. 4. SPECK Result - Xiaomi Analysis

Sample Heading (Third Level) Only two levels of headings should be numbered. Lower level headings remain unnumbered; they are formatted as run-in headings.

Sample Heading (Fourth Level) The contribution should contain no more than four levels of headings. Table 1 gives a summary of all heading levels.

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

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

Displayed equations are centered and set on a separate line.

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

Please try to avoid rasterized images for line-art diagrams and schemas. Whenever possible, use vector graphics instead (see Fig. 5).

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

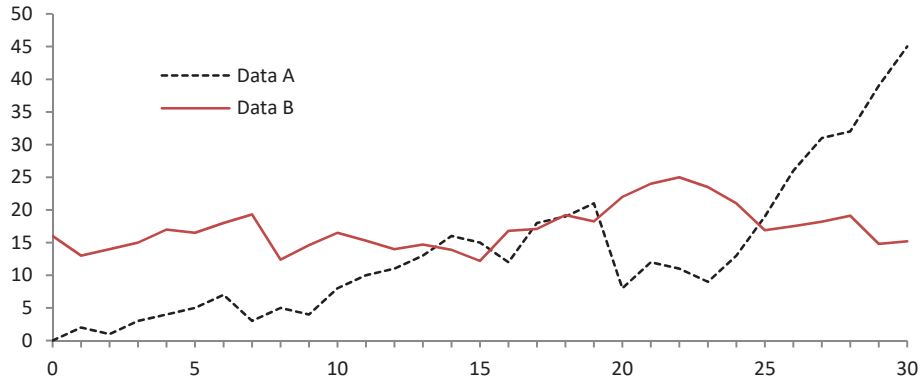

Fig. 5. A figure caption is always placed below the illustration. Please note that short captions are centered, while long ones are justified by the macro package automatically.

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

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

References

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