RANKING OF ANDROID APPS BASED ON SECURITY EVIDENCES

by

Ayush Maharjan

A Thesis

Submitted to the Faculty of Purdue University
In Partial Fulfillment of the Requirements for the degree of

Master of Science

Department of Computer and Information Science at IUPUI
Indianapolis, Indiana
December 2020

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Rajeev R. Raje, Chair
Department of Computer and Information Science

Dr. Mihran Tuceryan
Department of Computer and Information Science

Dr. Xukai Zou
Department of Computer and Information Science

Approved by:
Dr. Shiaofen Fang

Dedicated to my parents

ACKNOWLEDGMENTS

I cannot begin to express my deepest gratitude to my advisor Dr. Rajeev Raje for his guidance and supervision throughout my research. This research would not have been successful without his invaluable insights and encouragement. I would also like to thank my committee members Dr. Xukai Zou and Dr. Mihran Tuceryan for their support to my research.

I would like to extend my sincere thanks to my friend Nahida Sultana Chowdhury for valuable suggestions and ideas towards my research. It was an immense help sharing different ideas and thoughts with her for this research. I would also like to thank her for sharing the datasets that were used in the research.

I would also like to acknowledge the assistance of Nicole Wittlief and the Department of Computer Science throughout my time here at IUPUI. I would also like thank the Office of International Affairs and the Graduate Office for their assistance in working towards my degree.

I am extremely grateful to my parents for their support in pursuing my degree and encouragement throughout my research work. I am deeply indebted to my friend Barsa, my brother Aswin, and my cousins Sagun and Amita for the encouragement and motivation that they have provided throughout my work. Lastly, I would also like to thank all my friends and family who supported me in my journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

With the large number of Android apps available in app stores such as Google Play, it has become increasingly challenging to choose among the apps. The users generally select the apps based on the ratings and reviews of other users, or the recommendations from the app store. But it is very important to take the security into consideration while choosing an app with the increasing security and privacy concerns with mobile apps. This thesis proposes different ranking schemes for Android apps based on security apps evaluated from the static code analysis tools that are available. It proposes the ranking schemes based on the categories of evidences reported by the tools, based on the frequency of each category, and based on the severity of each evidence. The evidences are gathered, and rankings are generated based on the theory of Subjective Logic. In addition to these ranking schemes, the tools are themselves evaluated against the Ghera benchmark. Finally, this work proposes two additional schemes to combine the evidences from difference tools to provide a combined ranking.

CHAPTER 1. INTRODUCTION

There has been a significant increase in the usage of smartphones in the past decade [1]. Google's Android OS and Apple's iOS have been the two most popular operating systems that have been most widely used for the smartphone devices. Android OS dominates the smartphone market share, running in more than 70% of all mobile devices [2]. This popularity of smartphones has also led to development of large number of apps that run on these devices. As of November 2020, there are over 3 million Android Apps published in Google Play, the official marketplace for Android Apps [3].

In this competitive market, the users of these mobile devices have the flexibility of selecting apps for their particular needs from a large variety of apps. People generally rely on the app ratings and reviews, and the features mentioned in the description of the apps on the app store for selecting apps for their use. The app ratings and reviews alone can sometimes be biased and may not be a true reflection of the quality of the apps. Security is one of the most important internal metrics which should be considered during the selection of apps. This thesis proposes a way of ranking and selecting apps based on the security evidences gathered through the static code analysis of the apps.

The types of security risks posed by mobile apps are quite different from the risks involved with desktop or Web software. Most of the mobile applications rely on user data and constantly communicate through network with remote servers and devices. It is important to make sure that the data is protected within the device as well as when it is being transmitted over a communication channel such as WiFi, Bluetooth, NFC, etc. With the advancement of smartphones, people have become more dependent on such devices. Many apps use important personal data of the users (such as their photos, location, personal messages, etc.), which makes the security of data even more important. Hence, it is important to take the security of the applications into consideration while downloading an app. We are considering these security issues for ranking Android apps so that the users can consider them while downloading an app for their usage.

There have been numerous efforts on identifying and categorizing the security issues in mobile apps. Several open source and commercial static analysis tools are available that can detect security issues in the applications. These tools are generally used during development to detect and fix security and privacy issues. We use these tools as the sources of evidence to rank apps.

One of the challenges that needs to be considered, while collecting security-related evidences is that the issues detected by the static analysis tools are not all equivalent. Hence, we need to consider issues such as the frequency, severity, impact, etc., of the issues to get a better ranking of similar apps. There are standards such as OWASP Mobile Top Ten [4] and Common Weakness Enumeration (CWE) in Mobile Applications [5] that categorizes the security issues into different levels of severity. While these standards categorize the security vulnerabilities, each category within the standards can cover a broad number of vulnerabilities that can have different severity. During the security analysis, another standard metric that considers different issues is the Common Vulnerability Security Score (CVSS). CVSS is a metric that calculates a security score based on factors such as the type of attack, its severity, complexity of exploiting the vulnerability, and the impact of the vulnerability in terms of confidentiality, availability and integrity. In the research, the issues being detected are classified and scored using the CVSS and integrated into our ranking methodology.

Another challenge is that the tools may themselves not be reliable. Most of the static analysis tools are known to report false positives [6]. It is difficult to consider each issue when we are running static analysis on large number of real-world apps. This can be overcome by using a sample set of benchmark apps to analyze the reliability of the tools. This research uses Ghera android vulnerabilities benchmark [7] to benchmark each of the selected static analysis tools, and based on the performance on the benchmark, we have assigned them a reputation score. The Ghera benchmark is chosen over other available benchmarks due several factors including the number of issues reported across different categories, the nature of the issues reported, and the format of the benchmark apps are organized.

Finally, the degree of our belief (our trust) that an app is secure needs to be quantified. Subjective Logic [8] is used to represent the trust of different app based on the evidences generated by the static analysis tools. Subjective Logic allows us to represent an opinion about a proposition (in our research, the proposition being that the app is secure) using a tuple of belief, disbelief and uncertainty. Subjective Logic is used because it incorporates a degree of uncertainty unlike traditional binary logic. It also provides several operators that can be used to combine opinions. These operators are useful in calculating the trust of an app and combining the opinion of multiple static analysis tools.

In this research, we have proposed three basic ranking schemes that consider different factors for calculating the ranks. The first scheme uses the different categories of vulnerabilities present in the apps. The second scheme considers the frequency of vulnerabilities in each category. The third scheme considers the severity of each vulnerability based on the CVSS. We can use these three schemes to calculate ranks using each of the selected tools separately. We also provide two additional schemes to combine the ranks from the selected tools. Among the two schemes that combine the opinions of different tools, the first one considers each tool equally whereas the second tools assign a weight to each tool based on their performance in the benchmark.

The analysis is performed using three different static analysis tools. Our dataset contains 175 apps across 8 different categories. Five of the categories (finance, insurance, news, shopping, and travel) consist of 5 apps each. These five categories help in performing a detailed analysis of our ranking schemes. Three categories (games, photography, and tools) contain 50 apps in each category. We have used these three categories to validate our ranking schemes in a larger data set.

Any ranking of app is subjective in nature and depends on various factors that are taken into consideration and the weightage given those factors. There is no single ground truth that we can rely to measure the correctness of any ranking method. The ranking schemes that we have proposed in this research provide a justifiable ranking scheme based on the security evidences from different static analysis tools and provide a basis for similar research in the future.

1.1 Objective

The goal of the thesis is to analyze the security vulnerabilities present in android applications and develop schemes for ranking the applications based on the presence of security vulnerabilities.

1.2 Contribution

1. This thesis proposes three basic ranking schemes based on empirically collected security evidences using Subjective Logic, and two additional schemes to combine rankings from different static analysis tools from the basic schemes.
2. This research performs an empirical analysis and obtain the CVSS scores for the evidences detected by the static code analysis tools used in the experiments.

3. This effort researches common benchmarks and analyzes the reliability of the static code analysis tools for integrating them into the ranking scheme.
4. Finally, this work employs the proposed schemes on Apps collected from the Google Play (the official marketplace for Android apps).

CHAPTER 2. RELATED WORK AND BACKGROUND

This chapter discusses the background theory of the related concepts and describes prominent research efforts associated with the work for this research.

2.1 Key Concepts related to Android

There are several key concepts related to Android Apps that are useful while analyzing the security of the Android applications.

2.1.1 Android Components

An android app is comprised of four different types of components [9]:

1. Activities [10]: An Activity provides the user interface for the apps. It usually represents a single screen in the app. It displays the views to the users and handles user interactions. Activities also allow users to navigate to other activities.
2. Services [11]: Services is a component that allows handling of background tasks. It does not provide any user interface. Services can be used for tasks such as playing music in the background, syncing data over network, etc. Services are of three types:
    a. Foreground Services: These services are noticeable to the users and must display a notification. An example of foreground service is an audio app playing music in the background.
    b. Background Services: These services perform operations in the background without any notification to the users.
    c. Bound Services: These services are tied to other components and can exchange data with the components it is bounded to.
3. Broadcast Receivers [12]: Broadcast Receivers provide a mechanism for different events to be delivered outside the regular flow of the apps. For example, we can use broadcast receiver to listen for a broadcast announcing that the battery is low, and the receiver gets triggered even when the app is not running.

4. Content Providers [13]: Content providers manage the app data stored in any persistence mechanism. Other apps can query or modify the data through the content providers with appropriate permissions.

The activities, services, and broadcast receivers are activated using messages called Intents [14]. Intents are used to build components together at runtime. Intents can also contain data that is passed onto the components. Intents are of two types:

1. Explicit Intents: It specifies the target application's package name or a fully qualified component class name to trigger a component.
2. Implicit Intents: Implicit Intents use intent filters to identify the action to be performed, and the components that can satisfy the action are triggered to handle it. Implicit intents are generally used to allow communication between different apps.

Content Providers on the other hand are triggered using Content Resolvers [15]. The content resolvers provide CRUD (create, read, update, delete) methods for the data source defined in the content providers. The content providers are identified using a simple URI that uses the content:// schema.

2.1.2 **Permissions**

Permissions [16] are a security mechanism provided to protect the privacy of the users. Each app runs in its own sandbox environment and requires permissions to interact with the system resources and other app. Apps must declare permissions to use sensitive data such as performing network access, sending SMS, reading or writing the user's private data, etc. Permissions are declared in the manifest file.

There are three different protection levels of permissions [16]:

1. Normal Permissions: These permissions are general permissions that do not use private data of the users.
2. Signature Permissions: These permissions are granted only to the apps that are signed by the same certificate.
3. Dangerous Permissions: Dangerous permissions are required to access data or resources that involve user's private information.

Before Android 5.1.1, permissions were requested during installation, but from Android 6.0 onwards dangerous permissions are requested only in the runtime with the normal permission being requested at the installation time.

It is recommended to declare only the permissions that are really required by an app. For runtime permissions, it is also recommended to provide the context to why the permission is being asked for.

There are several pre-existing permissions defined in the android system. It is also possible to define custom permissions to restrict access to resources provided by the app.

2.1.3 **App Resources**

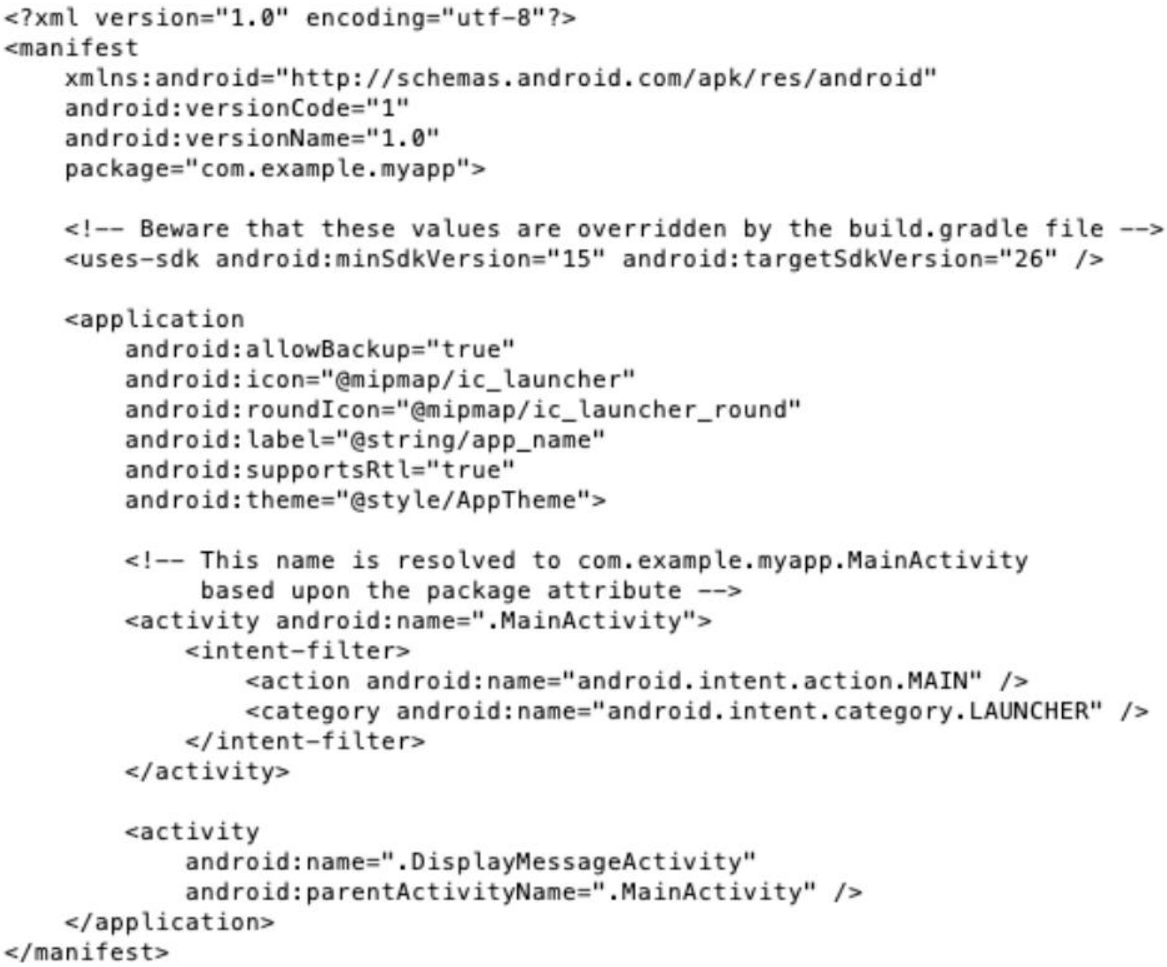
Resources such as images, audio files, layouts, styles, etc. are saved in the res folder. A unique identifier is generated for each of the resource by the SDK, and this unique code is accessible in the source code.

2.1.4 **The Manifest File**

The AndroidManifest.xml file is an XML file present in all applications that allows the system to know the details of the app. It declares all the components along with the application details such as the application name, package name, version code, version name, minSdk and targetSdk versions, permissions, etc.

Permissions can be declared using the <uses-permission> tag. The components can be declared using the following tags inside the <application> tag:

- <activity> for Activities
- <service> for Services
- <provider> for Content Providers
- <receiver> for Broadcast Receivers



```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="1"
    android:versionName="1.0"
    package="com.example.myapp">

    <!-- Beware that these values are overridden by the build.gradle file -->
    <uses-sdk android:minSdkVersion="15" android:targetSdkVersion="26" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <!-- This name is resolved to com.example.myapp.MainActivity
             based upon the package attribute -->
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity
            android:name=".DisplayMessageActivity"
            android:parentActivityName=".MainActivity" />
    </application>
</manifest>
```

Figure 1. Example of Android Manifest File (reproduced verbatim from [17])

An example of manifest from the android developer guide [17] is shown in Figure 1. The manifest declares two activities: MainActivity (which is the launcher activity that is launched when the app is opened) and the DisplayMessageActivity. We can also see the metadata related to the app in the manifest.

2.1.5 APK File

All the compiled code of an android app, along with data and resource files, is packaged as an archive file called the Android package (APK). The file has an extension ".apk".

The compiled code is in Dalvik Executable (dex) format. Tools such as dex2jar [18] and jadx [19] convert the dex to Java Archive (jar) format, so that the compiled Java classes can be analyzed using readily available code analysis tools.

2.2 Mobile Application Security

The security challenges for Mobile Apps are different from the security challenges posed by traditional desktop apps and Web apps. This can be primarily attributed to three factors:

1. The architecture of the mobile platforms such as Android and iOS provide sets of APIs that are significantly different from that of desktop or laptop operating systems. The platforms themselves take into consideration a lot of factors to make the mobile apps secure. The platforms use sandboxing environment to prevent mobile apps from causing harm to the device or other apps. They provide APIs to program the apps. Special APIs are provided for Inter-Process Communication (IPC). To use features such as Location and Internet, a permission model is provided. It is important the applications follow best security practices to avoid any leaks of sensitive information.
2. The purpose, pattern and frequency of the usage of mobile apps are also considerably different than those of desktop or Web apps. The users use the applications for various tasks to make their day-to-day lives easier. People use apps for managing their navigation, finance, education, healthcare, and many other aspects of their lives rather than just managing their contacts and phone calls. This means that the apps handle a lot of personal data such as location, financial data, and health data. The security challenges are further amplified by the fact that the mobile apps generally store these data locally as well as back it up in some remote server or other trusted endpoints. Thus, proper security measures should be taken to store the data in device, and while communicating these data with trusted endpoints.
3. The apps are usually packaged into an archived file (for example, APK for Android, and IPA for iOS). The source code is archived into these files which makes it easier for malicious users can decompile these archived files and exploit vulnerabilities present in the apps. They can also tamper with the archived file and send it users. The platform themselves provide mechanisms to prevent such attacks. These archived files are generally signed with the developer’s certificate and the users should only download the app from secure and well-known app stores. The apps need to ensure that they are difficult to tamper with or reverse engineer by using proper security measures such as code obfuscation.

Due to these differences in the vulnerabilities of mobile apps, the Open Web Application Security Project (OWASP) has a separate Mobile Security Project. They have identified a separate top ten categories of mobile vulnerabilities published in 2016 [4]. A summary of the top ten categories are as follows:

- M1 Improper Platform Usage: These vulnerabilities are due to misuse of platform APIs and the failure of using proper security controls offered by them.
- M2 Insecure Data Storage: Issues related to insecure data storage and unintended data leakage are covered under this category.
- M3 Insecure Communications: These security issues arise when the data being communicated is not secured when communicating with trusted endpoints.
- M4 Insecure Authentication: This category covers issues when the authentication and session management of end users is done poorly.
- M5 Insufficient Cryptography: These issues arise when cryptography is applied to sensitive data being stored or communicated, but the cryptography being implemented itself does not fulfil the required security standards.
- M6 Insecure Authorization: Issues with the authorization of users to use different features of the app are included in this category.
- M7 Client Code Quality: This category includes the code level problems such as buffer overflow and memory leaks.
- M8 Code Tampering: Making changes to the resources or code in the app binary or any other form of modifying the original app is covered under this category.
- M9 Reverse Engineering: Mobile apps can be reverse engineered, and its source code, assets and other resources can be exploited to extract sensitive information such as cryptography information, backend endpoints and intellectual property.
- M10 Extraneous Functionality: Unintended or malicious backdoor functionalities or other internal developmental issues that can lead to security exploits comprise this category.

These categories can be utilized to calculate the priority of different categories of findings in our research. The relevance of the top ten category decreases from M1 down to M10. But even within each of the ten categories, the findings can have a different impact and priority. One of the ways to quantify these aspects is the Common Vulnerability Scoring System (CVSS). [4]

2.3 Common Vulnerability Scoring System (CVSS)

The Common Vulnerability Scoring System (CVSS) is a specification maintained by the Forum of Incident Response and Security Teams (FIRST) that assigns a numerical value to the security vulnerabilities based on their severity. The CVSS score can be used to define the criticality of the vulnerabilities that are found in the apps.

This research uses version 3.0 of CVSS specification. The CVSS specification calculates a score based on three metric groups [20]:

- Basic Metric Group
- Temporal Metric Group
- Environmental Metric Group.

The Basic Metric Group can be further classified into Exploitability Metrics that defines the characteristics of the vulnerable component, the Scope that defines the extent to which the vulnerability can impact resources beyond its privileges, and the Impact metrics that defines the impact on the confidentiality, availability and integrity of the component. [20]

The Exploitability Metrics consists of the following sub-metrics [20]:

- Attack Vector (AV): The attack vector defines the context through which the vulnerability can be exploited. It can consist of the following values:
    - Network (N): The vulnerability is exposed through the network stack and the attacker's path is through the OSI layer 3.
    - Adjacent (A): The vulnerability is exposed through the network stack but is limited to the same shared physical or logical network and cannot be performed across the OSI layer 3 boundary.
    - Local (L): The vulnerability is local and is not exposed through the network stack. The attacker's path is through the read/write/execute capabilities.
    - Physical(P): The vulnerability requires a physical access to the device being exploited.
- Attack Complexity (AC): This metric defines the conditions beyond the attacker's control that must be present to exploit the vulnerability. The metric may have a value of low(L) or high(H) depending on whether special conditions that requires a measurable amount of preparation or execution is required for the exploitation of the vulnerability.

- Privileges Required (PR): This metric defines the level of privilege required to exploit the vulnerability. It’s value is None (N) if the attacker does not require any authorization; Low (L) if the attacker requires basic privileges that could only affect the settings and files owned by users; or High (H) if significant privileges that could affect component-wide settings and files are required.
- User Interaction (UI): This metric can have values None(N) or Required(R) depending on whether the exploitation of the vulnerability requires some interaction from the user.

The scope (S) defines whether the vulnerability in one software component can lead to changes across the other components in the system. If the attack only affects the authorized scope, its value is Unchanged (U). If the vulnerability affects resources beyond the authorized scope of the component being exploited, its scope is Changed (C) [20].

The Impact Metrics are defined as follows [20]:

- Confidentiality Impact (C): It measures the impact to the confidentiality of information being managed by the component caused by the exploitation of the vulnerability. If total confidentiality is lost, the values is High(H). If some of the information are exposed by the vulnerability but the attacker does not have control over what information is obtained, it should be categorized as Low(L). If there is no loss of confidentiality, the metric should have value None(N).
- Integrity Impact (I): It measures the impact on the integrity of the information being managed by the component caused by the exploitation of the vulnerability. It should have High(H) value if there is total loss of integrity, Low(L) if there is a limited amount of modification that can be caused by the attacker, or None(N) if there is no loss of integrity.
- Availability Metric (A): It measures the impact on availability of the component caused by the vulnerability. If the attack can cause total denial of access to resources, the vulnerability can be categorized as High (H). If the attack is causes reduced performance or interrupts availability temporarily, it can be classified as Low (L). If there is no impact on the availability of the system, the metric can be assigned value None (N).

The Temporal and Environmental Metric groups are optional. The temporal metrics define the current state of the exploit technique (Exploit Code Maturity), existence of patches or workarounds (Remediation Level) and confidence on the description of vulnerability (Report Confidence). The environmental metrics define the importance of Confidentiality, Integrity and Availability Requirements of the affected component to the organization. [20]

The CVSS metrics can be textually represented in the form of a vector string. The following table (extracted from Table 15 in [20]) from the CVSS specification denotes the symbols used for the vector string for the base metrics (the symbols for temporal and environmental metrics can be found in the specification):

Table 1. CVSS Vector String Base Metric Values

Metric	Values (V)
Attack Vector, AV	[N, A, L, P]
Attack Complexity, AC	[L, H]
Privileges Required, PR	[N, L, H]
User Interaction, UI	[N, R]
Scope, S	[U, C]
Confidentiality, C	[H, L, N]
Integrity, I	[H, L, N]
Availability, A	[H, L, N]

A CVSS vector string is of the form [20]:

$$CVSS:[Version]/AV:[V]/AC:[V]/PR:[V]/UI:[V]/S:[V]/C:[V]/I:[V]/A:[V]$$

An example of the CVSS vector string for a vulnerability with a Physical Attack Vector, Low Attack Complexity, None Privilege Required, Required User Interaction, Unchanged Scope, Low Confidentiality Impact, High Integrity Impact and None Availability Impact is:

$$CVSS:3.0/AV:P/AC:L/PR:N/UI:R/S:U/C:L/I:H/A:N$$

The metrics in the vector string can be in any order according to the specification. This format from the specification [20] has been used throughout this document for representing the CVSS metrics of a vulnerability.

The National Vulnerabilities Database (NVD) [21] is one of the popular vulnerabilities databases that used the CVSS as a metric for the vulnerabilities reported in the database. The NVD is a reliable source of information that we have used for the CVSS score for several vulnerabilities in this research.

For the security vulnerabilities not available in the database, the CVSS vector can be determined by analyzing the vulnerability and the CVSS score can be calculated using the formula described in the following section.

2.3.1 CVSS Calculation

Given a CVSS vector, the CVSS score can be calculated using the CVSS Calculator [22]. The calculation is based on the numerical values and the formulas defined in the CVSS specification. There are formulas defined for the temporal and environmental metrics in the specification. This study uses only the base metrics for simplicity.

In the specification [20], the Base Score is defined as a function of Impact Sub-Score (ISC) and Exploitability Sub-Score (ESC).

The ISC is calculated as follows [20]:

$$ISC = \begin{cases} 6.42 * ISC_{Base} \quad (if\ scope\ unchanged) \\ 7.52 * \ [ISC_{Base} - 0.029] - \ 3.25 * [ISC_{Base} - 0.02]^{15}\ (if\ scope\ changed) \end{cases}$$

Where,

$$ISC_{Base} = 1 - [\left(1 - Impact_{Conf}\right) * \ \left(1 - Impact_{Integ}\right) * \ (1 - Impact_{Avail})$$

The ESC is calculated as follows:

$$ESC = 8.22 * Attack\ Vector * Privilege\ Required * User\ Interaction$$

The Base Score is calculated as follows [20]:

$$Base\ Score = \begin{cases} 0\ [if\ ISC \leq 0] \\ round(\min(ISC + ESC, 10))\ [if\ \ ISC > 0\ and\ Scope\ Unchanged] \\ round(\min(1.08 * (ISC + ESC), 10))[if\ ISC > 0\ and\ Scope\ Changed] \end{cases}$$

The metric values (from the Table 16 of the specification [20]) can be found in table below:

Table 2. CVSS Metric Values

Metric	Metric Value	Numerical Value
Attack Vector	Network Adjacent Network Local Physical	0.85 0.62 0.55 0.2
Attack Complexity	Low High	0.77 0.44
Privilege Required	None Low High	0.85 0.62 (0.68 if Scope is Changed) 0.27 (0.50 if Scope is Changed)
User Interaction	None Required	0.85 0.62
$Impact_{Conf}$, $Impact_{Integ}$, $Impact_{Avail}$	High Low None	0.56 0.22 0

These metric values are used later during the calculation of CVSS for the issues detected by the tools.

2.4 Security Vulnerability Analysis

There have been numerous efforts for detection of vulnerabilities in Android apps. A lot of research has been done be security experts, and different tools have been developed for detecting vulnerabilities in the Android applications.

There are three major categories of tools for detecting security vulnerabilities:

1. Static Code Analysis Tools: These tools analyze the bytecode or the source code of the apps without executing the code. These tools generally perform quick analysis, but due to lack of visibility of data, they may contain large number of false positives. Qark [23], Androbugs [24], MobSF [25], and JAADAS [26] are some of the generic static open source tools available to be used for static analysis of Android apps.
2. Static Taint Analysis Tools: Taint Analysis Tools are specialized static analysis tools that detect information leakage. They track every data from every possible source all the way through to the sinks (where the data is used). These tools are specialized at finding the data leakages, but cannot detect other kinds of security vulnerabilities such as vulnerabilities related to the SSL, Web, permissions, etc. Flowdroid [27], DroidSafe [28], and Amandroid [29] are some examples of static analysis that are used for taint analysis of Android apps.
3. Dynamic Code Analysis Tools: Dynamic Analysis Tools run the analysis during or after execution of the code. They are capable of providing much more visibility and details to the vulnerabilities being detected, but they take a longer time to execute and can miss the vulnerabilities that were not triggered during the execution of the code. MobSF [25], in addition to static analysis, is also able to perform the Dynamic Code Analysis of Android apps.

2.4.1 **Benchmarking of Tools**

Besides development of tools, there have been efforts to analyze the effectiveness of the tools, and several benchmarks have been created for such analysis. The various benchmarks that were considered are discussed below.

The Ghera Android Vulnerabilities benchmark [7] provides source code for the benign app, malicious app and the secure app for each benchmark. The Benign app is the version of the application that exhibits the vulnerability, the malicious app is the application that exploits the vulnerability in the benign app, and the secure app is the application with the security vulnerability removed from the benign app. Each benchmark also provides a summary of the vulnerability along with the affected Android versions and description and example of the vulnerability being demonstrated through the benign and the secure apps. The Ghera benchmark has a total of 60 benchmarks categorized into seven groups – Cryptography, Inter-Component Communication (ICC), Networking, Non-API, Permission, Storage, System, and Web.

Damn Insecure and Vulnerable App (DIVA) [28] is an app that contains insecure and vulnerable code. It was originally intended as a learning tool for Android developers to understand different security vulnerabilities, but it has been used by security professionals for penetration testing. It includes various challenges such as insecure logging, hardcoding issues, insecure data storage, input validation issues, access control issues, etc. [28]

Purposefully Insecure and Vulnerable Android Application (PIVAA) [29] is another insecure and vulnerable app that was designed as an improvement over the outdated DIVA. It covers the following vulnerabilities [29]:

- Usage of Weak Initialization vector
- Possible Man-In-The-Middle (MITM) Attack
- Remote URL load in WebView
- Object Deserialization on Untrusted Resource
- User-Supplied Input in SQL queries
- Missing Tapjacking Protection
- Enabled Application Backup
- Enabled Debug Mode
- Weak Encryption

- Hardcoded Encryption Keys
- Dynamic Code Loading
- Creation of World Readable or Writable Files
- Usage of Unencrypted HTTP Protocol
- Weak Hashing Algorithms
- Predictable Random Number Generation
- Exported Android Components
- JS enabled in a WebView
- Temporary File Creation
- Hardcoded Insecure Data
- Untrusted Certificate Authorities (CA) Acceptance
- Usage of banned API functions
- Self-Signed Certificate Authority (CA) enabled in Webview
- Cleartext SQLite Database

DroidBench [30] is a micro-benchmark designed to evaluate the effectiveness of Android taint-analysis tools. It comprises of 120 test cases for data leakage in Android apps. The test cases cover the leakages related to Java (such as Arrays and Lists, Callbacks, Reflection) and Android APIs (such as Lifecycle, Inter-App Communication, Inter-Component Communication).

ICC-Bench [31] is a more specialized repository of benchmark apps focused towards Inter-Component data leakage in Android apps. It consists of 24 small apps representing various vulnerabilities related to ICC.

DialDroid-Bench [32] is another benchmark focused towards the Android taint-analysis tools that consists of 30 real world applications. It only consists of the apk files without any source code or vulnerability details making it difficult to put it into use for analysis of the tools.

This research chooses to use the Ghera benchmark over the other benchmark for several reasons. First, the Ghera benchmark covers a wider range of issues than other benchmarks across several categories. It covers a broader range of issues than the benchmarks such as DroidBench, ICC-Bench, DialDroid-Bench, which are focused towards the taint analysis tools. Secondly, the Ghera benchmark consists of micro-benchmarks that makes it easier to look at each issue separately. Though DIVA and PIVAA also represent categories that focuses issues not just related to the taint analysis, they contain all the errors in a single application. This makes the analysis difficult because it is difficult to verify if the issue is reported correctly. Lastly, the Ghera benchmark also provides good documentation of the issues and a version of the benchmark app with the issue fixed. This is particularly useful because it helps to recognize the false negatives reported by the tools.

2.5 Trust and Subjective Logic

Trust is the measurement of the degree of belief or disbelief of one entity towards another entity. In the thesis, our trust is related to the security of the android application. The trust of an application is based on the evidences gathered using different static analysis tools.

We are using Subjective Logic to represent the trust for the application. The subjective logic is a probabilistic model created by Jøsang [8] which defines an agent's opinion of a system in terms of belief, disbelief and uncertainty. We are using this opinion in terms of trustworthiness developed through the evidences gathered.

In traditional probabilistic model, we can consider a proposition is considered to be binary (either it is true or false). Subjective Logic takes into account the factor of uncertainty that we encounter in the real world. Subjective Logic proposes a belief model. The trust of a system is defined using $\omega = (b, d, u)$ tuple where the values b, d and u represent the belief, disbelief and uncertainty. An important property similar to traditional probabilistic model is that the sum of b, d, and u is always 1.

Jøsang has also described various operators [8] for combining the opinions from different sources. The operators are as follows:

- Conjunction: It is used to combine opinion of an agent about two different subjects using conjunction ("and" operation). Suppose that $\omega_x = (b_x, d_x, u_x)$ is an opinion that app x can be trusted and $\omega_y = (b_y, d_y, u_y)$ is an opinion that app y can be trusted. Then the opinion that both app x and app y can be trusted is represented as $\omega_{x \wedge y} = (b_{x \wedge y}, d_{x \wedge y}, u_{x \wedge y})$ such that

$$b_{x \wedge y} = b_x b_y \qquad \text{(Eq 1)}$$

$$d_{x \wedge y} = d_x + d_y - d_x d_y \qquad \text{(Eq 2)}$$

$$u_{x \wedge y} = b_x u_y + u_x b_y + u_x u_y \qquad \text{(Eq 3)}$$

- Disjunction: It is used to combine opinion of an agent about two different subjects using disjunction (“or” operation). Suppose that $\omega_x = (b_x, d_x, u_x)$ is an opinion that app *x* can be trusted and $\omega_y = (b_y, d_y, u_y)$ is an opinion that app *y* can be trusted. Then the opinion that either app *x* or app *y* can be trusted is represented as $\omega_{x \vee y} = (b_{x \vee y}, d_{x \vee y}, u_{x \vee y})$ such that

$$b_{x \wedge y} = b_x + b_y - b_x b_y \qquad \text{(Eq 4)}$$

$$d_{x \wedge y} = d_x d_y \qquad \text{(Eq 5)}$$

$$u_{x \wedge y} = d_x u_y + u_x d_y + u_x u_y \qquad \text{(Eq 6)}$$

- Negation: It is a unary operation that represents an opinion being false. Suppose that $\omega_x = (b_x, d_x, u_x)$ is an opinion that app *x* can be trusted. Then the opinion that the app *x* cannot be trusted is represented as $\neg\omega_x = (b_x, d_{\neg x}, u_{\neg x})$ such that

$$b_{\neg x} = d_x \qquad \text{(Eq 7)}$$

$$d_{\neg x} = b_x \qquad \text{(Eq 8)}$$

$$u_{\neg x} = u_x \qquad \text{(Eq 9)}$$

- Discounting (⊗): Discounting operation is used for chaining operations. Suppose that $\omega_x^t = (b_x^t, d_x^t, u_x^t)$ is an opinion that app *x* can be trusted according to evidences from tool *t*, and $\omega_t = (b_t, d_t, u_t)$ represent an opinion that the tool *t* can be trusted. Then the overall opinion that the app *x* can be trusted given the opinion about tool *t* can be represented as $\omega_x = \omega_t \otimes \omega_x^t = (b_x, d_x, u_x)$ such that

$$b_x = b_t b_x^t \qquad \text{(Eq 10)}$$

$$d_x = d_t d_x^t \qquad \text{(Eq 11)}$$

$$u_x = d_t + u_t + b_t u_x^t \qquad \text{(Eq 12)}$$

- Consensus Operator (⊕): The consensus operator is an operator that is used to combine opinions from two different agents. The consensus operator tries to reduce the uncertainty by combining two opinions. Suppose $\omega_{t1} = (b_{t1}, d_{t1}, u_{t1})$ and $\omega_{t2} = (b_{t2}, d_{t2}, u_{t2})$ are the opinions that an app x can be trusted according two tools. Then the combined opinion can be represented as $\omega_x = \omega_{t1} \oplus \omega_{t2} = (b_x, d_x, u_x)$ such that

$$b_x = {(b_{t1}u_{t2} + u_{t1}b_{t2})}/_{\kappa} \qquad \text{(Eq 13)}$$

$$d_x = {(d_{t1}u_{t2} + u_{t1}d_{t2})}/_{\kappa} \qquad \text{(Eq 14)}$$

$$u_x = {(u_{t1}u_{t2})}/_{\kappa} \qquad \text{(Eq 15)}$$

where

$$\kappa = u_{t1} + u_{t2} - u_{t1}u_{t2} \qquad \text{(Eq 16)}$$

The conjunction, disjunction and consensus operators can be applied in any order since they are commutative and associative.

Jøsang [33] has also proposed on Ordering operation to sort different opinions to order opinions about different agents. Given two applications, the application that can be trusted more is the application that has the highest $(b + u) / (b + d + 2u)$ if the values are equal, else the application with the lowest uncertainty u. We can use this comparison logic to order arbitrary number of applications.

For weighted consensus, Zhou et. al. [34] have proposed a cumulative weighted fusion operator. Given $\omega_{e1} = (b_{e1}, d_{e1}, u_{e1})$ and $\omega_{e2} = (b_{e2}, d_{e2}, u_{e2})$ from two different set of evidences with weights α and β respectively, we can use the fusion operator to calculate the combined opinion as $\omega_x = (b_x, d_x, u_x)$ such that

$$b_x = \frac{(\kappa - u_{e1}u_{e2})(\alpha\, b_{e1}u_{e2} + \beta\, b_{e2}u_{e1})}{\kappa(\alpha u_{e2} + \beta u_{e1} + (\alpha + \beta)u_{e1}u_{e2})} \qquad \text{(Eq 17)}$$

$$d_x = \frac{(\kappa - u_{e1}u_{e2})(\alpha\, d_{e1}u_{e2} + \beta\, d_{e2}u_{e1})}{\kappa(\alpha u_{e2} + \beta u_{e1} + (\alpha + \beta)u_{e1}u_{e2})} \qquad \text{(Eq 18)}$$

$$u_x = {(u_{e1}u_{e2})}/_{\kappa} \qquad \text{(Eq 19)}$$

where

$$\kappa = u_{t1} + u_{t2} - u_{t1}u_{t2} \qquad \text{(Eq 20)}$$

The values of u_{e1} and u_{e1} cannot be 0 or 1 for using this operator because denominator will be equal to 0.

Ceolin et. al. [35] have proposed the formula for calculating the subjective logic opinion tuple from the evidences as follows:

$$b = \frac{postive\ evidence}{total\ evidence + n} \qquad \text{(Eq 21)}$$

$$d = \frac{negative\ evidence}{total\ evidence + n} \qquad \text{(Eq 22)}$$

$$u = \frac{n}{total\ evidence + n} \qquad \text{(Eq 23)}$$

where n is indicates cardinality of the set of possible outcomes.

This study uses the value of n as 2 because our opinion about an app is to either trust it or not trust it based on the evidences.

2.6 Rank Correlation

Kendall Tau's rank correlation [36] is one of the popular methods to measure the degree of correlation between two ranking schemes. Kendall Tau distance is calculated by counting the number of pairwise disagreements between two rank orders. It is defined as:

$$\tau = \frac{n_c - n_d}{\binom{n}{2}} \qquad \text{(Eq 24)}$$

where n_c is the number of concordant pairs and n_d is the number of discordant pairs.

$\binom{n}{2} = \frac{n(n-1)}{2}$ is the binomial coefficient for the number of ways to choose two out of n items.

To address ties in the rankings, different versions of Kendall Tau have been proposed over the years. The "tau-b" version accounts for ties and is defined as:

$$\tau = \frac{n_c - n_d}{\sqrt{(n_c + n_d + T)(n_c + n_d + T)}} \qquad \text{(Eq 25)}$$

where T is number of ties in the first ranking and U is the number of ties in the second ranking.

SciPy library for Python language provides the method for calculating the Kendall Tau's correlation [37].

Kendall Tau's correlation has been used in the study to compare rankings generated by different schemes and tools.

2.7 Related Work

There has not been a lot of work in the research community to rank and compare Android applications. The ratings and reviews provided by the App Stores are used by most users to select their applications. Google Play does not provide a universal ranking methodology of android apps, but rather suggests apps based on user's preferences and history.

There are third-party organizations like AppBrain [40], and AppAnnie [3] that provide ranking of Android apps based on factors such as ratings, number of downloads, active users, etc. They either use a single metric or do not specify the calculation that was performed for calculating the ranking of apps.

Besides these, there are few other works that have focused towards ranking of Android apps in the research community.

Chowdhury et. al. [38] have proposed a ranking scheme based on the internal and external views of apps. They have used FindBugs, a static analysis tools for Java code, to perform the evaluation of internal view, and used sentiment analysis on the ratings to calculate the external rating. They propose a method to combine these two views for a holistic evaluation of the Android apps. This paper builds on a previous research [39] where a ranking scheme has been proposed based on the results of Findbugs, and shows disparity between the findings and the ratings and reviews provided by the users. In another paper [40], they have proposed another ranking scheme that takes security into consideration. They have evaluated apps based on the results of FlowDroid, and text analysis of security concerns from the user reviews. This thesis borrows many concepts from these papers but taking a more detailed approach on evaluating the security of the apps. Gallege et. al. [44] have also proposed a parallel approach for selecting and recommending the apps available in online marketplaces.

Aside from the efforts in ranking applications, there have been many studies for evaluating the analysis tools themselves.

Qiu et. al. [41] have performed an analysis between the static taint analysis tools FlowDroid, AmanDroid and DroidSafe. They have performed the comparison using the DroidBench and ICC-Bench Benchmark Suites. In this research, we have performed a similar analysis for generic static analysis tools that were used.

Pauck et. al. [42] have performed an empirical evaluation on the static taint analysis tools used in the research community. They have used the DroidBench [30] to perform the analysis on six different tools. They have also proposed ReproDroid framework to perform an accurate and reproducable evaluation of the static analysis tools to overcome the differences in the evaluation techniques used by the authors of the tools.

A survey of android security threats and defenses was conducted by Rashidi et. al. [43]. Many of the threats identified are still relevant, whereas some have become outdated. There were several tools gathered in the survey, but none of the tools fit the needs of this research.

In this chapter, we have introduced all the background and theory required for our proposed approach. We have borrowed many concepts from the related work and tried to overcome some of their shortcomings in our research.

CHAPTER 3. PROPOSED APPROACH

This chapter proposes different ranking schemes that are used in our analysis. It also describes the dataset and the selection of tools. Finally, a detailed analysis of the capabilities of the tools used on our study is presented.

3.1 Ranking Scheme

A generic framework for the ranking of the android apps is show in the figure below.

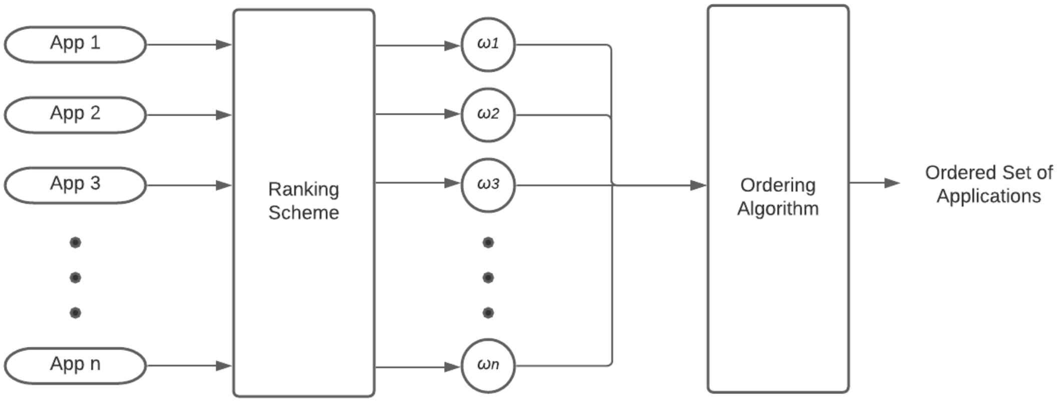


Figure 2. Generic Ranking Framework

Given a set of *n* applications, the framework should return them as an ordered set of apps by calculating their ranks based on the evidences gathered from the various static analysis tools. The schemes take the apk files as input and generate a subjective opinion on the applications. The ordering operation of subjective logic is then used as a ranking algorithm on these subjective logic opinions to get the ranks of the applications.

This study proposes three different base schemes to generate the subjective opinions on the trustworthiness of the applications. Each static analysis tools in our experiment are used in these schemes to gather the evidences and generate opinions. Two more schemes are proposed for combining the opinions from different static analysis tools. The schemes are discussed in more detail in the subsections below.

3.1.1 Base Scheme based on the categories of evidences (B1)

The first scheme uses the categories of good practices and security vulnerabilities that are reported by the tools.

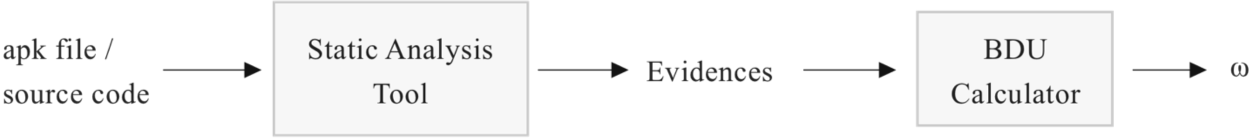


Figure 3. Base Scheme based on categories of evidences (B1)

All the static analysis tools generate different categories of evidences. Each category representing good practice is treated as positive evidence, and each potential security vulnerability is treated as negative evidence. The BDU calculator uses the count of positive and negative categories of evidences to output the subjective logic tuple ω using equations 21-23.

The ranking generated by this scheme will represent the trustworthiness of applications based on the different kinds of vulnerabilities that may be present in the application.

3.1.2 Base Scheme based on the frequency of evidences (B2)

The second base scheme considers the frequency of occurrence of each category of the evidences.

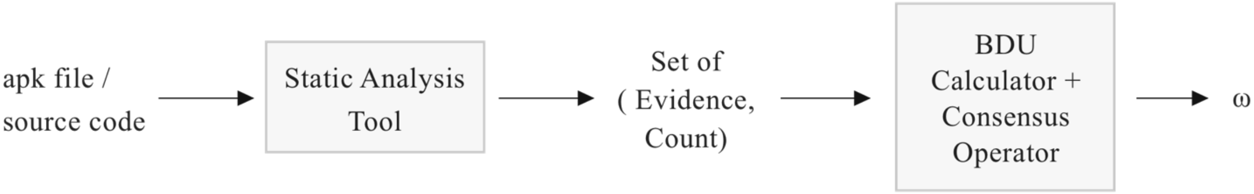


Figure 4. Base Scheme based on the frequency of evidences (B2)

In this scheme, the different categories of evidences along with the frequency of the occurrences of each of the findings are collected. Given the category and its count, the subjective logic can be used to calculate subjective logic tuples (ω_{e1}, ω_{e2}, … ω_{en}) using equations 21-23 where the count is used as the number of positive/negative evidence. These opinions for different categories of evidences can be combined using the consensus operator (given by equations 13-16) to give the final opinion ω of the scheme.

The ranking generated by this scheme takes into consideration the number of attack surfaces present in the application.

3.1.3 Base Scheme based on the severity of evidences (B3)

Besides the frequency of the evidences, the severity of each evidence could be an important metric for ranking the applications. This scheme adds weights to the consensus operation used in scheme B2.

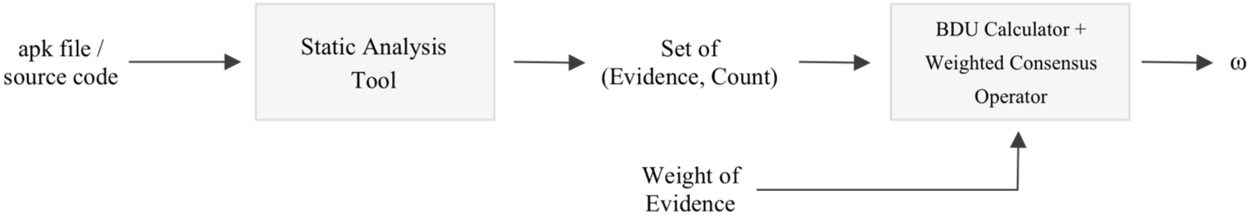


Figure 5. Base Scheme based on the severity of evidences (B3)

The static analysis tools return the frequency of each evidence. In addition to the frequency, a weight representing the severity of evidences is calculated for each evidence.

To calculate the severity of evidences, we explored various approaches. We tried to assign Common Weakness Enumeration (CWE) [5] to each evidence, but the enumerations were not available for all the evidences reported by the tools. Another approach was to use the OWASP Mobile Top 10 [4], but the ten categories were broad and the vulnerabilities within each category could have different severity. On the other hand, the CVSS score could be calculated separately for each evidences, and it provided a wholesome score based on various factors such as attack vector, attack complexity, scope, impact (confidentiality, integrity, availability), etc. that we have already discussed in detail in Chapter 2. Thus, we decided to use the CVSS as the weight for this scheme.

The consensus operator in the base scheme (B2) can be replaced with a weighted consensus operator using the calculated weight. The weighted consensus operator is given by equations 17-20.

The ranking generated by this scheme considers the different attack surfaces and their potential impact.

3.1.4 Consensus Scheme treating each tool equally (C1)

One of the ways of combining the opinions is to treat each tool equally. The scheme can be represented as shown below.

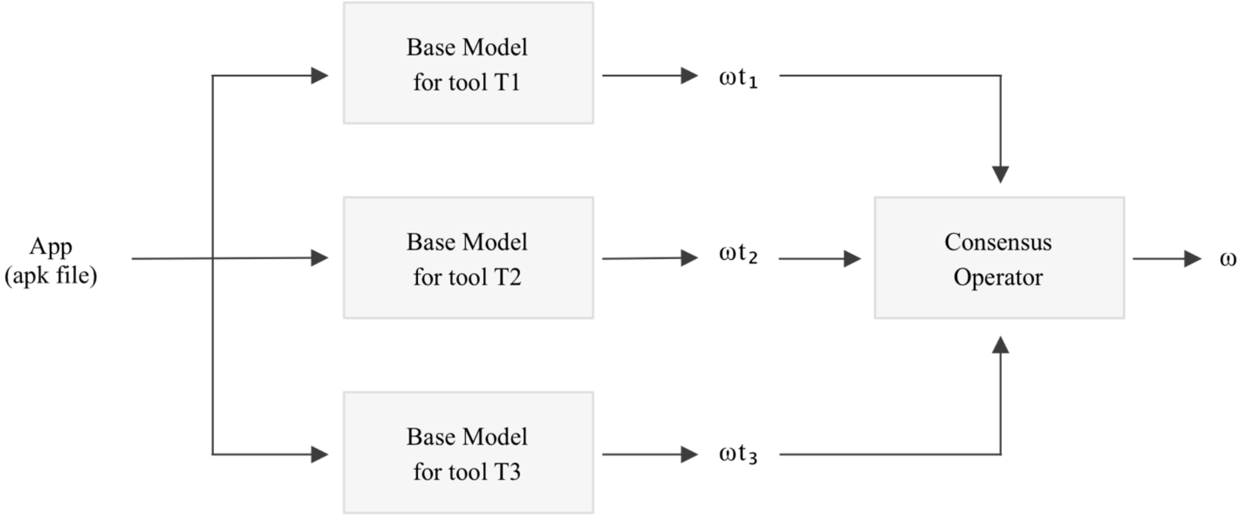


Figure 6. Consensus Scheme treating each tool equally (C1)

This scheme can utilize any of the base schemes to get the opinions from the different tools and use the consensus operator (equation 13-16) to combine the results.

3.1.5 Consensus Scheme based on the trust of the tools (C2)

This scheme adds the trust of the tools as an additional factor. The results of running the benchmarks can be used to calculate the trust of the tools. The trust is discussed in detail in Section 4.1. Assuming the trust of tools are known, the discounting operator (equation 10-12) can be used to weigh the opinions from each tool before applying the consensus operator (equation 13-16).

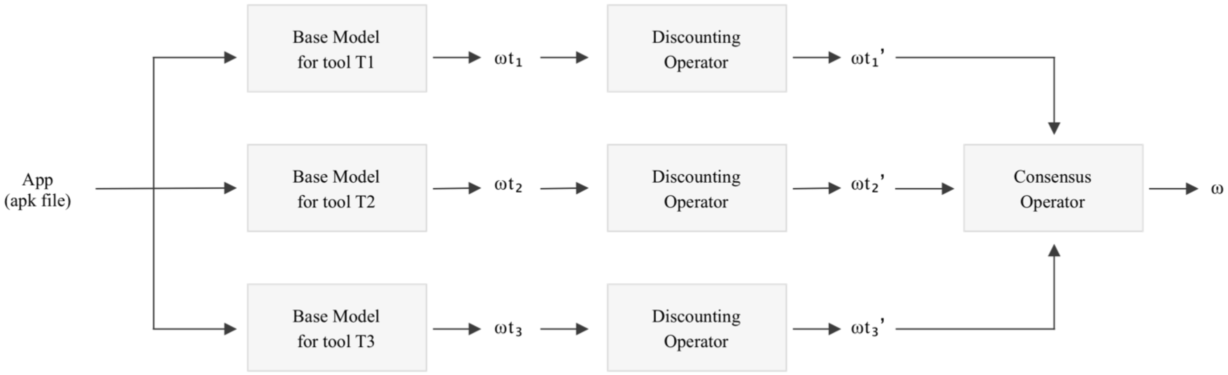


Figure 7. Consensus Scheme using the trust of tools (C2)

3.2 Dataset

The dataset consists of 175 applications from 7 different categories. The apk files were downloaded from APKPure [44]. The categories and number of applications in each category is listed in table below.

Table 3. Dataset Count

Category	**Number of Applications**
News	5
Travel	5
Shopping	5
Insurance	5
Finance	5
Games	50
Tools	50
Photography	50

Each category consists of applications with similar functionality. For example, the games are all 2048 puzzles, tools are all scientific calculators, photography apps are all photo editors. Having similar functionality will results in a fair comparison and ranking of applications.

The five categories have 5 applications are used for a detailed analysis of the ranking schemes. The three other categories having a larger application set are used to verify our initial analysis and to calculate correlation between different schemes and tools.

3.3 Tools Selection

For the selection of tools, we focused on generic static analysis tools that covers many aspects related to the security of the tools. The taint analysis tools are focused only towards data leakages are were not considered. Dynamic code analysis tools take a long time for analysis and may not cover all execution paths. Among the potential tools identified, JAADAS [26] result in error for the analysis of most of the applications. MobSF [25], AndroBugs Framework [24], and QARK [23] were the tools that were tested against various application and could run successfully. Thus, we are using these three tools for our analysis.

3.4 Tool Analysis

All the tools used in this study have different formats for input, output, and storing results. In this section we are exploring the capabilities of the tools for detecting various kinds of good practices and vulnerabilities.

3.4.1 **MobSF**

MobSF provides a user interface to upload apk files and perform a detailed analysis on the apk file. It displays the analysis results in the web app but it can also return the results in json format through Rest API. It can also perform analysis on iOS applications.

MobSF extracts all the metadata of the apk including the name, package name, the launcher activity, minimum SDK, maximum SDK, version code and version name. It also tracks the manifest along with all the activities, services, receiver, and providers for the app. It performs various kinds of security analysis such as checking signer certificate, checking permissions, binary analysis, manifest analysis, code analysis, file analysis, and malware analysis. It also allows dynamic analysis on the apps.

Among the analysis that the tool provides, the manifest analysis, and code analysis contain the results of the static analysis related to most of the security vulnerabilities. Other analysis did not provide any significant evidences that could be used in our calculations.

In the following sections we look at the code analysis and manifest analysis done by the tool in detail.

Code Analysis

The code analysis returns the good practices and potential vulnerabilities in the application code. The tool is capable of finding 46 different categories of security evidences. The findings are categorized as one of the following four levels:

- good (16 categories)
- info (4 categories)
- warning (5 categories)
- high (21 categories)

The good practices found by the code analysis are always reported under the 'good' level. The other categories represent the security vulnerabilities.

Although the three levels (high, info, warning) for the vulnerabilities seem to represent the severity of the vulnerability, a detailed of the analysis of the type of evidences indicates that it is not true. It seems to be representing the likelihood of the finding to be an actual security vulnerability. For example, among the findings reported by MobSF [25], "Insecure WebView Implementation. Execution of user controlled code in WebView is a critical Security Hole." has a warning level, but it is of a high severity vulnerability, and "App creates temp file. Sensitive information should never be written into a temp file." has a high level, but the severity is not as high. Thus, it is preferable to treat all the three levels equally and analyze each category of evidence for its severity using other metrics such as CVSS.

Manifest Analysis

The manifest rules are defined under the tool's source code defines 45 categories with a code, title, name, level, and description for each category. But the results returned after analysis does not contain the code which is the unique value for the categories. Several code sets have the same name as shown in Table 4 (extracted from the source code of MobSF [25]).

Table 4. MobSF Manifest categories from source code (from [25])

Code	**Name**
a_allowbackup_miss	Application Data can be Backed up [android:allowBackup] flag is missing.
a_allowbackup	Application Data can be Backed up [android:allowBackup=true]
a_testonly	Application is in Test Mode [android:testOnly=true]
a_dailer_code	Dailer Code: Found [android:scheme="android_secret_code"]
a_sms_receiver_port	Data SMS Receiver Set on Port: Found [android:port]
a_debuggable	Debug Enabled For App [android:debuggable=true]
a_high_action_priority	High Action Priority [android:priority]
a_high_intent_priority	High Intent Priority [android:priority]
a_improper_provider	Improper Content Provider Permissions
a_not_protected	is not Protected. [android:exported=true]
c_not_protected	is not Protected.[[Content Provider, targetSdkVersion < 17]
a_not_protected_filter	is not Protected.An intent-filter exists.
c_prot_unknown_appl	is Protected by a permission at application level, but the protection level of the permission should be checked.[Content Provider, targetSdkVersion < 17]
c_prot_normal_new_appl	is Protected by a permission at the application level should be checked, but the protection level of the permission if the application runs on a device where the the API level is less than 17.[Content Provider, targetSdkVersion >= 17]
c_prot_unknown_new_appl	is Protected by a permission at the application level, but the protection level of the permission should be checked if the application runs on a device where the the API level is less than 17.[Content Provider, targetSdkVersion >= 17]

Table 4. Continued

c_prot_dangter_appl	is Protected by a permission at the application level, but the protection level of the permission should be checked if the application runs on a device where the the API level is less than 17.[Content Provider, targetSdkVersion >= 17]
a_prot_danger_appl	is Protected by a permission at the application level, but the protection level of the permission should be checked.[android:exported=true]
a_prot_normal_appl	is Protected by a permission at the application level, but the protection level of the permission should be checked.[android:exported=true]
a_prot_sign_sys_appl	is Protected by a permission at the application level, but the protection level of the permission should be checked.[android:exported=true]
c_prot_danger_new	is Protected by a permission at the application level, but the protection level of the permission should be checked.[Content Provider, targetSdkVersion < 17]
c_prot_normal_appl	is Protected by a permission at the application level, but the protection level of the permission should be checked.[Content Provider, targetSdkVersion < 17]
c_prot_sign_sys_appl	is Protected by a permission at the application level, but the protection level of the permission should be checked.[Content Provider, targetSdkVersion < 17]
c_prot_sign_sys_new_appl	is Protected by a permission at the application level, but the protection level of the permission should be checked.[Content Provider, targetSdkVersion >= 17]
c_prot_sign_appl	is Protected by a permission at the application level.[Content Provider, targetSdkVersion < 17]

Table 4. Continued

c_prot_sign_new_appl	is Protected by a permission at the application level.[Content Provider, targetSdkVersion >= 17]
a_prot_unknown_appl	is Protected by a permission at the application, but the protection level of the permission should be checked.[android:exported=true]
c_prot_unknown_new	is Protected by a permission, but the protection level of the permission should be checked if the application runs on a device where the the API level is less than 17.[Content Provider, targetSdkVersion >= 17]
c_prot_normal_new	is Protected by a permission, but the protection level of the permission should be checked if the application runs on a device where the the API level is less than 17 [Content Provider, targetSdkVersion >= 17]
c_prot_danger_new_appl	is Protected by a permission, but the protection level of the permission should be checked if the application runs on a device where the the API level is less than 17.[Content Provider, targetSdkVersion >= 17]
a_prot_danger	is Protected by a permission, but the protection level of the permission should be checked.[android:exported=true]
a_prot_normal	is Protected by a permission, but the protection level of the permission should be checked.[android:exported=true]
a_prot_sign_sys	is Protected by a permission, but the protection level of the permission should be checked.[android:exported=true]
a_prot_unknown	is Protected by a permission, but the protection level of the permission should be checked.[android:exported=true]
c_prot_danger	is Protected by a permission, but the protection level of the permission should be checked.[Content Provider, targetSdkVersion < 17]

Table 4. Continued

c_prot_normal	is Protected by a permission, but the protection level of the permission should be checked.[Content Provider, targetSdkVersion < 17]
c_prot_sign_sys	is Protected by a permission, but the protection level of the permission should be checked.[Content Provider, targetSdkVersion < 17]
c_prot_unknown	is Protected by a permission, but the protection level of the permission should be checked.[Content Provider, targetSdkVersion < 17]
c_prot_sign_sys_new	is Protected by a permission, but the protection level of the permission should be checked.[Content Provider, targetSdkVersion >= 17]
a_prot_sign	is Protected by a permission.[android:exported=true]
c_prot_sign	is Protected by a permission.[Content Provider, targetSdkVersion < 17]
c_prot_sign_new	is Protected by a permission.[Content Provider, targetSdkVersion >= 17]
a_launchmode	Launch Mode of Activity is not standard.
a_prot_sign_appl	Protected by a permission at the application level.[android:exported=true]
a_taskaffinity	TaskAffinity is set for Activity
c_not_protected2	would not be Protected if the application ran on a device where the the API level was less than 17.[Content Provider, targetSdkVersion >= 17]

The common names used for multiple codes are as follows:

- “c_prot_unknown_new_appl” and “c_prot_danger_appl” codes both have the name “is Protected by a permission at the application level should be checked, but the protection level of the permission if the application runs on a device where the the API level is less than 17.[Content Provider, targetSdkVersion >= 17]”.

- “a_prot_danger_appl”, “a_prot_normal_appl”, and “a_prot_sign_sys_appl” all have the name “is Protected by a permission at the application level, but the protection level of the permission should be checked.[android:exported=true]”.
- “c_prot_danger_new”, “c_prot_normal_appl”, and “c_prot_sys_appl” all have the name “is Protected by a permission at the application level, but the protection level of the permission should be checked.[Content Provider, targetSdkVersion < 17]”.
- “c_prot_unknwon_new”, “c_prot_normal_new” and “c_prot_danger_new_appl” all have the name “is Protected by a permission, but the protection level of the permission should be checked if the application runs on a device where the the API level is less than 17.[Content Provider, targetSdkVersion >= 17]”.
- “a_prot_danger”, “a_prot_normal”, “a_prot_sign_sys”, and “a_prot_unknown” all have the name “is Protected by a permission, but the protection level of the permission should be checked.[android:exported=true]”.
- “c_prot_danger”, “c_prot_normal”, “c_prot_sign_sys”, and “c_prot_unknown” all have the name “is Protected by a permission, but the protection level of the permission should be checked.[Content Provider, targetSdkVersion < 17]”.

After removing the duplicates, we marked each finding as a “vulnerability” or a “good practice”. Though the tool provides “high”, “medium” and “info” levels, the “info” level finding may be a “vulnerability” or a “good practice”. The final dataset consists of 32 categories out of which 6 are good practices and 26 vulnerabilities as represented in Table 5 (where the “Name” represents a vulnerability or good practice as extracted from MobSF [25]).

Table 5. Final List of findings for MobSF Manifest Analysis

Name	**Type**
Application Data can be Backed up [android:allowBackup] flag is missing.	vulnerability
Application Data can be Backed up [android:allowBackup=true]	vulnerability
Application is in Test Mode [android:testOnly=true]	vulnerability
Dailer Code: Found [android:scheme="android_secret_code"]	vulnerability
Data SMS Receiver Set on Port: Found [android:port]	vulnerability
Debug Enabled For App [android:debuggable=true]	vulnerability
High Action Priority [android:priority]	vulnerability
High Intent Priority [android:priority]	vulnerability
Improper Content Provider Permissions	vulnerability
is not Protected. [android:exported=true]	vulnerability
is not Protected.[[Content Provider, targetSdkVersion < 17]	vulnerability
is not Protected.An intent-filter exists.	vulnerability
is Protected by a permission at application level, but the protection level of the permission should be checked.[Content Provider, targetSdkVersion < 17]	vulnerability
is Protected by a permission at the application level should be checked, but the protection level of the permission if the application runs on a device where the the API level is less than 17.[Content Provider, targetSdkVersion >= 17]	vulnerability
is Protected by a permission at the application level, but the protection level of the permission should be checked if the application runs on a device where the the API level is less than 17.[Content Provider, targetSdkVersion >= 17]	vulnerability
is Protected by a permission at the application level, but the protection level of the permission should be checked.[android:exported=true]	vulnerability
is Protected by a permission at the application level, but the protection level of the permission should be checked.[Content Provider, targetSdkVersion < 17]	vulnerability
is Protected by a permission at the application level, but the protection level of the permission should be checked.[Content Provider, targetSdkVersion >= 17]	vulnerability
is Protected by a permission at the application level.[Content Provider, targetSdkVersion < 17]	good
is Protected by a permission at the application level.[Content Provider, targetSdkVersion >= 17]	good
is Protected by a permission at the application, but the protection level of the permission should be checked.[android:exported=true]	vulnerability
is Protected by a permission, but the protection level of the permission should be checked if the application runs on a device where the the API level is less than 17 [Content Provider, targetSdkVersion >= 17]	vulnerability

Table 5. Continued

is Protected by a permission, but the protection level of the permission should be checked.[android:exported=true]	vulnerability
is Protected by a permission, but the protection level of the permission should be checked.[Content Provider, targetSdkVersion < 17]	vulnerability
is Protected by a permission, but the protection level of the permission should be checked.[Content Provider, targetSdkVersion >= 17]	vulnerability
is Protected by a permission.[android:exported=true]	good
is Protected by a permission.[Content Provider, targetSdkVersion < 17]	good
is Protected by a permission.[Content Provider, targetSdkVersion >= 17]	good
Launch Mode of Activity is not standard.	vulnerability
Protected by a permission at the application level.[android:exported=true]	good
TaskAffinity is set for Activity	vulnerability
would not be Protected if the application ran on a device where the the API level was less than 17.[Content Provider, targetSdkVersion >= 17]	vulnerability

3.4.2 **AndroBugs**

AndroBugs is a command line tool that can perform fast analysis on large number of applications. It is a Python program that provides various commands to perform analysis on a single apk or a set of apk files. It provides commands to display the results in command line and also outputs the results into text files. It internally uses MongoDB to store the results. We are leveraging this database directly to query the analysis results.

AndroBugs reports 51 different categories of finding with four levels (info, warning, critical, and notice). Instead of reporting only the findings it returns all 51 categories, and uses "info" level to notify the absence of the category. "Warning" and "Category" represent the various types vulnerabilities. The "Notice" level can have a generic notice that is neither a good practice, nor a vulnerability, or it can also report good practices and vulnerabilities. After manually categorizing the "Notice" level we obtain a total of 38 categories as vulnerabilities, 2 categories as generic messages (that can be ignored during analysis) and 11 categories as good practices as shown in Table 6 (with the Finding codes reported by AndroBugs [24] reproduced verbatim).

Table 6. All categories of findings reported by Androbugs and their type

Finding	**Type**
ALLOW_BACKUP	vulnerability
COMMAND	vulnerability
COMMAND_MAYBE_SYSTEM	vulnerability
DB_DEPRECATED_USE1	vulnerability
DEBUGGABLE	vulnerability
DYNAMIC_CODE_LOADING	vulnerability
EXTERNAL_STORAGE	vulnerability
FILE_DELETE	vulnerability
FRAGMENT_INJECTION	vulnerability
HACKER_BASE64_STRING_DECODE	vulnerability
HACKER_KEYSTORE_NO_PWD	vulnerability
HTTPURLCONNECTION_BUG	vulnerability
KEYSTORE_TYPE_CHECK	vulnerability
MASTER_KEY	vulnerability
MODE_WORLD_READABLE_OR_MODE_WORLD_WRITABLE	vulnerability
PERMISSION_DANGEROUS	vulnerability
PERMISSION_EXPORTED	vulnerability
PERMISSION_GROUP_EMPTY_VALUE	vulnerability
PERMISSION_IMPLICIT_SERVICE	vulnerability
PERMISSION_INTENT_FILTER_MISCONFIG	vulnerability
PERMISSION_NO_PREFIX_EXPORTED	vulnerability
PERMISSION_NORMAL	vulnerability
PERMISSION_PROVIDER_EXPLICIT_EXPORTED	vulnerability
PERMISSION_PROVIDER_IMPLICIT_EXPORTED	vulnerability
SENSITIVE_DEVICE_ID	vulnerability
SENSITIVE_SECURE_ANDROID_ID	vulnerability
SENSITIVE_SMS	vulnerability

Table 6. Continued

SSL_CN1	vulnerability
SSL_CN2	vulnerability
SSL_CN3	vulnerability
SSL_DEFAULT_SCHEMA_NAME	vulnerability
SSL_URLS_NOT_IN_HTTPS	vulnerability
SSL_WEBVIEW	vulnerability
SSL_X509	vulnerability
USE_PERMISSION_SYSTEM_APP	vulnerability
WEBVIEW_ALLOW_FILE_ACCESS	vulnerability
WEBVIEW_JS_ENABLED	vulnerability
WEBVIEW_RCE	vulnerability
FRAMEWORK_MONODROID	generic notice
MANIFEST_GCM	generic notice
DB_SEE	good practice
DB_SQLCIPHER	good practice
DB_SQLITE_JOURNAL	good practice
HACKER_DB_KEY	good practice
HACKER_DEBUGGABLE_CHECK	good practice
HACKER_INSTALL_SOURCE_CHECK	good practice
HACKER_KEYSTORE_LOCATION1	good practice
HACKER_KEYSTORE_SSL_PINNING	good practice
HACKER_PREVENT_SCREENSHOT_CHECK	good practice
HACKER_SIGNATURE_CHECK	good practice
SHARED_USER_ID	good practice

3.4.3 Qark

Qark is a static analysis tool that detects security vulnerabilities in Android apps. It is programmed in Python and provides a command line interface. It can analyze both source code and apk files. It reports the results in html or json format. Each finding reports the name, category, line number, severity (error/vulnerability/info/warning), description, and the file where the vulnerability was detected. It is not capable of detecting good practices. We are not using the severity from the tool but instead calculating our own severity based on CVSS.

Qark does not enlist the vulnerabilities it detects, so we had to manually extract it from the source code. A total of 45 vulnerabilities that could be detected were identified. The vulnerabilities along with the CVSS scores are listed in the CVSS calculation section later.

3.5 CVSS Calculation

Since we are using the CVSS score as a weight on ranking scheme based on the severity of evidences, we calculate the CVSS for the different findings reported by the tools. CVSS scores are usually assigned for the security vulnerabilities discovered by security experts separately for each specific app. Instead, we are assigning a generic score depending on the description and vulnerabilities reported on the National Vulnerability Database (NVD) [21].

The CVSS is not used in general for good practices, but to add weights to good practices, we are assigning the CVSS scores for good practices based on the vulnerability it tries to prevent. If there is a vulnerability *V* with a CVSS score *X*, we assign a good practice *G* that prevents *V* the same score *X*.

The CVSS calculations for the different tools are discussed in detail in the following subsections below.

3.5.1 MobSF

The MobSF provides the CVSS score for 23 out of 46 Code Analysis findings. The defined findings are listed in Table 7 (extracted, and reported verbatim below, from MobSF [25]). For the 23 rest static analysis findings (extracted, and reported verbatim below, from MobSF [25]), the CVSS scores were calculated as shown in Table 8.

Table 7. CVSS Scores Provided by MobSF [25]

Finding	**CVSS**
App can read/write to External Storage. Any App can read data written to External Storage.	5.5
App can write to App Directory. Sensitive Information should be encrypted.	3.9
App creates temp file. Sensitive information should never be written into a temp file.	5.5
App uses SQLite Database and execute raw SQL query. Untrusted user input in raw SQL queries can cause SQL Injection. Also sensitive information should be encrypted and written to the database.	5.9
Files may contain hardcoded sensitive informations like usernames, passwords, keys etc.	7.4
Hidden elements in view can be used to hide data from user. But this data can be leaked	4.3
Insecure Implementation of SSL. Trusting all the certificates or accepting self-signed certificates is a critical Security Hole. This application is vulnerable to MITM attacks	7.4
Insecure WebView Implementation. Execution of user controlled code in WebView is a critical Security Hole.	8.8
Insecure WebView Implementation. WebView ignores SSL Certificate errors and accept any SSL Certificate. This application is vulnerable to MITM attacks	7.4
IP Address disclosure	4.3
MD5 is a weak hash known to have hash collisions.	7.4
Remote WebView debugging is enabled.	5.4
SHA-1 is a weak hash known to have hash collisions.	5.9
The App logs information. Sensitive information should never be logged.	7.5
The App may use weak IVs like "0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00" or "0x01,0x02,0x03,0x04,0x05,0x06,0x07". Not using a random IV makes the resulting ciphertext much more predictable and susceptible to a dictionary attack.	9.8
The App uses an insecure Random Number Generator.	7.5

Table 7. Continued

The App uses ECB mode in Cryptographic encryption algorithm. ECB mode is known to be weak as it results in the same ciphertext for identical blocks of plaintext.	5.9
The file is World Readable and Writable. Any App can read/write to the file	6
The file is World Readable. Any App can read from the file	4
The file is World Writable. Any App can write to the file	6
This App uses Java Hash Code. It's a weak hash function and should never be used in Secure Crypto Implementation.	2.3
This App uses RSA Crypto without OAEP padding. The purpose of the padding scheme is to prevent a number of attacks on RSA that only work when the encryption is performed without padding.	5.9
Weak Hash algorithm used	7.4
WebView load files from external storage. Files in external storage can be modified by any application.	5

Table 8. Calculated CVSS for MobSF Code Analysis

Finding	**Vector**	**CVSS**
These activities prevent screenshot when they go to background.	CVSS:3.0/AV:P/AC:L/PR:L/UI:R/S:U/C:H/I:N/A:N	4.1
This App uses SQL Cipher. But the secret may be hardcoded.	CVSS:3.0/AV:P/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H	6.6
This app has capabilities to prevent tapjacking attacks.	CVSS:3.0/AV:P/AC:H/PR:H/UI:R/S:U/C:H/I:L/A:L	4.9
This App uses SQL Cipher. SQLCipher provides 256-bit AES encryption to sqlite database files.	CVSS:3.0/AV:P/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:H	6.1
This app listens to Clipboard changes. Some malwares also listen to Clipboard changes.	CVSS:3.0/AV:P/AC:H/PR:L/UI:R/S:C/C:H/I:N/A:N	4.7
This App copies data to clipboard. Sensitive data should not be copied to clipboard as other applications can access it.	CVSS:3.0/AV:P/AC:H/PR:H/UI:R/S:U/C:L/I:L/A:N	2.7
This App detects frida server.	CVSS:3.0/AV:N/AC:H/PR:H/UI:N/S:U/C:H/I:L/A:L	5.5
This App uses an SSL Pinning Library (org.thoughtcrime.ssl.pinning) to prevent MITM attacks in secure communication channel.	CVSS:3.0/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:N/A:N	5.9
This App has capabilities to prevent against Screenshots from Recent Task History/ Now On Tap etc.	CVSS:3.0/AV:P/AC:L/PR:L/UI:R/S:U/C:H/I:N/A:N	4.1

Table 8. Continued

This App may request root (Super User) privileges.	CVSS:3.0/AV:P/AC:H/PR:H/UI:N/S:C/C:H/I:H/A:L	6.8
This App may have root detection capabilities.	CVSS:3.0/AV:P/AC:H/PR:H/UI:N/S:C/C:H/I:H/A:L	6.8
This App use Realm Database with Encryption.	CVSS:3.0/AV:P/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:H	6.1
This App may uses Safenet API.	CVSS:3.0/AV:P/AC:L/PR:L/UI:N/S:U/C:L/I:H/A:L	5.4
The App mayuse package signature for tamper detection.	CVSS:3.0/AV:P/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:N	5.4
DexGuard Signer Certificate Tamper Detection code is identified.	CVSS:3.0/AV:P/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:N	5.4
DexGuard App Tamper Detection code is identified.	CVSS:3.0/AV:P/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:N	5.4
DexGuard Root Detection code is identified.	CVSS:3.0/AV:P/AC:H/PR:H/UI:N/S:C/C:H/I:H/A:L	6.8
DexGuard code to detect weather the App is signed with a debug key or not is identified.	CVSS:3.0/AV:P/AC:H/PR:H/UI:N/S:U/C:L/I:H/A:N	4.4
DexGuard Emulator Detection code is identified.	CVSS:3.0/AV:P/AC:H/PR:H/UI:N/S:U/C:H/I:L/A:N	4.4
DexGuard Debugger Detection code is identified.	CVSS:3.0/AV:P/AC:H/PR:H/UI:N/S:U/C:H/I:L/A:N	4.4
This App download files using Android Download Manager	CVSS:3.0/AV:P/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N	2.1
DexGuard Debug Detection code to detect wheather an App is debuggable or not is identified.	CVSS:3.0/AV:P/AC:H/PR:H/UI:N/S:U/C:L/I:H/A:N	4.4

Similarly, the calculated CVSS score for Manifest Analysis for each finding (extracted, and reported verbatim below, from MobSF [25]) is shown below:

Table 9. Calculated CVSS for MobSF Manifest Analysis

Findings	**Vector**	**CVSS**
Application Data can be Backed up [android:allowBackup] flag is missing.	CVSS:3.0/AV:N/AC:H/PR:H/UI:N/S:U/C:H/I:N/A:N	4.4
Application Data can be Backed up [android:allowBackup=true]	CVSS:3.0/AV:N/AC:H/PR:H/UI:N/S:U/C:H/I:N/A:N	4.4
Application is in Test Mode [android:testOnly=true]	CVSS:3.0/AV:P/AC:H/PR:L/UI:N/S:U/C:H/I:N/A:L	4.6
Dailer Code: Found [android:scheme="android_secret_code"]	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:L	7.3
Data SMS Receiver Set on Port: Found [android:port]	CVSS:3.0/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N	3.7
Debug Enabled For App [android:debuggable=true]	CVSS:3.0/AV:P/AC:H/PR:L/UI:N/S:U/C:H/I:N/A:L	4.6
High Action Priority [android:priority]	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:C/C:N/I:N/A:L	4.3
High Intent Priority [android:priority]	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:C/C:N/I:N/A:L	4.3
Improper Content Provider Permissions	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:L	7.3
is not Protected. [android:exported=true]	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	6.2
is not Protected.[[Content Provider, targetSdkVersion < 17]	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	6.2

Table 9. Continued

is not Protected.An intent-filter exists.	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	6.2
is Protected by a permission at application level, but the protection level of the permission should be checked.[Content Provider, targetSdkVersion < 17]	CVSS:3.0/AV:L/AC:L/PR:H/UI:N/S:U/C:H/I:N/A:N	4.4
is Protected by a permission at the application level should be checked, but the protection level of the permission if the application runs on a device where the the API level is less than 17.[Content Provider, targetSdkVersion >= 17]	CVSS:3.0/AV:L/AC:L/PR:H/UI:N/S:U/C:H/I:N/A:N	4.4
is Protected by a permission at the application level, but the protection level of the permission should be checked if the application runs on a device where the the API level is less than 17.[Content Provider, targetSdkVersion >= 17]	CVSS:3.0/AV:L/AC:L/PR:H/UI:N/S:U/C:H/I:N/A:N	4.4
is Protected by a permission at the application level, but the protection level of the permission should be checked.[android:exported=true]	CVSS:3.0/AV:L/AC:L/PR:H/UI:N/S:U/C:H/I:N/A:N	4.4

Table 9. Continued

is Protected by a permission at the application level, but the protection level of the permission should be checked.[Content Provider, targetSdkVersion < 17]	CVSS:3.0/AV:L/AC:L/PR:H/UI:N/S:U/C:H/I:N/A:N	4.4
is Protected by a permission at the application level, but the protection level of the permission should be checked.[Content Provider, targetSdkVersion >= 17]	CVSS:3.0/AV:L/AC:L/PR:H/UI:N/S:U/C:H/I:N/A:N	4.4
is Protected by a permission at the application level.[Content Provider, targetSdkVersion < 17]	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	6.2
is Protected by a permission at the application level.[Content Provider, targetSdkVersion >= 17]	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	6.2
is Protected by a permission at the application, but the protection level of the permission should be checked.[android:exported=true]	CVSS:3.0/AV:L/AC:L/PR:H/UI:N/S:U/C:H/I:N/A:N	4.4
is Protected by a permission, but the protection level of the permission should be checked if the application runs on a device where the the API level is less than 17 [Content Provider, targetSdkVersion >= 17]	CVSS:3.0/AV:L/AC:L/PR:H/UI:N/S:U/C:H/I:N/A:N	4.4

Table 9. Continued

is Protected by a permission, but the protection level of the permission should be checked.[android:exported=true]	CVSS:3.0/AV:L/AC:L/PR:H/UI:N/S:U/C:H/I:N/A:N	4.4
is Protected by a permission, but the protection level of the permission should be checked.[Content Provider, targetSdkVersion < 17]	CVSS:3.0/AV:L/AC:L/PR:H/UI:N/S:U/C:H/I:N/A:N	4.4
is Protected by a permission, but the protection level of the permission should be checked.[Content Provider, targetSdkVersion >= 17]	CVSS:3.0/AV:L/AC:L/PR:H/UI:N/S:U/C:H/I:N/A:N	4.4
is Protected by a permission.[android:exported=true]	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	6.2
is Protected by a permission.[Content Provider, targetSdkVersion < 17]	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	6.2
is Protected by a permission.[Content Provider, targetSdkVersion >= 17]	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	6.2
Launch Mode of Activity is not standard.	CVSS:3.0/AV:L/AC:H/PR:N/UI:N/S:U/C:H/I:N/A:N	5.1
Protected by a permission at the application level.[android:exported=true]	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	6.2
TaskAffinity is set for Activity	CVSS:3.0/AV:L/AC:H/PR:N/UI:N/S:U/C:H/I:N/A:N	5.1
would not be Protected if the application ran on a device where the the API level was less than 17.[Content Provider, targetSdkVersion >= 17]	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	6.2

3.5.2 Androbugs

The CVSS scores for the findings (reported by AndroBugs [24] reproduced verbatim) are as follows:

Table 10. Calculated CVSS for Androbugs

Findings	**Vector**	**CVSS**
ALLOW_BACKUP	CVSS:3.0/AV:N/AC:H/PR:H/UI:N/S:U/C:H/I:N/A:N	4.4
COMMAND	CVSS:3.0/AV:L/AC:L/PR:H/UI:N/S:C/C:H/I:L/A:L	7.3
COMMAND_MAYBE_SYSTEM	CVSS:3.0/AV:L/AC:L/PR:H/UI:N/S:C/C:H/I:H/A:H	8.2
DB_DEPRECATED_USE1	CVSS:3.0/AV:L/AC:L/PR:H/UI:N/S:U/C:H/I:L/A:L	5.6
DB_SEE	CVSS:3.0/AV:P/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:H	6.1
DB_SQLCIPHER	CVSS:3.0/AV:P/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:H	6.1
DB_SQLITE_JOURNAL	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	7.5
DEBUGGABLE	CVSS:3.0/AV:P/AC:H/PR:L/UI:N/S:U/C:H/I:N/A:L	4.6
DYNAMIC_CODE_LOADING	CVSS:3.0/AV:L/AC:H/PR:L/UI:N/S:U/C:H/I:L/A:L	5.8
EXTERNAL_STORAGE	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:L	7.3
FILE_DELETE	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	6.2
FRAGMENT_INJECTION	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N	9.1
HACKER_BASE64_STRING_DECODE	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:N/I:H/A:N	6.2
HACKER_DB_KEY	CVSS:3.0/AV:P/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:H	6.1
HACKER_DEBUGGABLE_CHECK	CVSS:3.0/AV:P/AC:H/PR:L/UI:N/S:U/C:H/I:N/A:L	4.6
HACKER_INSTALL_SOURCE_CHECK	CVSS:3.0/AV:P/AC:H/PR:L/UI:N/S:U/C:H/I:L/A:L	5.1
HACKER_KEYSTORE_LOCATION1	CVSS:3.0/AV:L/AC:L/PR:H/UI:N/S:U/C:H/I:N/A:N	4.4
HACKER_KEYSTORE_NO_PWD	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	6.2
HACKER_KEYSTORE_SSL_PINNING	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	6.2
HACKER_PREVENT_SCREENSHOT_CHECK	CVSS:3.0/AV:P/AC:L/PR:L/UI:R/S:U/C:H/I:N/A:N	4.1
HACKER_SIGNATURE_CHECK	CVSS:3.0/AV:L/AC:H/PR:L/UI:N/S:U/C:H/I:N/A:N	4.7
HTTPURLCONNECTION_BUG	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	5.3
KEYSTORE_TYPE_CHECK	CVSS:3.0/AV:P/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:H	6.1
MASTER_KEY	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H	9.8

Table 10. Continued

MODE_WORLD_READABLE_ OR_MODE_WORLD_WRITABLE	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:L	5.9
PERMISSION_DANGEROUS	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H	8.4
PERMISSION_EXPORTED	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H	8.4
PERMISSION_GROUP_EMPTY_VALUE	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H	8.4
PERMISSION_IMPLICIT_SERVICE	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H	8.4
PERMISSION_INTENT_FILTER_ MISCONFIG	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H	8.4
PERMISSION_NO_PREFIX_EXPORTED	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H	8.4
PERMISSION_NORMAL	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H	8.4
PERMISSION_PROVIDER_EXPLICIT_EXPORTED	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H	8.4
PERMISSION_PROVIDER_IMPLICIT_EXPORTED	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N	7.7
SENSITIVE_DEVICE_ID	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	4
SENSITIVE_SECURE_ANDROID_ID	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	4
SENSITIVE_SMS	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	5.3
SHARED_USER_ID	CVSS:3.0/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N	5.5
SSL_CN1	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	5.3
SSL_CN2	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	5.3
SSL_CN3	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	5.3
SSL_DEFAULT_SCHEMA_NAME	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	7.5
SSL_URLS_NOT_IN_HTTPS	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	7.5
SSL_WEBVIEW	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	7.5
SSL_X509	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	7.5
USE_PERMISSION_SYSTEM_APP	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	6.2
WEBVIEW_ALLOW_FILE_ACCESS	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	7.5
WEBVIEW_JS_ENABLED	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	7.5
WEBVIEW_RCE	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	7.5

3.5.3 QARK

The findings (vulnerabilities as reported by QARK [23] reproduced verbatim) and their calculated CVSS are listed below:

Table 11. Calculated CVSS for Qark

Findings	**Vector**	**CVSS**
Dynamic broadcast receiver found	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	7.5
Sticky broadcast sent	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	7.5
Broadcast sent with receiverPermission with minimum SDK under 21	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	7.5
Broadcast sent with receiverPermission	CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N	6.5
Broadcast sent as specific user without receiverPermission	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	7.5
Broadcast sent as specific user with receiverPermission with minimum SDK under 21	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	7.5
Broadcast sent as specific user with receiverPermission	CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N	6.5
Ordered broadcast sent with receiverPermission with minimum SDK under 21	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	7.5
Ordered broadcast sent with receiverPermission	CVSS:3.0/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:N/A:N	5.9
Sticky broadcast sent	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	7.5
Empty certificate method	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N	9.1

Table 11. Continued

Empty (return) certificate method	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N	9.1
Unsafe implementation of onReceivedSslError	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N	9.1
Allow all hostname verifier used	CVSS:3.0/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N	7.4
setHostnameVerifier set to ALLOW_ALL	CVSS:3.0/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N	7.4
ECB Cipher Usage	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N	8.2
Encryption keys are packaged with the application	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N	8.2
RSA Cipher Usage	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N	8.2
Random number generator is seeded with SecureSeed	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	5.3
Logging found	CVSS:3.0/AV:P/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	4.6
Potential API Key found	CVSS:3.0/AV:P/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	4.6
External storage used	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:L	7.3
File Permissions	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:L	5.9
Hardcoded HTTP url found	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	5.3
Insecure functions found	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	4
Phone number or IMEI detected	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	4
Potientially vulnerable check permission function called	CVSS:3.0/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N	5.5
Potential task hijacking	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	4
Empty pending intent found	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	6.2
Backup is allowed in manifest	CVSS:3.0/AV:N/AC:H/PR:H/UI:N/S:U/C:H/I:N/A:N	4.4
android:path tag used	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	5.3

Table 11. Continued

Custom permissions are enabled in the manifest	CVSS:3.0/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N	5.5
Manifest is manually set to debug	CVSS:3.0/AV:P/AC:H/PR:L/UI:N/S:U/C:H/I:N/A:L	4.6
Exported tags	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N	8.2
Tap Jacking possible	CVSS:3.0/AV:P/AC:H/PR:H/UI:R/S:U/C:H/I:L/A:L	4.9
launchMode=singleTask found	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	4
android:allowTaskReparenting='true' found	CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	4
Webview uses addJavascriptInterface pre-API 17	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N	8.2
Javascript enabled in Webview	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	7.5
BaseURL set for Webview	CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N	4.3
Remote debugging enabled in Webview	CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:L	7.1
Webview enables content access	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	7.5
Webview enables file access	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	7.5
Webview enables universal access for JavaScript	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	7.5
Webview enables DOM Storage	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	7.5

CHAPTER 4. EXPERIMENTAL RESULTS

The results from our experiments are analyzed and discussed in this chapter. First, the analysis of the tools on the benchmark is shown, followed by results of the ranking schemes.

4.1 Benchmark Analysis

The Ghera benchmark consists of 61 benchmark applications each with two versions of the application: benign(B) and secure(S). The vulnerability should be detected on the benign but not on the secure(S) app. The results of the benchmark are shown in the table below.

Table 12. Ghera Results

Category	Vulnerability	Qark		Androbugs		MobSF	
		B	S	B	S	B	S
Crypto	BlockCipher-ECB-InformationExposure-Lean						
	BlockCipher-NonRandomIV-InformationExposure-Lean						
	ConstantKey-ForgeryAttack-Lean					✓	
	ExposedCredentials-InformationExposure-Lean			✓	✗		
	PBE-ConstantSalt-InformationExposure-Lean	✓				✓	✗
ICC	DynamicRegBroadcastReceiver-UnrestrictedAccess-Lean	✓	✗				
	EmptyPendingIntent-PrivEscalation-Lean	✓	✗				
	FragmentInjection-PrivEscalation-Lean						

Table 12. Continued

ICC	HighPriority-ActivityHijack-Lean						
	ImplicitPendingIntent-IntentHijack-Lean						
	InadequatePathPermission-InformationExposure-Lean			✓		✓	
	IncorrectHandlingImplicitIntent-UnauthorizedAccess-Lean	✓	×	✓		✓	
	NoValidityCheckOnBroadcastMsg-UnintendedInvocation-Lean	✓	×			✓	×
	OrderedBroadcast-DataInjection-Lean						
	StickyBroadcast-DataInjection-Lean	✓					
	TaskAffinity-ActivityHijack-Lean					✓	
	TaskAffinity-LauncherActivity-Lean						
	TaskAffinity-PhisingAttack-Lean					✓	
	TaskAffinityAndReparenting-PhisingAndDoSAttack-Lean						
	UnhandledException-DOS-Lean						
	UnprotectedBroadcastRecv-PrivEscalation-Lean	✓	×	✓		✓	
	WeakChecksOnDynamicInvocation-DataInjection-Lean	✓	×			✓	
Networking	CheckValidity-InformationExposure-Lean	✓					
	IncorrectHostNameVerification-MITM-Lean	✓		✓			
	InsecureSSLSocket-MITM-Lean			✓	×	✓	×

Table 12. Continued

Networking	InsecureSSLSocketFactory-MITM-Lean			✓		✓	×
	InvalidCertificateAuthority-MITM-Lean	✓		✓			
	OpenSocket-InformationLean-Lean						
	UnEncryptedSocketComm-MITM-Lean						
	UnpinnedCertificates-MITM-Lean						
NonAPI	MergeManifest-UnintendedBehavior-Lean						
	OutdatedLibrary-DirectoryTraversal-Lean						
Permission	UnnecessaryPerms-PrivEscalation-Lean						
	WeakPermission-UnauthorizedAccess-Lean			✓		✓	
Storage	ExternalStorage-DataInjection-Lean	✓					
	ExternalStorage-InformationLeak-Lean	✓					
	InternalStorage-DirectoryTraversal-Lean						
	InternalToExternalStorage-InformationLeak-Lean						
	SQLite-execSQL-Lean						
	SQLite-RawQuery-Lean					✓	×
	SQLite-SQLInjection-Lean					✓	×
System	CheckCallingOrSelfPermission-PrivilegeEscalation-Lean	✓	×				

Table 12. Continued

System	CheckPermission-PrivliegeEscalation-Lean	✓	✗				
	ClipboardUse-InformationExposure-Lean					✓	✗
	DynamicCodeLoading-CodeInjection-Lean						
	EnforceCallingOrSelfPermission-PrivilegeEscalation-Lean						
	EnforcePermission-PrivilegeEscalation						
	UniqueIDs-IdentityLeak-Lean	✓		✓			
Web	HttpConnection-MITM-Lean						
	JavaScriptExecution-CodeInjection-Lean	✓		✓			
	UnsafeIntentURLImpl-InformaitonExposure-Lean						
	WebView-CookieOverwrite-Lean						
	WebView-NoUserPermission-InformationExposure-Lean					✓	✗
	WebViewAllowContentAccess-UnauthorizedFileAccess-Lean	✓					
	WebViewAllowFileAccess-UnauthorizedFileAccess-Lean	✓	✗	✓	✗		
	WebViewIgnoreSSLWarning-MITM-Lean			✓		✓	
	WebViewInterceptRequest-MITM-Lean					✓	✗
	WebViewLoadDataWithBaseUrl-UnauthorizedFileAccess-Lean						

Table 12. Continued

Web	WebviewOverrideUrl-MITM-Lean						
	WebviewProceed-UnauthorizedAccess-Lean						

Qark detected the highest number (19) of the benign applications but also falsely identified vulnerabilities in nine of the secure application. MobSF had a similar performance, correctly identifying 18 and erroneously marking 9 applications. AndroBugs the lowest number (13) of vulnerability identifications in benign apps, but it had the just 2 false positive for secure applications.

We can also see that the different tools performed well on different categories of the benchmark. The Qark and MobSF had most benign application detections in the ICC category, but MobSF had lower number of false positives for secure applications. MobSF suffers from wrongly identifying the secure applications in other categories while Qark performs relatively well. AndroBugs is good at detecting Network, Web and ICC categories which are all related to communication with external agents. The overall count for each categorized in Table 13.

Each benign application identified is a true positive, and the ones failed to be identified are false negatives. The secure applications where the vulnerability is detected is a false negative, otherwise they are true negatives. We can calculate the F-score as summarized in Table 14.

Table 13. Ghera count by category

Category	Qark		Androbugs		MobSF	
	Benign	Secure	Benign	Secure	Benign	Secure
Crypto	1	0	1	1	2	1
ICC	7	6	3	0	7	1
Networking	3	0	4	1	2	2
NonAPI	0	0	0	0	0	0
Permission	0	0	1	0	1	0
Storage	2	0	0	0	2	2
System	3	2	1	0	1	1
Web	3	1	3	1	3	2
Total	19	9	13	3	18	9

Table 14. Precision, Recall, and F-Score for Ghera Benchmark Results

Tool	**Precision**	**Recall**	**F-Score**
MobSF	0.66666667	0.29508197	0.40909091
Qark	0.67857143	0.31147541	0.42696629
AndroBugs	0.8125	0.21311475	0.33766234

To calculate the trust of the tools we treat the positively identified benign apps as positive evidence, and the negatively identified secure apps as negative evidences. Thus, we get ω_{mobsf} = (0.6207,0.3103,0.0690), ω_{qark} = (0.6333,0.3000,0.0667) and ω_{andro} = (0.7222,0.1667,0.1111).

4.2 Individual Tool Results

In this section, we analyze the results from the three basic ranking schemes for each tool individually. Then, we evaluate the correlation between the rankings generated by different tools and different schemes.

4.2.1 **MobSF Results**

The results on the 5 categories having 5 app each are show in Table 15.

Table 15. MobSF Summary

Category	**App**	**BDU (B1)**	**Rank (B1)**	**BDU (B2)**	**Rank (B2)**	**BDU (B3)**	**Rank (B3)**
finance	App 1	(0.1, 0.8, 0.1)	3	(0.0048, 0.9933, 0.0019)	3	(0.0065, 0.9916, 0.0019)	3
	App 2	(0.1, 0.7, 0.2)	1	(0.0526, 0.8421, 0.1053)	1	(0.0725, 0.8222, 0.1053)	1
	App 3	(0.1, 0.7, 0.2)	1	(0.0045, 0.9864, 0.0091)	2	(0.0052, 0.9857, 0.0091)	2
	App 4	(0.0625, 0.8125, 0.125)	5	(0.0026, 0.9948, 0.0026)	4	(0.003, 0.9944, 0.0026)	4
	App 5	(0.1, 0.8, 0.1)	3	(0.0018, 0.9976, 0.0006)	5	(0.0039, 0.9955, 0.0006)	5
insurance	App 1	(0.0, 0.8, 0.2)	4	(0.0, 0.9944, 0.0056)	4	(0.0, 0.9944, 0.0056)	5
	App 2	(0.1053, 0.7895, 0.1053)	2	(0.0064, 0.9918, 0.0018)	2	(0.0092, 0.989, 0.0018)	2
	App 3	(0.0, 0.8667, 0.1333)	5	(0.0, 0.9937, 0.0063)	3	(0.0, 0.9937, 0.0063)	3
	App 4	(0.125, 0.75, 0.125)	1	(0.0223, 0.9665, 0.0112)	1	(0.0237, 0.9651, 0.0112)	1
	App 5	(0.1, 0.8, 0.1)	3	(0.0025, 0.9949, 0.0025)	5	(0.0031, 0.9944, 0.0025)	4
news	App 1	(0.0667, 0.8, 0.1333)	2	(0.0033, 0.99, 0.0067)	4	(0.0046, 0.9887, 0.0067)	4
	App 2	(0.0833, 0.75, 0.1667)	1	(0.0109, 0.9674, 0.0217)	1	(0.0161, 0.9622, 0.0217)	1
	App 3	(0.0625, 0.8125, 0.125)	4	(0.0025, 0.9959, 0.0016)	5	(0.0037, 0.9947, 0.0016)	5
	App 4	(0.0556, 0.8333, 0.1111)	5	(0.0079, 0.9868, 0.0053)	3	(0.0089, 0.9858, 0.0053)	2
	App 5	(0.0667, 0.8, 0.1333)	2	(0.0047, 0.986, 0.0093)	2	(0.005, 0.9857, 0.0093)	2
shopping	App 1	(0.1053, 0.7895, 0.1053)	2	(0.0034, 0.9932, 0.0034)	3	(0.004, 0.9926, 0.0034)	3
	App 2	(0.0588, 0.8235, 0.1176)	4	(0.0031, 0.9938, 0.0031)	4	(0.0043, 0.9927, 0.0031)	4
	App 3	(0.0667, 0.8, 0.1333)	3	(0.0007, 0.998, 0.0013)	5	(0.0014, 0.9973, 0.0013)	5
	App 4	(0.0, 0.75, 0.25)	1	(0.0, 0.977, 0.023)	1	(0.0, 0.977, 0.023)	1
	App 5	(0.0556, 0.8333, 0.1111)	5	(0.0082, 0.9886, 0.0033)	2	(0.0127, 0.984, 0.0033)	2
travel	App 1	(0.0833, 0.75, 0.1667)	1	(0.0031, 0.9937, 0.0031)	1	(0.0056, 0.9913, 0.0031)	1
	App 2	(0.0667, 0.8, 0.1333)	3	(0.0011, 0.9967, 0.0022)	5	(0.0017, 0.9962, 0.0022)	5
	App 3	(0.0833, 0.75, 0.1667)	1	(0.0031, 0.9937, 0.0031)	1	(0.0056, 0.9913, 0.0031)	2
	App 4	(0.0667, 0.8, 0.1333)	3	(0.0032, 0.9946, 0.0021)	3	(0.0049, 0.9929, 0.0021)	3
	App 5	(0.05, 0.85, 0.1)	5	(0.003, 0.9951, 0.002)	4	(0.0041, 0.9939, 0.002)	4

We can see that the Base Scheme B1 has a lot of ties. This is expected because both applications may have different vulnerabilities, but the number of categories reported may be the same. Base Schemes B2 and B3 have similar higher values of ‘d’ because MobSF does not report counts for good practices. This leads to higher level of disbelief in the latter two schemes.

The schemes B2 and B3 have similar rankings. This is due to the fact that most of the time some common vulnerabilities are reported for the applications.

We can use the Kendall Tau correlation to calculate the similarity between the schemes. The following table demonstrates the correlation between the schemes on all the datasets including games, photography and tools.

Table 16. Kendall Tau’s Correlation for MobSF

category	**B1 vs B2**	**B2 vs B3**	**B3 vs B1**
finance	0.670820393	1	0.670820393
insurance	0.4	0.948683298	0.527046277
shopping	0.2	1	0.2
news	0.527046277	0.948683298	0.444444444
travel	0.707106781	1	0.707106781
games	0.648099549	0.996512642	0.651685162
photography	0.564841854	0.9839968	0.568855313
tools	0.458365524	0.996450754	0.46206202

We can see that the B2 and B3 schemes have very high correlation. There is also a correlation of the two schemes with the B1 scheme. The correlations in the finance, insurance, shopping and news can vary depending on the selection of the apps. The larger datasets give a more precise results for the similarity of rankings.

4.2.2 **AndroBugs Results**

The results for Androbugs is summarized in Table 17.

Table 17. AndroBugs Summary

Category	App	BDU (B1)	Rank (B1)	BDU (B2)	Rank (B2)	BDU (B3)	Rank (B3)
finance	App 1	(0.2174, 0.7391, 0.0435)	2	(0.0725, 0.9171, 0.0104)	5	(0.0567, 0.9329, 0.0104)	5
	App 2	(0.2174, 0.7391, 0.0435)	2	(0.169, 0.8028, 0.0282)	1	(0.1413, 0.8305, 0.0282)	1
	App 3	(0.2222, 0.7333, 0.0444)	1	(0.1087, 0.8696, 0.0217)	2	(0.0917, 0.8866, 0.0217)	2
	App 4	(0.2128, 0.7447, 0.0426)	5	(0.0815, 0.9099, 0.0086)	3	(0.0613, 0.9301, 0.0086)	3
	App 5	(0.2174, 0.7391, 0.0435)	2	(0.0791, 0.9096, 0.0113)	3	(0.0618, 0.9269, 0.0113)	4
insurance	App 1	(0.2128, 0.7447, 0.0426)	2	(0.1905, 0.7778, 0.0317)	1	(0.1562, 0.8121, 0.0317)	1
	App 2	(0.2128, 0.7447, 0.0426)	2	(0.123, 0.8607, 0.0164)	3	(0.0987, 0.8849, 0.0164)	3
	App 3	(0.2128, 0.7447, 0.0426)	2	(0.0886, 0.8987, 0.0127)	4	(0.0691, 0.9182, 0.0127)	4
	App 4	(0.234, 0.7234, 0.0426)	1	(0.1429, 0.8413, 0.0159)	2	(0.1115, 0.8726, 0.0159)	2
	App 5	(0.2, 0.7556, 0.0444)	5	(0.0695, 0.9198, 0.0107)	5	(0.0552, 0.9342, 0.0107)	5
news	App 1	(0.2174, 0.7391, 0.0435)	1	(0.119, 0.8714, 0.0095)	2	(0.0929, 0.8975, 0.0095)	2
	App 2	(0.2128, 0.7447, 0.0426)	3	(0.1218, 0.8654, 0.0128)	1	(0.0929, 0.8943, 0.0128)	1
	App 3	(0.2128, 0.7447, 0.0426)	3	(0.0964, 0.8916, 0.012)	4	(0.0761, 0.9118, 0.012)	4
	App 4	(0.1957, 0.7609, 0.0435)	5	(0.039, 0.9539, 0.0071)	5	(0.0307, 0.9622, 0.0071)	5
	App 5	(0.2174, 0.7391, 0.0435)	1	(0.106, 0.8808, 0.0132)	3	(0.0846, 0.9021, 0.0132)	3
shopping	App 1	(0.2174, 0.7391, 0.0435)	2	(0.1152, 0.8727, 0.0121)	4	(0.0864, 0.9014, 0.0121)	4
	App 2	(0.2128, 0.7447, 0.0426)	3	(0.0938, 0.8958, 0.0104)	5	(0.0719, 0.9177, 0.0104)	5
	App 3	(0.2128, 0.7447, 0.0426)	3	(0.1371, 0.8468, 0.0161)	3	(0.1118, 0.872, 0.0161)	3
	App 4	(0.2128, 0.7447, 0.0426)	3	(0.1887, 0.7736, 0.0377)	1	(0.156, 0.8063, 0.0377)	1
	App 5	(0.234, 0.7234, 0.0426)	1	(0.1538, 0.8308, 0.0154)	2	(0.1244, 0.8602, 0.0154)	2
travel	App 1	(0.2128, 0.7447, 0.0426)	3	(0.125, 0.8611, 0.0139)	1	(0.1008, 0.8853, 0.0139)	1
	App 2	(0.2128, 0.7447, 0.0426)	3	(0.0971, 0.8835, 0.0194)	3	(0.0805, 0.9001, 0.0194)	3
	App 3	(0.2128, 0.7447, 0.0426)	3	(0.125, 0.8611, 0.0139)	1	(0.1008, 0.8853, 0.0139)	1
	App 4	(0.234, 0.7234, 0.0426)	1	(0.0947, 0.8947, 0.0105)	4	(0.0735, 0.9159, 0.0105)	4
	App 5	(0.234, 0.7234, 0.0426)	1	(0.0557, 0.9395, 0.0048)	5	(0.0416, 0.9535, 0.0048)	5

We see results similar to the MobSF for similar reasons. The B1 scheme consists of even larger number of ties because the number of findings reported by AndroBugs is lower. Many applications could have the same number of evidences. The B2 and B3 schemes are highly correlated as demonstrated by the Kendall Tau's correlation calculation in the table below.

Table 18. Kendall Tau's Correlation for Androbugs

category	**B1 vs B2**	**B2 vs B3**	**B3 vs B1**
finance	0.251976315	0.948683298	0.358568583
insurance	0.597614305	1	0.597614305
shopping	0.119522861	1	0.119522861
news	0.447213595	1	0.447213595
travel	-0.816496581	1	-0.816496581
games	-0.365652197	0.948721067	-0.352139418
photography	0.082234745	0.966503591	0.073149194
tools	-0.450951328	0.944591931	-0.437625285

One of the things we can note here is that we see negative correlation between B1 and the other two schemes.

4.2.3 QARK Results

For Qark, many of the dataset in the smaller categories resulted in an error. Thus, we could calculate the b, d, u value only for a subset of the data. The summary of results is demonstrated in the table below:

Table 19. Qark Summary

Category	**App**	**BDU (B1)**	**Rank (B1)**	**BDU (B2)**	**Rank (B2)**	**BDU (B3)**	**Rank (B3)**
finance	App 1						
	App 2	(0.0, 0.7143, 0.2857)	1	(0.0, 0.9286, 0.0714)	1	(0.0, 0.9286, 0.0714)	1
	App 3	(0.0, 0.8462, 0.1538)	2	(0.0, 0.9964, 0.0036)	2	(0.0, 0.9964, 0.0036)	2
	App 4						
	App 5						

Table 19. Continued

insurance	App 1	(0.0, 0.6, 0.4)	1	(0.0, 0.9444, 0.0556)	1	(0.0, 0.9444, 0.0556)	1
	App 2						
	App 3						
	App 4	(0.0, 0.9, 0.1)	2	(0.0, 0.9964, 0.0036)	2	(0.0, 0.9964, 0.0036)	2
	App 5	(0.0, 0.9, 0.1)	2	(0.0, 0.9996, 0.0004)	3	(0.0, 0.9996, 0.0004)	3
news	App 1	(0.0, 0.8667, 0.1333)	1	(0.0, 0.986, 0.014)	1	(0.0, 0.986, 0.014)	1
	App 2						
	App 3	(0.0, 0.9167, 0.0833)	3	(0.0, 0.9989, 0.0011)	4	(0.0, 0.9989, 0.0011)	4
	App 4	(0.0, 0.9167, 0.0833)	3	(0.0, 0.9977, 0.0023)	3	(0.0, 0.9977, 0.0023)	3
	App 5	(0.0, 0.8947, 0.1053)	2	(0.0, 0.9973, 0.0027)	2	(0.0, 0.9973, 0.0027)	2
shopping	App 1	(0.0, 0.9048, 0.0952)	1	(0.0, 0.9984, 0.0016)	4	(0.0, 0.9984, 0.0016)	4
	App 2	(0.0, 0.9091, 0.0909)	2	(0.0, 0.9974, 0.0026)	3	(0.0, 0.9974, 0.0026)	3
	App 3						
	App 4	(0.0, 0.8571, 0.1429)	3	(0.0, 0.9773, 0.0227)	1	(0.0, 0.9773, 0.0227)	1
	App 5	(0.0, 0.8947, 0.1053)	4	(0.0, 0.997, 0.003)	2	(0.0, 0.997, 0.003)	2
travel	App 1	(0.0, 0.8667, 0.1333)	1	(0.0, 0.9988, 0.0012)	2	(0.0, 0.9988, 0.0012)	2
	App 2						
	App 3	(0.0, 0.8667, 0.1333)	1	(0.0, 0.9988, 0.0012)	2	(0.0, 0.9988, 0.0012)	2
	App 4	(0.0, 0.8889, 0.1111)	3	(0.0, 0.9991, 0.0009)	4	(0.0, 0.9991, 0.0009)	4
	App 5	(0.0, 0.913, 0.087)	4	(0.0, 0.9986, 0.0014)	1	(0.0, 0.9986, 0.0014)	1

All the missing data are the applications for which the analysis failed. We can also see that the tuple values on B2 and B3 schemes are exactly the same. This is due to the fact that there are no good practices reported by the tool. We still see a positive correlation with the B1 scheme. This is further demonstrated by the Kendall Tau’s correlation in Table 20.

Table 20. Kendall Tau's Correlation for Qark

category	B1 vs B2	B2 vs B3	B3 vs B1
finance	1	1	1
insurance	0.816496581	1	0.816496581
shopping	0.666666667	1	0.666666667
news	0.912870929	1	0.912870929
travel	-0.2	1	-0.2
games	0.640355391	1	0.640355391
photography	0.606648847	1	0.606648847
tools	0.703807994	1	0.703807994

4.2.4 Correlation of the ranks between the Tools

Besides analyzing the correlation between the schemes, we have also analyzed the correlation between the tools themselves. For each scheme, we can calculate the correlations for the rankings generated by the various tools.

Table 21. Correlation between rankings generated by the tools for B1

category	mobsf vs qark	qark vs androbugs	androbugs vs mobsf
finance	undefined	-1	0.801783726
insurance	-0.816496581	0	0.358568583
shopping	0.333333333	0.182574186	-0.358568583
news	0.8	0.8	0.471404521
travel	1	-0.894427191	-0.721687836
games	0.507690657	-0.408967417	-0.305989558
photography	0.302809776	0.004331948	0.247434671
tools	0.579153007	-0.542180557	-0.275921074

We can see that there is no significant correlation between the rankings generated. There is a slight negative correlation between androbugs and the other two tools. We have seen a similar pattern in the benchmark dataset where MobSF and Qark were reporting large number of true positives and false positives whereas Androbugs is returning only a smaller number of vulnerabilities and good practices. This scheme is similar to benchmark as it only counts the number of categories reported.

Table 22. Correlation between rankings generated by the tools for B2

category	mobsf vs qark	qark vs androbugs	androbugs vs mobsf
finance	1	1	0.527046277
insurance	0.333333333	1	0.4
shopping	0.666666667	0.666666667	0.6
news	0.333333333	0.666666667	0.4
travel	-0.2	-0.2	0.555555556
games	0.658024443	0.522747551	0.585155824
photography	0.533408653	0.25033483	0.293323024
tools	0.557796088	0.367218452	0.305415481

We can see that when the frequency of the findings is included in the scheme, we no longer see a negative correlation between Androbugs and the other two tools.

Table 23. Correlation between rankings generated by the tools for B3

category	mobsf vs qark	qark vs androbugs	androbugs vs mobsf
finance	1	1	0.4
insurance	0	1	0.316227766
shopping	0.666666667	0.666666667	0.6
news	0.182574186	0.666666667	0.316227766
travel	-0.2	-0.2	0.555555556
games	0.658024443	0.521408887	0.587105835
photography	0.539210355	0.263874275	0.292144907
tools	0.561365983	0.396201065	0.323881011

With the ranking scheme B3, we see a similar pattern. The ranking generated by all the tools are slightly correlated to each other.

4.3 Combined tool results

We have evaluated the rankings generated by the individual tools and ranking schemes. In this section, we evaluate the rankings generated by the combined ranking schemes that we had proposed earlier.

4.3.1 Consensus Scheme treating each tool equally (C1) Results

The summary of results on the consensus scheme treating each tool equally is in the table below:

Table 24. C1 Summary

Category	**App**	**BDU (B1)**	**Rank (B1)**	**BDU(B2)**	**Rank (B2)**	**BDU (B3)**	**Rank (B3)**
finance	App 1	(0.1875, 0.7813, 0.0312)	2	(0.0154, 0.983, 0.0016)	4	(0.0144, 0.984, 0.0016)	4
	App 2	(0.1864, 0.7797, 0.0339)	1	(0.114, 0.8684, 0.0175)	1	(0.1001, 0.8824, 0.0175)	1
	App 3	(0.1719, 0.7969, 0.0313)	4	(0.0127, 0.985, 0.0023)	3	(0.011, 0.9867, 0.0023)	3
	App 4	(0.1803, 0.7869, 0.0328)	5	(0.0211, 0.9769, 0.002)	2	(0.0167, 0.9813, 0.002)	2
	App 5	(0.1875, 0.7813, 0.0312)	2	(0.0056, 0.9938, 0.0006)	5	(0.0068, 0.9927, 0.0006)	5
insurance	App 1	(0.1724, 0.7931, 0.0345)	2	(0.0266, 0.969, 0.0044)	2	(0.0218, 0.9738, 0.0044)	2
	App 2	(0.1875, 0.7813, 0.0312)	1	(0.0181, 0.9802, 0.0016)	4	(0.0182, 0.9802, 0.0016)	4
	App 3	(0.1667, 0.8, 0.0333)	3	(0.0297, 0.966, 0.0042)	1	(0.0232, 0.9726, 0.0042)	1
	App 4	(0.1646, 0.8101, 0.0253)	4	(0.0255, 0.9722, 0.0023)	3	(0.0212, 0.9765, 0.0023)	3
	App 5	(0.1358, 0.8395, 0.0247)	5	(0.0026, 0.9971, 0.0003)	5	(0.0022, 0.9975, 0.0003)	5
shopping	App 1	(0.1463, 0.8293, 0.0244)	4	(0.0106, 0.9884, 0.001)	5	(0.0084, 0.9906, 0.001)	5
	App 2	(0.1341, 0.8415, 0.0244)	5	(0.0126, 0.9862, 0.0013)	3	(0.0104, 0.9883, 0.0013)	3
	App 3	(0.1833, 0.7833, 0.0333)	1	(0.0109, 0.9879, 0.0012)	4	(0.0097, 0.9891, 0.0012)	4
	App 4	(0.1538, 0.8154, 0.0308)	2	(0.0446, 0.9464, 0.0089)	1	(0.0369, 0.9542, 0.0089)	1
	App 5	(0.15, 0.825, 0.025)	3	(0.0178, 0.9808, 0.0014)	2	(0.017, 0.9816, 0.0014)	2

Table 24. Continued

news	App 1	(0.1528, 0.8194, 0.0278)	2	(0.0401, 0.9569, 0.0031)	2	(0.0322, 0.9647, 0.0031)	2
	App 2	(0.193, 0.7719, 0.0351)	1	(0.0813, 0.9106, 0.0081)	1	(0.0649, 0.927, 0.0081)	1
	App 3	(0.1325, 0.8434, 0.0241)	4	(0.0059, 0.9935, 0.0006)	5	(0.0053, 0.9941, 0.0006)	5
	App 4	(0.119, 0.8571, 0.0238)	5	(0.0091, 0.9895, 0.0013)	4	(0.0079, 0.9908, 0.0013)	4
	App 5	(0.1447, 0.8289, 0.0263)	3	(0.0153, 0.9829, 0.0018)	3	(0.0125, 0.9857, 0.0018)	3
travel	App 1	(0.1571, 0.8143, 0.0286)	2	(0.0082, 0.991, 0.0008)	3	(0.0074, 0.9918, 0.0008)	3
	App 2	(0.1833, 0.7833, 0.0333)	1	(0.0107, 0.9873, 0.002)	1	(0.0096, 0.9884, 0.002)	1
	App 3	(0.1571, 0.8143, 0.0286)	3	(0.0082, 0.991, 0.0008)	3	(0.0074, 0.9918, 0.0008)	3
	App 4	(0.1579, 0.8158, 0.0263)	4	(0.0063, 0.9931, 0.0006)	5	(0.0055, 0.9939, 0.0006)	5
	App 5	(0.1395, 0.8372, 0.0233)	5	(0.009, 0.9903, 0.0007)	2	(0.0074, 0.9919, 0.0007)	2

We can see that the number of ties is reduced when we combine multiple tools. We can also observe that the rankings generated by B2 and B3 are highly correlated. This is further illustrated by the Kendall Tau's correlation in table below:

Table 25. Kendall Tau's Correlation for C1

category	B1 vs B2	B2 vs B3	B3 vs B1
finance	0.316227766	1	0.316227766
insurance	0.2	1	0.2
shopping	0.2	1	0.2
news	0.8	1	0.8
travel	0.333333333	0.555555556	0.777777778
games	0.637573235	0.982112436	0.635838342
photography	0.663654909	0.983569135	0.673557399
tools	0.510261994	0.964515543	0.491188387

The high correlation between the combined rankings generated using B2 and B3 are confirmed in the larger datasets as well.

4.3.2 Consensus Scheme based on the trust of the tools (C2) Results

For Consensus Scheme that takes the trust of the tools into factoring, the results are summarized in the table below:

Table 26. C2 Summary

Category	**App**	**BDU (B1)**	**Rank (B1)**	**BDU(B2)**	**Rank (B2)**	**BDU (B3)**	**Rank (B3)**
finance	App 1	(0.1675, 0.2888, 0.5437)	1	(0.0411, 0.3742, 0.5848)	3	(0.0335, 0.3774, 0.5891)	3
	App 2	(0.1484, 0.3636, 0.4879)	4	(0.098, 0.4361, 0.4659)	4	(0.0932, 0.4377, 0.4691)	4
	App 3	(0.146, 0.3827, 0.4713)	5	(0.0487, 0.4886, 0.4627)	5	(0.0414, 0.4926, 0.466)	5
	App 4	(0.149, 0.2939, 0.5571)	3	(0.0448, 0.3726, 0.5825)	1	(0.0341, 0.3773, 0.5885)	2
	App 5	(0.1675, 0.2888, 0.5437)	1	(0.043, 0.3737, 0.5833)	2	(0.0348, 0.3772, 0.588)	1
insurance	App 1	(0.11, 0.373, 0.517)	4	(0.0851, 0.4617, 0.4533)	3	(0.0691, 0.4704, 0.4605)	3
	App 2	(0.1676, 0.2873, 0.5451)	1	(0.0692, 0.3612, 0.5697)	1	(0.0574, 0.3662, 0.5764)	1
	App 3	(0.1214, 0.308, 0.5706)	2	(0.0474, 0.3705, 0.5822)	2	(0.0368, 0.3751, 0.5881)	2
	App 4	(0.1573, 0.3937, 0.4489)	3	(0.071, 0.4756, 0.4534)	4	(0.0577, 0.4829, 0.4595)	4
	App 5	(0.1315, 0.412, 0.4565)	5	(0.0306, 0.5007, 0.4687)	5	(0.0246, 0.504, 0.4713)	5
shopping	App 1	(0.1414, 0.4064, 0.4521)	2	(0.051, 0.489, 0.46)	4	(0.0386, 0.4959, 0.4655)	4
	App 2	(0.1217, 0.4179, 0.4604)	5	(0.0414, 0.4943, 0.4642)	5	(0.0324, 0.4993, 0.4683)	5
	App 3	(0.1512, 0.2909, 0.5579)	1	(0.0741, 0.3595, 0.5664)	1	(0.0606, 0.3654, 0.574)	1
	App 4	(0.1046, 0.4038, 0.4916)	4	(0.0839, 0.4638, 0.4523)	2	(0.0687, 0.4722, 0.4591)	2
	App 5	(0.131, 0.4122, 0.4568)	3	(0.0702, 0.4778, 0.452)	3	(0.0588, 0.4839, 0.4572)	3
news	App 1	(0.1286, 0.4059, 0.4655)	2	(0.053, 0.4859, 0.4611)	2	(0.0419, 0.492, 0.4661)	2
	App 2	(0.1599, 0.2789, 0.5611)	1	(0.0712, 0.3558, 0.573)	1	(0.0581, 0.3612, 0.5807)	1
	App 3	(0.123, 0.417, 0.4599)	4	(0.0423, 0.4942, 0.4635)	4	(0.0339, 0.4988, 0.4673)	4
	App 4	(0.112, 0.4251, 0.4629)	5	(0.0198, 0.5061, 0.4741)	5	(0.0167, 0.5077, 0.4756)	5
	App 5	(0.1277, 0.4102, 0.4621)	3	(0.0475, 0.4898, 0.4627)	3	(0.0383, 0.4949, 0.4668)	3

Table 26. Continued

travel	App 1	(0.1336, 0.3977, 0.4687)	2	(0.0553, 0.4866, 0.4581)	2	(0.0455, 0.4919, 0.4625)	2
	App 2	(0.1512, 0.2909, 0.5579)	1	(0.0524, 0.3685, 0.579)	1	(0.0438, 0.3723, 0.5839)	1
	App 3	(0.1336, 0.3977, 0.4687)	2	(0.0553, 0.4866, 0.4581)	2	(0.0455, 0.4919, 0.4625)	2
	App 4	(0.1359, 0.4052, 0.4589)	3	(0.0419, 0.4944, 0.4637)	4	(0.0333, 0.4991, 0.4675)	4
	App 5	(0.128, 0.4181, 0.4539)	4	(0.0248, 0.5042, 0.471)	5	(0.0193, 0.5072, 0.4735)	5

Looking at the BDU-values, we can see that this ranking scheme balances out the low belief values due to larger number of negative evidences. The trust values of the tool calculated using the benchmark results help in achieving this balance.

The Kendall Tau's correlation can be seen in the table below:

Table 27. Kendall Tau's Correlation for C2

category	**B1 vs B2**	**B2 vs B3**	**B3 vs B1**
finance	0.527046277	0.8	0.737864787
insurance	0.8	1	0.8
shopping	0.4	1	0.4
news	1	1	1
travel	1	1	1
games	0.594867537	0.971890971	0.600017905
photography	0.289581722	0.94538187	0.289354926
tools	0.555506202	0.985445567	0.565523043

Similar to the C1 ranking scheme, the consensus based on B2 ranking scheme and B3 ranking scheme are heavily correlated. This can be seen in categories that have lesser number of apps for which we have performed detailed analysis as well as in the larger datasets. There is also a slight correlation between B1 and B2, and B1 and B3 as well.

4.4 Summary

In this chapter, we have evaluated the tools against the benchmark and analyzed the results from the different ranking schemes proposed in this thesis.

In the evaluation of tools against the benchmark, we had seen that all MobSF and Qark tools had similar performances, and AndroBugs had the worst performance. Their F1-scores were 0.41 for MobSF, 0.42 for Qark, and 0.33 for AndroBugs.

The ranking scheme based on only the categories of evidences (B1) generally have lots of ties. This can be primarily attributed to the fact that many apps could contain same number of security vulnerabilities and good practices. This is a really naïve approach and is not recommended to be used.

The ranking scheme based on the frequency of evidences (B2) may provide a better reflection of the app rankings. The rankings still have a slight correlation with ranking scheme B1 due to the fact that the apps with higher number of categories of evidences have higher frequency of evidences in many cases. However, there are cases when apps with smaller or equal number of categories of evidences have those evidences in large frequencies that may alter the rankings drastically.

The ranking scheme based on the severity of evidences (B3) builds upon the ranking scheme B2 and is highly correlated to it. Similar to ranking scheme B2, the ranking scheme B3 also has slight correlation with ranking scheme B1 in most of the cases, and for similar reasons. Many of the apps have similar categories, and many of the categories have similar severity. This means that only when the apps have very distinct severity levels of the evidences, the rankings are altered from ranking scheme B2.

We also looked at two ways to combine the rankings from different tools. In the first case (ranking scheme C1), we considered evidences from all tools equally and in second case (ranking scheme C2), we added weights to the tools based on its performance on benchmark. We noticed that the trends we saw while using the tools individually were present in the combined rankings as well.

After comparing the different ranking schemes, we recommend using the ranking scheme B3 as it takes the most criteria into consideration. Though it is similar to ranking scheme B2, it is a more accurate reflection of the security vulnerabilities and good practices present in an app.

Depending on a user's preference of tools, the rankings can be generated using a single tool or a combination of tools. The results for the combined rankings consider all three analysis tools but depending on the relevance of tool for a user, we could also use only a subset of the three tools or add more tools.

There are no established rankings related to the security of the apps that can be considered as the ground truth to compare against. There are third party app rankings such as AppBrain [47], and AppAnnie [48] which provide rankings based on the number of downloads. Other efforts carried out by Chowdhury et al. [40] [41] [42] and Gallege et al. [49] combine evidences obtained both from the code analysis and user reviews. Since we are evaluating only the security evidences, it would not be appropriate to compare our ranking schemes to these other approaches. The ranking comparison, in addition, would require the use of same dataset – which is not the case with the other approaches. Thus, we have limited the correlation calculation to the different ranking schemes proposed in this research.

CHAPTER 5. CONCLUSION AND FUTURE WORK

This thesis has proposed several ranking schemes for ranking Android apps based on the different security evidences generated by static analysis tools. Three ranking schemes were proposed for ranking apps based on number of categories of the evidences, the frequency of the evidences, and the severity of the evidences. The results from the proposed ranking schemes were evaluated, and the ranking scheme based on the severity of evidences is the recommended ranking scheme, because it considers most factor for ranking the apps.

Two additional ranking schemes were also proposed to combine rankings from different tools. The first combined ranking scheme combines the tools considering each tool to be equally weighted, and the second combined ranking scheme considers the weightage of tools based on the performance of the tools in the benchmark. The combined ranking schemes can be used to generate a combined ranking for any of the base ranking models for individual tools.

In summary, the major contributions of the thesis are as follows:

- This research evaluated each tool individually to identify the findings reported by the tool and determine the CVSS score for each finding that is reported by the tools.
- Different static analysis tools available to evaluate android apps were identified.
- The benchmarks used in the research community for the static analysis tool for Android were identified and the Ghera benchmark was selected for evaluating the tools.
- The static code analysis tools were evaluated against the Ghera benchmark.
- Subjective Logic was used to represent the evaluation of the apps and ranking them.
- Three rankings schemes were proposed and the results for each tool were evaluated for each ranking scheme.
- Results for two additional schemes to combined rankings from different tools were evaluated.
- The ranking scheme based on the severity was the recommended scheme as it took most criteria into factoring, but other schemes may still be used depending on a user's preference. A user can also use rankings from a single tool or a combination of tools based on preference.

There are several limitations to the work in this thesis. The limitations that were identified were as follows:

- The CVSS score used in this research is generally determined by the security experts for each vulnerability identified in an app. We have generalized to apply the CVSS for the vulnerabilities identified by the tools without looking at each app based on the description, and the data from the NVD database.
- The CVSS score was also calculated for the good practices based on the vulnerabilities it tries to mitigate. The CVSS score is originally designed to be assigned for vulnerabilities only.
- The static code analysis tools tend to have lots of false positives, and we have tried to mitigate it by evaluating the tools against the benchmark. But there are still many findings that could be false positives while calculating the rankings.
- Qark and AndroBugs have not been updated recently and may not detect vulnerabilities introduced in newer versions of Android

This work provides a base for many future endeavors. This work can be extended to a larger dataset of apps and across other categories. The set of tools can also be expanded as new tools are developed. New schemes can also be developed by identifying other criteria for evaluating apps. As there is more research on holistic ranking of apps, the work in this thesis can be integrated to take the security into consideration.

REFERENCES

[1] PewResearch.Org, "Mobile Fact Sheet," Pew Research Center, 12 June 2019. [Online]. Available: https://www.pewresearch.org/internet/fact-sheet/mobile/.

[2] StatCounter, "Mobile Operating System Market Share Worldwide," [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide. [Accessed 14 January 2020].

[3] AppBrain, "Number of Android apps on Google Play," AppBrain, [Online]. Available: https://www.appbrain.com/stats/number-of-android-apps. [Accessed 15 November 2020].

[4] OWASP.org, "OWASP Mobile Top Ten," OWASP, 2016. [Online]. Available: https://www.owasp.org/index.php/OWASP_Mobile_Top_10. [Accessed 14 January 2020].

[5] MITRE.org, "CWE-VIEW: Weaknesses in Mobile Applications," MITRE.org, 29 05 2013. [Online]. Available: https://cwe.mitre.org/data/definitions/919.html. [Accessed 14 January 2020].

[6] B. Johnson, Y. Song and E. Murphy-Hill, "Why don't software developers use static analysis tools to find bugs?," in *2013 35th Internation Conference on Software engineering (ICSE)*, San Francisco, 2013.

[7] J. Mitra and V.-P. Ranganath, "Ghera: A Repository of Android App Vulnerability Benchmarks," in *Proceedings of PROMISE*, Toronto, 2017.

[8] A. Jøsang, "A logic for uncertain probabilities," *International Journal of Uncertainity, Fuzziness and Knowledge-Based Systems,* vol. 9, no. 3, 2001.

[9] Android Developers, "Application Fundamentals," Google Developers, [Online]. Available: https://developer.android.com/guide/components/fundamentals. [Accessed 14 January 2019].

[10] Android Developers, "Introduction to Activities," Google Developers, [Online]. Available: https://developer.android.com/guide/components/activities/intro-activities. [Accessed 14 January 2020].

[11] Android Developers, "Services Overview," Google, [Online]. Available: https://developer.android.com/guide/components/services. [Accessed 14 January 2019].

[12] Android Developers, "Broadcasts Overview," Google Developers, [Online]. Available: https://developer.android.com/guide/components/broadcasts. [Accessed 14 January 2020].

[13] Android Developers, "Content Providers," Google Developers, [Online]. Available: https://developer.android.com/guide/topics/providers/content-providers. [Accessed 14 January 2020].

[14] Android Developers, "Intents and Intent Filters," Google Developers, [Online]. Available: https://developer.android.com/guide/components/intents-filters. [Accessed 14 January 2020].

[15] A. Lockwood, "Content providers and Content Resolvers," Android Design Patterns, 25 06 2012. [Online]. Available: https://www.androiddesignpatterns.com/2012/06/content-resolvers-and-content-providers.html. [Accessed 14 January 2020].

[16] Android Developers, "Permissions Overview," Google Developers, [Online]. Available: https://developer.android.com/guide/topics/permissions/overview. [Accessed 14 January 2020].

[17] Android Developers, "App Manifest Overview," Google Developers, [Online]. Available: https://developer.android.com/guide/topics/manifest/manifest-intro. [Accessed 14 January 2020].

[18] "Dex2Jar," [Online]. Available: https://github.com/pxb1988/dex2jar. [Accessed 2 March 2020].

[19] "jadx," [Online]. Available: https://github.com/skylot/jadx. [Accessed 2 March 2020].

[20] FIRST.org, "CVSS 3.0 Specification Document," FIRST.org, [Online]. Available: https://www.first.org/cvss/v3.0/specification-document. [Accessed 14 January 2020].

[21] NIST.gov, "National Vulnerability Database (NVD)," Information Technology Laboratory, [Online]. Available: https://nvd.nist.gov/. [Accessed 2 February 2020].

[22] FIRST.org, "CVSS 3.0 Calculator," [Online]. Available: https://www.first.org/cvss/calculator/3.0. [Accessed 14 January 2020].

[23] LinkedIn, "QARK," [Online]. Available: https://github.com/linkedin/qark.

[24] "Androbugs Framework," [Online]. Available: https://github.com/AndroBugs/AndroBugs_Framework.

[25] "Mobile Security Framework," [Online]. Available: https://github.com/MobSF/Mobile-Security-Framework-MobSF.

[26] "JAADAS," [Online]. Available: https://github.com/flankerhqd/JAADAS.

[27] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau and P. McDaniel, "FlowDroid: precise, context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps," *ACM SIGPLAN Notices,* vol. 49, no. 6, 2014.

[28] M. Gordon, K. Deokhwan, J. Perkins, L. Gilham, N. Nguyen and M. Rinard, "Information-Flow Analysis of Android Applications in DroidSafe," in *NDSS Symposium 2015*, San Diego, 2015.

[29] F. Wei, S. Roy, X. Ou and R. , "Amandroid: A Precise and General Inter-Component Data Flow Analysis Framework for Security Vetting of Android Apps," *ACM Transactions on Privacy and Security,* vol. 21, no. 3, 2018.

[30] Payatu, "DIVA Android," [Online]. Available: https://github.com/payatu/diva-android. [Accessed 4 August 2020].

[31] H.-T. B. (ImmuniWeb), "Purposefully Insecure and Vulnerable android Application," [Online]. Available: https://github.com/HTBridge/pivaa. [Accessed 4 August 2020].

[32] C. Fritz, S. Arzt and S. Rasthofer, "DroidBench 2.0," [Online]. Available: https://github.com/secure-software-engineering/DroidBench. [Accessed 4 August 2020].

[33] F. Wei, "ICC-Bench," [Online]. Available: https://github.com/fgwei/ICC-Bench.

[34] A. Bosu, "DialDroid-Bench," [Online]. Available: https://github.com/amiangshu/dialdroid-bench. [Accessed 4 August 2020].

[35] A. Jøsang, "Aritifical Reasoning with Subjective Logic," in *2nd Australian Workshop on Commonsense Reasoning*, Perth, 1997.

[36] H. Zhou, W. Shi, Z. Liang and B. Liang, "Using new fusion operations to improve trust expresiveness of subjective logic," *Wuhan University Journal of Natural Sciences,* vol. 16, 2011.

[37] D. Ceolin, P. Groth and W. R. van Hage, "Calculating the Trust of Event Description using Provenance".

[38] H. Abdi, "The Kendall Rank Correlation Coefficient," in *Encyclopedia of Measurement and Statistics*, Thousand Oaks, Sage, 2007.

[39] SciPy.org, "SciPy Reference," SciPy.org, [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kendalltau.html#r4cd1899fa369-2. [Accessed 24 February 2020].

[40] AppBrain, "Google Play Ranking," AppBrain, [Online]. Available: https://www.appbrain.com/stats/google-play-rankings. [Accessed 1 December 2020].

[41] N. S. Chowdhury and R. R. Raje, "A Holistic Ranking Scheme for Apps," in *2018 21st International Conference of Computer and Information Technology (ICCIT)*, 2018.

[42] N. S. Chowdhury and R. R. Raje, "Disparity between the programmatic views and the user perceptions of mobile apps," in *2017 20th International Conference of Computer and Information Technology (ICCIT)*, 2017.

[43] N. S. Chowdhury and R. R. Raje, "SERS: A Security-related and Evidence-based RankingScheme for Mobile Apps," in *The First IEEE International Conference on Trust,Privacy and Security in Intelligent Systems, and Applications*, 2019.

[44] L. S. Gallege and R. Raje, "Parallel methods for evidence and trust based selection and recommendation of software apps from online marketplaces," in *Proceedings of the 12th Annual Conference on Cyber and Information Security Research*, Oak Ridge, 2017.

[45] L. Qiu, Y. Wang and J. Rubin, "Analyzing the analyzers: FlowDroid/IccTa, Amandroid and DroidSafe," in *ISSTA 2018: Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2018.

[46] F. Pauck, E. Bodden and H. Wehrheim, "Do Android Taint Analysis Tools Keep Their Promises," in *ESEC/FSE 2018: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018.

[47] B. Rashidi and C. J. Fung, "A Survey of Android Security Threats and Defenses," *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable,* vol. 6, pp. 3-35, 2015.

[48] "APKPure," [Online]. Available: https://apkpure.com/. [Accessed 2 March 2020].

[49] AppAnnie, "Top Apps on Google Play," AppAnnie, [Online]. Available: https://www.appannie.com/en/apps/google-play/top/. [Accessed 1 December 2020].