Android Application Collusion Attacks

Analysis

By

Hafiz Muhammad Arslan Maqsood

A thesis submitted to the faculty of Information Security Department, Military College of Signals, National University of Sciences and Technology, Rawalpindi in partial fulfilment of the requirements for the degree of MS in Information Security

December 2018

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS Thesis written by **Hafiz Muhammad Arslan Maqsood** Registration No. **00000199401**, of **Military College of Signals** has been vetted by undersigned, found complete in all respect as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial, fulfillment for award of MS degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have been also incorporated in the said thesis.

Signature: _____

Name of Supervisor:_____

Date: _____

Signature (HOD):_____

Date: _____

Signature (Dean/Principal):_____

Date: _____

Declaration

I hereby declare that no portion of work presented in this thesis has been submitted in support of another award or qualification either at this institution or elsewhere

Dedication

This thesis is dedicated to MY FAMILY, TEACHERS AND FRIENDS for their love, endless support and encouragement

Acknowledgement

I would like to convey my gratitude to my supervisor, Maj (R) Muhammad Faisal Amjad, PhD, for his supervision and constant support. His invaluable help of constructive comments and suggestions throughout the experimental and thesis work are major contributions to the success of this research. Also, I would thank my committee members; Asst Prof Mian Muhammad Waseem Iqbal and Lect Madam Narmeen Shafqat for their support.

I am thankful to my colleagues Lt Cdr Kaleem Ullah PN, Maj Shahid Rafiq, Mr. Asad Mehdi, Mr. M Abdul Rahman and Mr. Adeel Shah who helped me out during my thesis work.

Copyright Notice

Abstract

Android applications can bypass current security model of Android OS, when working together which is said to be application collusion. Android has no special check for inter application communication. These capabilities can easily go unnoticed because only individual application's permissions are shown. To overcome this weakness we developed a mechanism which can detect intent based inter-application data flow and found if there is any collusion between applications. Our main focus is to detect data flow between applications and also extracts data which is being sent between applications. We develop a tool, IADF Analyzer, which tells us which applications are sending data to each other. After getting this information, we manually check if receiving application has permissions to receive that data. As a proof of concept, we analyse a potentially vulnerable application i.e. Sieve developed by MWRSecurity [15]. While analyzing real world application from Google Play store we found one activity of GroupMe application is prone to privacy leakage. IADF Analyzer can be used by developers and security analysts while developing or analysing applications, respectively.

Contents

List of Figures

List of Tables

List of Tables

Chapter 1

Introduction

Android is exponentially growing mobile OS in market. Android is open source and based on Linux kernel. In addition, android further launched their software for TV and wearable devices. Android was first launched in 2007 and released 8 versions till now. Currently android Oreo 8.1 is the latest release. Android Open Source Project (AOSP) is the core source code which can be customize under Apache License.

Android has a large user base, there are two billion active users in May, 2017 and 3.3 million apps featured in Google Play store. Android is declared as the bestselling mobile OS according to Statista [1] .

As Android is being widely accepted by big mobile devices manufacturers. Basic concept of android is to reuse application components to reduce system processing, increase efficiency and to minimise developer's burden. Due to this, apps can share their features to other apps. This feature of android can be misused by developers to share sensitive user or system data [7].

1.1 Application Fundamentals

Android apps can be written using Kotlin, Java, and C++ languages [2] . Android builtin SDK tools automatically compile code and any other data associated with APK, an Android package. All the resources and classes files are packaged to one archived file named as APK. Android run all the applications in separate sandboxes and protect all the following features:

- Each application is treated as a different user in Android, as Linux treats each user.
- By default, each application is assigned a different user ID, known to only system.
- All applications run in isolation. Android create separate VM for each application which start its own Linux process which starts when application starts and ends its operation when user close the application.

The Android system follows Linux for implementation of the principle of least privilege. That is, by default, each application has only access to the components that are required for a particular task and no more. This feature creates a very secure environment, so that each application should have permission to access the resources. However, Android offer ways to share data between applications

1.2 Application Components

Android applications have different components which are considered as the building block of android application. User can interact with each component physically or by implementing some piece of code. Four different types of components are explained below:

- Services
- Activities
- Content providers
- Broadcast receivers

Below is the explanation of each component:

1.2.1 Activity

Activity is the entry point of an application. It act as user interface where user can interact with application and perform specific task.

1.3 The Manifest File

Android manifest is the main and very important file. It consider as the backbone of application. Before starting any operation on application, android first check the properties and permission which a developer has mentioned in manifest. After this consultation, android start any procedure.

- Mention the list of permissions, for example Internet access or read-access to the user's contacts
- In manifest developer should mention API level for application is designed
- Declares all the features which are provided by the application.

1.4 Android Security Model

Android framework depends on three basic security features:

- Permissions
- App Signing
- Sandbox

1.4.1 Permissions

Android is permission based OS. If an app wants to access system resources or user data then it needs to ask for proper permissions. Developer should have to mention every permission in manifest file before using it in app. For Example, if app developer wants to access location information of user then he should mention

android.permission.ACCESS_COARSE_LOCATION

OR

android.permission.ACCESS_FINE_LOCATION in manifest.

1.4.2 Inter Component Communication Channels

There are two channel through which inter app communication is possible.

- Overt Channels
- Covert Channels

1.4.2.1 Overt Channels

Overt channels are legal channels provided by android to communicate with other apps. Some overt channels are mentioned below:

- Intents
- Content Providers (Databases)
- Remote Method calls

1.4.2.2 Covert Channels

Covert channels are illegal practices used by hackers or security analysts to fetch private data. These channels include:

- Timing Channels
- Storage Channels

1.5 Problem Statement

As each application should have permission to access resources. But to communicate with other applications there is no mechanism to check for permissions. A malicious application developer can misuse this mechanism to share sensitive information between apps. For Example, if one application has location information then he can send this information to other application through intents or databases without any special permission. Secondly, there is no comprehensive framework which can be follow by security analyst to analyze inter application communication.

1.6 Methodology

Our main focus is to detect data flow between applications and also extracts data which is being sent between applications. We divided our work into two parts, one is data flow analysis and other other is validation by using existing tools.

For data flow analysis, we first reverse engineer all apps one by one by using dex2jar and decompiler, will be discuss later in this chapter. After reverse engineering, we extract intent based features and save that data into CSV. In second part, we reverse engineer manifest file to human readable format then extract features from activities. After extracting intent-filter from activities, cross match intents to check that which activities are sharing data without permissions.

1.7 Thesis Contribution

To the best of our knowledge, the proposed technique in this thesis is not been used previously. Moreover, we validate our work by using existing tools named as drozer.

Main Contribution of our work is as follows:

- First, we reverse engineer each app to extract intents data which is being sent by one app to another.
- Second, drozer (application analyses framework) is used to verify that which activity is sending or receiving sensitive data.

1.8 Thesis Outline

This thesis is divided into five chapters:

- Chapter 1: This chapter introduces the topic, describes research objectives and importance of topic to the national needs. It also highlights contributions of this research.
- Chapter 2: Contains literature review of inter app commination. In which researchers mentioned threats mechanisms and possible prevention mechanisms
- Chapter 3: Proposed mechanism used to analyze data flow between apps and the working of our custom software
- Chapter 4: Contains the Analysis of potentially vulnerable app called sieve developed by MWRinfosecurity as a proof of concept.

- Chapter 5: Covers the analysis of real world application downloaded from Google Play store.

- Chapter 6: Conclude our thesis work and contains proposal for future work.

Chapter 2

Literature Review

Many techniques are developed by security researchers to detect and mitigate application collusion attacks. Research use taint analysis, data flow analysis and source and sink analysis to detect which app is sending data to other app [5].

In this chapter, we will review related work done by other researchers previously. Mainly, we will cover their work, techniques and future work [6]. Below is the detail of work done by other researchers:

2.1 SEALANT

SEALANT is an online software developed to detect inter app communication and identify vulnerabilities in ICCs. Moreover, a user can control those ICC channels on runtime. SEALANT follow each ICC path based on sender, receiver and intents. It matches sending data with receiving data, if there is any matched it consider it as a path. Receiving app mentions that what type of data can be received by an app, it sending data

is being matched by any intent filter then it mark it as ICC. Second main feature of this software is that it visualizes the ICC paths. This feature is helpful for analyst to visualize malicious paths quickly. There are two main parts pf SEALANT, Analyzer and Interceptor. [4]

2.1.1 Analzer

Analyzer perform static analysis of targeted apps and extract features which is called ICC paths. After extracting all these paths, Analyzer mark vulnerable paths. At the end Analyzer generates a list of vulnerable paths to further match with intents on runtime. Analyzer has four further modules listed below:

- Analyze target apps
- Build ICC Graph
- Find Vulnerable paths
- Generate SEALANT list

2.1.2 Interceptor

Interceptor module monitors and analyzes the vulnerable ICC paths. It analyzes each instance of ICC, whenever ICC is requested, it checks ICC path with the pre generated paths list. If that path is available in pre-generated list, then it is marked as vulnerable. Interceptor is further divided into:

- Blocker

- ChoiceDataBase
- List Provider

2.1.3 Limitations

There are several limitations with this method. It does not work if code is dynamically loading. Secondly app collusion attacks can be performed by covert channels like file system sharing. SEALANT do not cover these types of attacks. In future, by combining SEALANT with kernel level solutions may cover these weaknesses.

2.2 ComDroid

ComDroid detects application communication and find vulnerabilities by analyzing message passing mechanisms. It divides its work into two modules. One is Intent analysis and other is component analysis. It statically analyze intents and perform intra procedural data flow analysis. It extracts features like Action, flags set, categories and extra data [8]. Second part of this software analyses manifest file. It extracts each activity and its associated data. Component analysis part treats each activity as a separate part.

There are some limitation in this software. ComDroid tracks each intent and do not differentiates if and switch statements. ComDroid generates warning but not verifying the existence of vulnerability. Analyzing through this software do not infer the intention of developer.

2.3 PermissionFlow

PermissionFlow statically detect the unauthorized access to permission protected information. They claim that PermissionFlow can detect three types of attacks, intent spoofing, confused deputy and permission collusion [11].

Three core modules Permission Mapper, Rules generator and Decision maker collaborate each other to detect vulnerabilities. Permission Mapper detects which component requires permission. Second module, rule generator generates rules for tainting and decision maker decides that which which component is vulnerable.

PermissionFlow generates many false positives because of repeated access of resources which requires permission. Secondly it gives false negatives for those apps which are transferring protected information.

2.4 FUSE

FUSE is another static analysis tool which analyze single app leveraging lint tool to extend its operation to multi-apps analysis. First part of FUSE analyze single app then based on security policies it analyses multi-apps. [4]

It analyses single app depending on its manifest data structure. Extended manifest can generate information flow graphs from sources to sinks. In second step, for multi app analysis, it collects all the manifest data to generate inter app information flow.

2.5 IccTA

For taint analysis [10] IccTA is software which claims that it is context aware taint analysis tool. IccTA take APKs as input and converts it into intermediate representation called jimple. After converting to intermediate representation if generates ICC graphs and related information. For extracting ICC links, IccTA identifies sources and target components. For control flow graph analysis, IccTA leverages FlowDroid [9]. IccTA generates report for privacy leakage and store information in databases for further use. [5]

Chapter 3

Methodology

3.1 Overview

In this chapter, we discuss our methodology and technique. We divided our work into two parts, one is data flow analysis and other other is validation by using existing tools.

For data flow analysis, First we reverse engineer all applicatios one by one by using dex2jar and procyon decompiler, will be discuss later in this chapter. After reverse engineer we extract data sharing between activities and save that data into CSV.

Second, we reverse manifest file to human readable format then extracts features from activities. After extracting intent-filter from activities, cross match intents to check that which activity is communicating or sending data to which app.

Our customized software is based on Python 3.7 which is depending on dex2jar [12], APKTool [13], and Procyon decompiler [14].

3.2 Dependencies

- Dex2jar
- APKTool
- Procyon Decompiler

3.2.1 Dex2jar

Dex2jar [12] is a tool that converts android .dex files to jar file or java .class files to jar files. Those jar file can further be decompiled to convert code into human readable format.

3.2.1.1 APKTool [13]

A reverse engineering tool for APK files. It converts resources to nearly original format. User can rebuild the package after making some changes. It is developed by Sourcetoad.

3.2.1.2 Procyon Decompiler

A decompiler developed by PROCYON [14] is used in this project. This decompiler efficiently converts jar files to java file which can be easily readable and modifiable by human beings. There are following components of this suit.

1. Core Framework
2. Decompiler

3.3 Flow Chart

Below is the flow chart of technique which we follow to analyze application collusion.

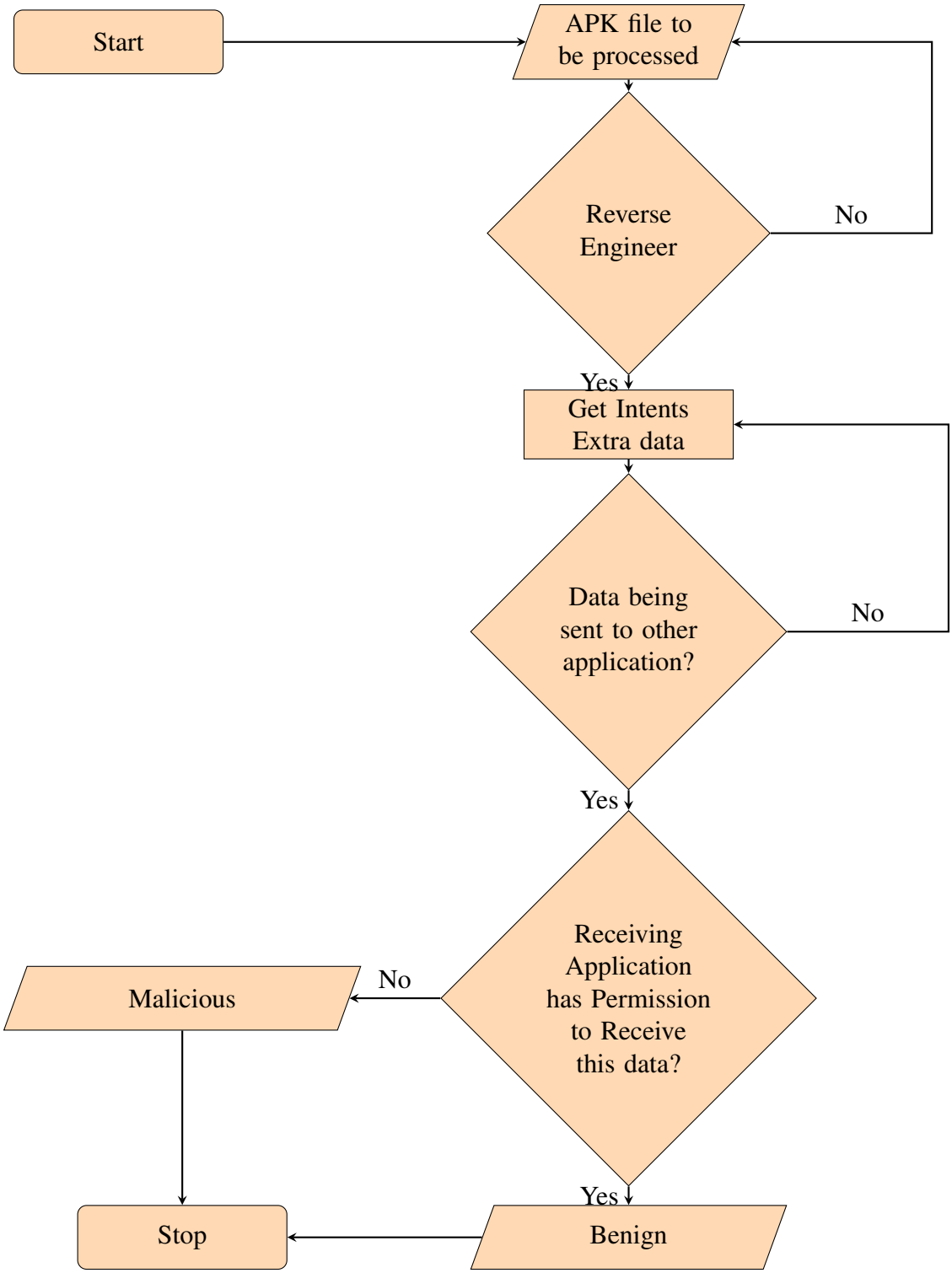


Figure 3.1: Methodology

3.4 Reverse Engineering APK Flow Chart

Reversing an APK file contains 3 steps leveraging 3 tools to effectively decompile APKs for further use.

Steps

1. First, a command line tool ?unzip? is used to unzip .apk file to extract resources
2. Then dex2jar software is used to convert classes.dex files to .jar file
3. At last, Procyon decompiler is used to reverse jar files to java file which will then further used to extract features of our requirement.

3.5 Manifest Extractor

Module used for extracting manifest file?s data is divided into two step. First we used APKTool to get manifest file. APKTool is a software used to reverse engineer APK file to nearly original format. We use APKTool to get manifest file because it converts manifest binary to human readable format.

Second, we develop a tool which further extracts activities intent-filter data and save it as CSV file.

Below are the steps: **Steps**

1. Convert manifest file using APKTool
2. Extract intent-filter?s data
3. Save into CSV file

3.6 Intent Data Extractor

Data Flow analysis module is developed to extract intent based data which is being sent from one app to another app. First this module reverses the app then extracts features.

Following are the steps used by this module:

Steps

1. Dex2jar is used to convert Classes.dex files of APKs into jar format.
2. Jar files needs to be convert to java format. Procyon Decompiler is used for this purpose.
3. Extract data of each intent which is being used to start activity.
4. Save data to CSV file

Chapter 4

Analysis of SIEVE using DROZER

4.1 SIEVE

Sieve is the training application which is made for learning purposes containing intentional vulnerabilities [15]. There are many such applications available online. We selected sieve because it suites our interest. Sieve has many vulnerabilities regarding inter app communication.

Sieve is basically a password manager app which allows users to save any online services? username and password. It encrypts all passwords in database by using master password and PIN provided by user. This app seems secure apparently but we can bypass this app in many ways as discussed in detail later in this chapter.

NOTE: All passwords used in this app are hypothetical just to clear the concept.

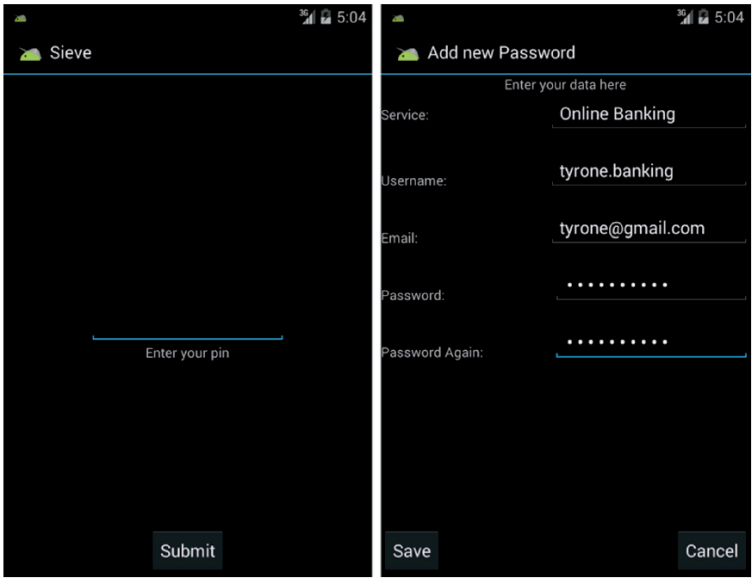


Figure 4.1: Sieve Launch Activity

4.2 DROZER

Drozer is an open source framework for analyzing android applications available on github [15]. We can extend its functionality by importing it into our python project. Drozer can be used from command line to analyze applications on our live environment i.e. Mobile phone.

4.3 Setting up Environment

Following are the requirements for drozer to analyze apps on device.

- Drozer
- Agent
- Sieve

- ADB [16]
- Java [17]

4.4 Running Drozer

Below four steps are required for drozer to run.

1. Open agent app and ON the server
2. Open terminal on PC and connect server by using command (Adb forward tcp:31415 tcp:31415)
3. Open drozer console (drozer console connect)

Now drozer is in working. We can analyze our app i.e. sieve

4.5 Analyzing SIEVE

Each component i.e. Activity, services, broadcast receivers and content providers are analyzed separately.

4.5.1 Exploiting Activities

Exploiting activities through drozer requires following steps:

Steps

1. Check Attack surface

2. Get exported activities info

3. Start activity

NOTE: Not every activity is demonstrated here due to keep thesis work neat and easy to understand. Only vulnerable activity is demonstrated

4.5.1.1 Demonstration

First we check attack surface. In android applications, attack surface means that which activity is exported and what requirements it mentioned in manifest file. We can get this information in by using following command:

```
$ run app.package.attacksurface com.mwr.example.sieve
```

We can see that 3 activities are exported

Figure 4.2: Attack Surface

Second, get information of each activity by using the command

```
$ run app.activity.info -a com.mwr.example.sieve
```

Figure 4.3: Activity Information

we can see that all three activities are exposed what information is required to open each activity. One activity named as com.mwr.example.sieve.PWLIST seems vulnerable. Now we can open this activity and see what information we can get.

At the end, We start this activity by using the command:

```
$ run app.activity.start --component com.mwr.example.sieve
com.mwr.example.sieve.PWLIST
```

Activity reveals all the accounts information saved in this app. Below is the screenshot

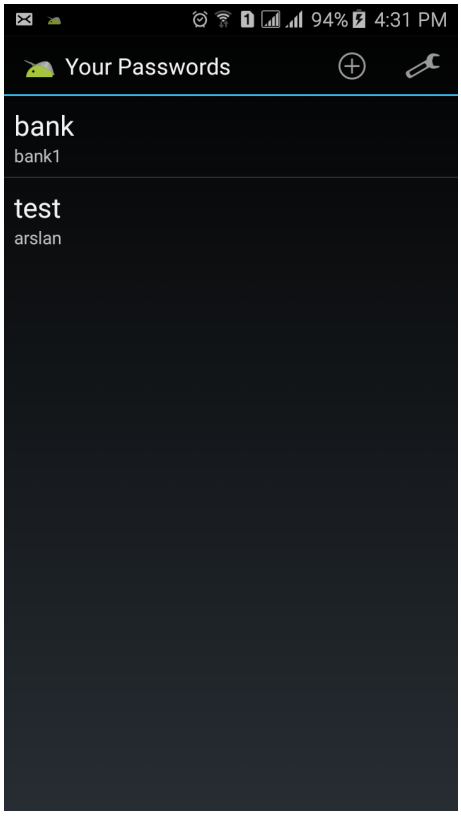


Figure 4.4: All Accounts saved in SIEVE

4.5.2 Exploiting Content Providers

We follow exactly the same steps as we follow in exploiting activities.

Steps

1. Check Attack surface
2. Get exported content provider info
3. Find Uri of each directory
4. Query that Uri to open database

4.5.2.1 Demonstration

While checking attack surface in exploiting activity, we get to know that 2 content providers are exported. Now we need to know that what permissions those providers require to access databases.

First get provider information by using the command:

```
$ run app.provider.info -a com.mwr.example.sieve
```

```
OpenSource:~ arslaan$ drozer console connect -c "run app.provider.info -a com.mwr.example.sieve"
Selecting a7d89703f7fc1ca0 (samsung SM-J320F 5.1.1)

Package: com.mwr.example.sieve
  Authority: com.mwr.example.sieve.DBContentProvider
    Read Permission: null
    Write Permission: null
    Content Provider: com.mwr.example.sieve.DBContentProvider
    Multiprocess Allowed: True
    Grant Uri Permissions: False
    Path Permissions:
      Path: /Keys
        Type: PATTERN_LITERAL
        Read Permission: com.mwr.example.sieve.READ_KEYS
        Write Permission: com.mwr.example.sieve.WRITE_KEYS
  Authority: com.mwr.example.sieve.FileBackupProvider
    Read Permission: null
    Write Permission: null
    Content Provider: com.mwr.example.sieve.FileBackupProvider
    Multiprocess Allowed: True
    Grant Uri Permissions: False
```

Figure 4.5: All Exported Providers

As result shows, two content providers do not require any permission to access.

To find the exact URI of provider to be queried, we need to find URI of each dir. We use the command below to check the URI.

```
$ run app.provider.finduri com.mwr.example.sieve
```

This commad reveals all the uri available

Here we can see that the URI ended with **passwords** is of our interest.

```
OpenSource:~ arslaan$ drozer console connect -c "run app.provider.finduri com.mwr.example.sieve"
Selecting a7d89703f7fc1ca0 (samsung SM-J320F 5.1.1)

Scanning com.mwr.example.sieve...
content://com.mwr.example.sieve.DBContentProvider/
content://com.mwr.example.sieve.FileBackupProvider/
content://com.mwr.example.sieve.DBContentProvider
content://com.mwr.example.sieve.DBContentProvider/Passwords/
content://com.mwr.example.sieve.DBContentProvider/Keys/
content://com.mwr.example.sieve.FileBackupProvider
content://com.mwr.example.sieve.DBContentProvider/Passwords
content://com.mwr.example.sieve.DBContentProvider/Keys
OpenSource:~ arslaan$
```

Figure 4.6: List of all URIs available

Now we try to query this URI and see what we get.

```
OpenSource:~ arslaan$ drozer console connect -c "run app.provider.query content://com.mwr.example.sieve.
DBContentProvider/Passwords"
Selecting a7d89703f7fc1ca0 (samsung SM-J320F 5.1.1)

| _id | service | username | password                                                | email             |
| 1   | test    | arslan   | iz9w3EmbiHLciSCJbofvgrt+I3h8bvQI7w== (Base64-encoded) | arslan@arslan.com |
| 2   | bank    | bank1    | r+QsnX6eHoHyEP8KPBj7uaIPFQloY48= (Base64-encoded)      | arslan@bank1.com  |

OpenSource:~ arslaan$
```

Figure 4.7: Database of all Accounts saved

This URI is accessible and reveals all the account with username and password hashes. Now anyone can steal passwords or replace these hashes with new one OR can register a new account with same password hash. Some other attacks like SQLi are also possible.

4.6 Results

This section include all the results found while analysing Sieve app. Following is the number of exported and non-exported components.

Table 4.1: Number of Activities Exported and Non-Exported.

Activity Type	Exported	Non Exported
Activity	3	4
ContentProviders	2	3
Services	2	0
BroadcastReceivers	0	0

As shown in table above that 3 activities, 2 CP and 2 services are exported. so after analysing all these we came to know that 1 activity and 1 CP is prone to attack.

Table 4.2: Number of Vulnerable Activities

Activity Type	Number of Vulnerable Activities
Activity	1
ContentProviders	1
Services	0
BroadcastReceivers	0

Chapter 5

IADF Analyzer

IADF Analyzer is Inter Application Data Flow Analyzer which can detect intent based data flow from one app to other. It has two major modules. First module extract data from manifest file which is required by any app to access an activity, this module is named as Manifest data extractor. Second module is used to extract extra data information from intents. It also tells which apps are communicating with each other and what data is carried by an intent. Intent is asynchronous message passing technique officially supported by Android.

Extracted data is saved in CSV file which is further used to analyse sensitivity of data and categorisation of data paths. Based on these paths we categorise activities.

5.1 Manifest Data Extractor

Manifest Data Extractor extract xml data from AndroidManifest.xml by parsing this file to python module. To parse xml file in python we use lxml library which can efficiently

extracts xml data and save it to whatever format we require. For the sake of thesis, we save parsed data in CSV file which can be easily read and analysed in MS Excel. Visualisation of extracted data in Excel is also very handy because of built in support by MS.

Extracted data contains below information:

- Sending Activity Name
- Receiving Activity Name
- Exported/Non-Exported
- Permissions
- Action
- Category
- Data Type and Scheme

All this information is required by sending app to match it's intent data with this intent-filter data if it wants to communicate.

5.2 Intent Data Extractor

Intent Data Extract extract intent's information which is being sent through one app to other. Intent's information of sending app should match to the intent-filter's information mentioned by other app in manifest file. To extract this information, we first reverse all

classes of APK package. After reversing, intent's information is extracted by searching strings of predefined functions like setType, setAction, setCategory etc.

All below information is extracted and saved in CSV file.

- Activity name to be received
- Action
- Category
- Flags
- Extra Data

5.3 Results

All apps are real world applications downloaded from Google Play store. Total 2716 activities from 28 apps analysed. Out of 2716, only 179 activities are exported which can be accessible from outside world. After analysing all these exported components we came to know that only one activity is prone to attack.

Table 5.1: Number of Activities Exported and Non-Exported.

Activity Type	Exported	Non Exported
Activity	179	1404
Content Providers	0	654
Services	97	214
Broadcast Receivers	60	104

5.3.1 Explanation of Vulnerable Component GroupME

GroupMe is a social media app which allow users to make and join groups of interest, find new friends and chat with each other.

Activity name "com.groupme.androidcom.groupme.android.settings.SettingsActivity" is found exported. This activity contains all settings information regarding user. To the best of our knowledge settings activity should not be exported so that no other app or user can access it in any way. Exposing settings to other users is not a good practice.

Table 5.2: Number of Vulnerable Activities

Activity Type	No. of Vulnerable Activities
Activity	1
Content Providers	0
Services	0
Broadcast Receivers	0

5.3.1.1 Analyzing

After reverse engineering and analyzing that which application is prone to attack we use DROZER to send custom intents. As already explain that Drozer is open source android testing framework which allows us to send custom intents to applications. we use this to send intents to application activity which is prone to attack. After opening that activity, we came to know that it reveals all settings information related to particular user.

Following is the procedure used to verify that GroupMe reveals settings. All the steps are same as we follow in analyzing Sieve in chapter 4.

Steps

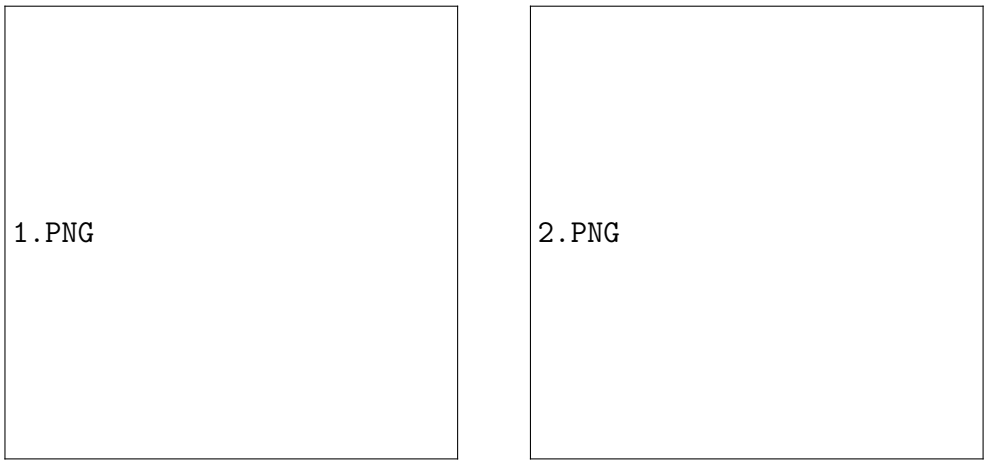
1. First we check which activities are exported. SettingsActivity is seems vulnerable.

```
OpenSource:~ arslaan$ drozer console connect -c "run app.activity.info -a com.groupme.android"
Selecting a7d89703f7fc1ca0 (samsung SM-J320F 5.1.1)

Package: com.groupme.android
  com.groupme.android.HomeActivity
    Permission: null
  com.groupme.android.chat.ChatActivity
    Parent Activity: com.groupme.android.HomeActivity
    Permission: null
  com.groupme.android.group.JoinGroupActivity
    Permission: null
  com.groupme.android.contacts.AddContactActivity
    Permission: null
  com.groupme.android.message.MessageDetailActivity
    Parent Activity: com.groupme.android.chat.ChatActivity
    Permission: null
  com.groupme.android.profile.ProfileActivity
    Permission: null
  com.groupme.android.settings.SettingsActivity
    Permission: null
  com.groupme.android.support.SupportActivity
    Permission: null
  com.groupme.android.sharing.SharingActivity
    Permission: null
  com.groupme.android.powerup.store.PowerUpStoreActivity
    Permission: null
  com.groupme.android.contacts.AddToGroupActivity
    Permission: null
```

Figure 5.1: Exported Activities and Attack Surface. Setting activity is exported

2. Now we try open SettingsActivtiy and see we is behind this activity.



(a) label 1 **(b)** label 2

Figure 5.2: Setting activity. Attacker can change any of this setting

Now we can change or modify any of the settings to irritate user.

By following all these steps we can analyze any application to check application collusion attacks. Application collusion can take place when two or more applications are communicating with each other legally and sharing such data which other application has not permission to access it. Android OS is not checking that which kind of data is sending and either they have permissions or not. Our mechanism can help developers and security analysts to check if their application is prone to such attack or not.

Chapter 6

Conclusion

Android application collusion attacks can compromise user privacy and can reveal system information. We develop a tool which can detect intent based data flows and developer weaknesses in terms of defining components in android manifest files. Our tool is robust and comprehensive. We can analyse as many apps as we want in one go. First it reverses the app and extracts data required for inter app communication analysis. After extracting all information it automatically save the data in CSV file which can be further used for visualisation. It also tells the analyst that what data is sharing between applications. After knowing all this information we have to manually analyse data sensitivity and data type. At the end we categorise all the vulnerabilities based on all possible attacks which may be launched through Application collusion.

Future Work

In future we will enhance this tool by adding some more features. Data types, categorisation and Visualisation is not included as yet in this work. Before launching this tool we are planning to add all these features in this tool.

List of Abbreviations and Symbols

Abbreviations

APK	Android Package
CSV	Comma Separated Version
URI	Uniform Resource Identifier
CP	Content Provider
IADF	Inter App Data Flow
MS	MicroSoft
OS	Operating System
AOSP	Android Operating System
TV	Televission
SDK	Software Development Kit
ID	Identity

VM	Virtual Machine
API	Application Programming Interface
IPC	Inter Process Communication
ICC	Inter Component Communication

References

[1] Android Application Statistics, Statista, Available here, https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/

[2] Application Fundamentals, Android, Available here, https://www.developers.android.com/

[3] Interacting with Other Apps, Android, Available here, https://developer.android.com/training/basics/intents/

[4] Lee, Youn Kyu, et al. "A SEALANT for inter-app security holes in android." Proceedings of the 39th International Conference on Software Engineering. IEEE Press, 2017.

[5] Klieber, William, et al. "Android taint flow analysis for app sets." Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis. ACM, 2014.

[6] Li, Li, et al. "Apkcombiner: Combining multiple android apps to support inter-

app analysis." IFIP International Information Security Conference. Springer, Cham, 2015.

[7] Felt, Adrienne Porter, et al. "Android permissions demystified." Proceedings of the 18th ACM conference on Computer and communications security. ACM, 2011.

[8] Chin, Erika, et al. "Analyzing inter-application communication in Android." Proceedings of the 9th international conference on Mobile systems, applications, and services. ACM, 2011.

[9] Arzt, Steven, et al. "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps." Acm Sigplan Notices 49.6 (2014): 259-269.

[10] Fritz, Christian, et al. "Highly precise taint analysis for android applications." (2013).

[11] Holavanalli, Shashank, et al. "Flow permissions for android." Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering. IEEE Press, 2013.

[12] Fora, Pau Oliva. "Beginners guide to reverse engineering android apps." RSA Conference. 2014.

[13] Winsniewski, R. "Android?apktool: A tool for reverse engineering android apk files." (2012).

[14] Strobel, Mike. "Procyon/java decompiler." (2016).

[15] MWR Security, Drozer, Available here https://labs.mwrinfosecurity.com/tools/drozer/

[16] Android Debug Bridge, Available here https://developer.android.com/studio/command-line/adb

[17] Java, Available here https://java.com/en/download/