Exploitation and Defense against Android Collusion

By

Sadaf Rasheed

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

Aug 2022

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS Thesis written by **Ms. Sadaf Rasheed**, Registration No. **00000278498**, of **Military College of Signals** has been vetted by undersigned, found complete in all respects as per NUST Statutes/Regulations/MS Policy, 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 and local evaluators of the scholar have also been incorporated in the said thesis.

Signature: _____

Name of Supervisor: **Assoc Prof Dr. Mian M Waseem Iqbal**

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

“In the name of Allah, the most Beneficent, the most Merciful"

I dedicate this thesis to my mother, sister, and teachers who supported me each step of the way.

ACKNOWLEDGEMENTS

All praises to Allah for the strengths and His blessing in completing this thesis.

I would like to convey my gratitude to my supervisor, Associate Professor Dr. Mian Muhammad Waseem Iqbal for his supervision and constant support. His invaluable help of constructive comments and suggestions throughout the experimental and thesis works are major contributions to the success of this research.

Last, but not the least, I am highly thankful to my parents. They have always stood by my dreams and aspirations and have been a great source of inspiration for me. I would like to thank them for all their care, love and support through my times of stress and excitement.

ABSTRACT

Each app in a mobile requires a certain type of user's data in order to function for example WhatsApp requires contacts, Facebook requires access to Gallery/Camera. Applications ask for permissions to its users before accessing their personal data saved in mobile and users can allow or deny these permissions, however, denying these permissions will limit app's functionality. Since almost every app is accessing some level of user's private data saved on mobile, it is difficult to classify which app is benign and which one is malicious. Android Collusion is a new type of attack where two or more apps collude to access user's data illicitly or perform malicious activity. It is not necessary that every app involved in this attack is malicious. The aim of this research is to determine new ways of application collusion between two apps in android and find possible ways to avert from such attacks occurring.

Contents

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

OS	Operating System
PII	Personally Identifiable Information
DB	Database
SQL	Structured Query Language
API	Application Programming Interface
APK	Android Package Kit
URL	Uniform Resource Locators
SDK	Software Development Kit

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Initially mobile phones were made with the aim of communicating without any wired telephonic line. However, gradually the development in science and technology led to many new discoveries in day-to-day life and evolution in mobile phones as well. This evolution facilitated its users to a next level. Other than attending phone calls, mobile phones provided conveniences like sending SMS, MMS, listening to radio etc. But in the early 2000s, advancements in mobile phones further emerged and revolutionized the whole concept of mobile phones due to which they were renamed as smartphones. Today smartphones are so much more than just sending SMS or attending phones calls. They can assist with web browsing, Bluetooth, emails, GPS, weather and news updates, ecommerce, online banking and so much more. Smartphones have undoubtedly provided ease at life by facilitating all these features and there is literally no such huge difference left between a smartphone and a computer. A smartphone can pretty much do all the jobs that requires computer/PC to some extent. However, everything comes with a price, and in this case, it is users' private data such as web history, credentials, location, contacts, personally identifiable information (PII). While all the features of smartphones were being introduced to assist its users' in performing day to day activities with ease, different types of attacks were also being discovered alongside that puts users' data at high risk. Therefore, it would not be wrong to say that every smartphone feature that assists its users at some tasks, compromises some level of user's private data. For example, if a user logs into his banking app to perform some transactions, the user is given a choice to store these credentials on his device for future use. Once stored, these credentials become vulnerable to different types of attacks in smartphones. Just like computers, smartphones are also vulnerable to attacks especially if it is comprised of user's private data. Some of the known attacks in smartphones are, data theft, spyware, phishing attack, network spoofing etc.

All these attacks are studied, and much research have been done on it already and though these attacks if performed, can bring harm to user's assets, data etc. attackers are now onto something even bigger. A unique android-based attack is in the market known as application collusion. Application collusion is always comprised of at least two applications. In this, the attacker makes two or more applications, group together and communicate i.e., share data, in order to perform malicious activities.

This research aims to learn new techniques and methods of performing application collusion so that new areas of threats and vulnerabilities in Android can be discovered and provide a solution on preventing from such attacks to enhance the security.

1.2 PROBLEM STATEMENT

Application collusion is a type of attack where attacker makes an application that can illegally access user's personal data without user's permission. It is easier to gain deep level access into a user's device by designing an application and masquerading it to be as legitimate and then making it communicate with other applications installed to share some sensitive data. In this way the attacker without any complicated coding can access users' private data and perform harmful activities. The user, however in this case has no knowledge that his private data is being accessed. The main motive of every intruder after getting access into any network or device is to remain hidden and not get caught, this is because he wants as much information and data of user as he can get and therefore, for this purpose he makes sure never to perform any action which could lead to make the user aware of any irregular activity. Hence, once the private data is accessed illegally, the attacker can easily use this data for any malicious purpose such as stealing credentials, send out data, monitoring user's location, eavesdropping etc.

Previously several research have been conducted on how application collusion could possibly be detected and prevented but they all come with their own drawbacks. Moreover, there are various reasons why it is difficult to detect or prevent this attack. Some of the reasons could be as follows:

- Presence of covert communication channels.
- There are still new areas or techniques of this attack to be discovered therefore there can't be a strong solution on how to detect or prevent from it entirely.
- Unavailability of present colluding applications is the most important factor why this type of attack has still not been addressed properly.
- There is a large number of applications being developed every single day, grouping each one of them in order to detect this attack is a huge computational work and impossible.
- Since the user under this attack is unaware that his data is being accessed illicitly, it is difficult to apply a preventive solution to it unless detected properly.

- The Android OS allows itself for applications to communicate to share data with some built in features.

Considering all the above-mentioned reasons, this study will work on a new version of application collusion attack. It will be shown how an application with low security can be exploited to share data with a malicious application that appear to be benign. Furthermore, a framework to detect application collusion in this scenario is also proposed which works on a probability matrix.

1.3 RESEARCH OBJECTIVE

The main objectives of this study are:

- To study and analyze existing Android collusion attack parameters.
- Propose collusion detection and prevention mechanism.

1.4 SCOPE OF RESEARCH

Application Collusion is relatively a new type of attack and there is yet a vast amount of research to be conducted to discover its new approaches. The available prevention and detection tools for application collusion do not entirely prevent or detect this attack and the results are not completely accurate. Since there are multiple factors, this attack is based upon, none of the previous research has yet covered all the potential factors of it.

This study will be conducted in two divisions, offensive and defensive. In offensive section, the aim is to discover a new technique to perform collusion attack and bring into light the present vulnerabilities in Android. In defensive section, the goal is to perform deep analysis of present tools and propose preventative solution in form of a probability matrix that could be applied so it can help identify potential colluding applications in a device.

1.5 SIGNIFICANCE OF RESEARCH

An application in a smartphone requires some user's private data to function, for example Google maps require an access to user's location, WhatsApp requires an access to user's contacts, Instagram requires an access to user's Camera and Gallery. Each application developed in Android has some access to user's data in order to function. Applications ask for such access requests in form of permissions. It is up to the user to allow or deny these permissions. If the permission is granted, the applications have control to read and write over user's private data and therefore, this private data is always vulnerable to attacks like data theft, identity theft, spyware etc. However, Android assures that every application developed is authentic and their data is protected within the device, but since Android OS is an open

source, and too common around the world, it is the foremost target for attackers to intrude into users' mobile phones and misuse their personal data after getting access to it. Application collusion is known to be an attack where user is not aware that his data is being read or modified by other applications through intruders. Moreover, this type of attack exists till date, and there is still a vast range to be discovered in this field since it is not addressed entirely due to some setbacks. This research is going to help understand new potential approaches leading on to application collusion attacks and how one can avoid from being affected by such threats.

CHAPTER 2

LITERATURE REVIEW

2.1 INTRODUCTION:

Smartphones are known to be one of the biggest advancements in technology that has brought massive impact in everyone's lives. People use smartphones for work, education, social life, and personal use. Just like computers, smartphones also have operating systems. By now the two most prominent operating systems are IOS and Android. IOS is supported by Apple and is only designed for Apple devices such as iPhone, MacBook, iPad, etc. Android, on the other hand, is supported by Google. It is an open source and more common among people which is why it is more exposed to threats and attacks. It has been in the market for about 15 years now. According to studies in August 2021, Android has over 3 billion active users which makes 39% of the entire population. [1] Due to its openness, a large number of developers use it as a community driven projects and therefore its new features and updates are better in quality as well as faster than any other official manufacturer channels. [2]. Android OS is comprised of multiple applications. Applications can be described as a software that runs on a smartphone. They provide users with services. These services are usually small, particular software components with some measure of function.

2.2 ANDROID APPLICATIONS AND PERMISSIONS:

Android applications need user's data to function. Applications request to access user's data in form of permissions. The user either denies or allows them. If an app is denied a permission, it cannot access user's data. If the permission is granted, the app can read and even write over user's data in some cases. [3] However, there is one major drawback of allowing apps to read data i.e., the user's data remains exposed to various types of android attacks. Therefore, the major benefit of using permissions is to take user's consent, which means the user is aware of the possible threats and risks and still willingly allows the apps to view his data in order to use the app's functionality.

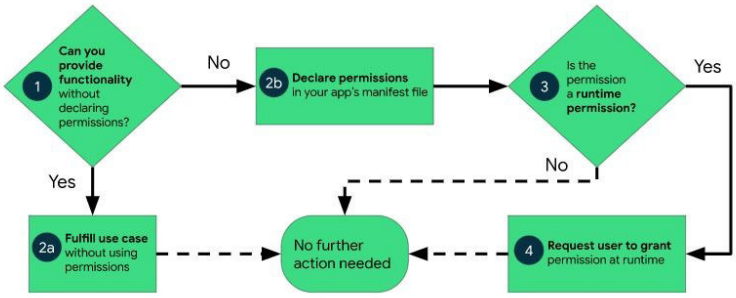

Figure 1 - permission flow

According to the release of Android OS (Android Marshmallow 6.0 - API 23) in 2017, all required permissions in an app must be declared in its manifest file with the tag of "user permissions", which means that no app provider can access user's data without displaying user with custom dialog prompt about permissions. [4]

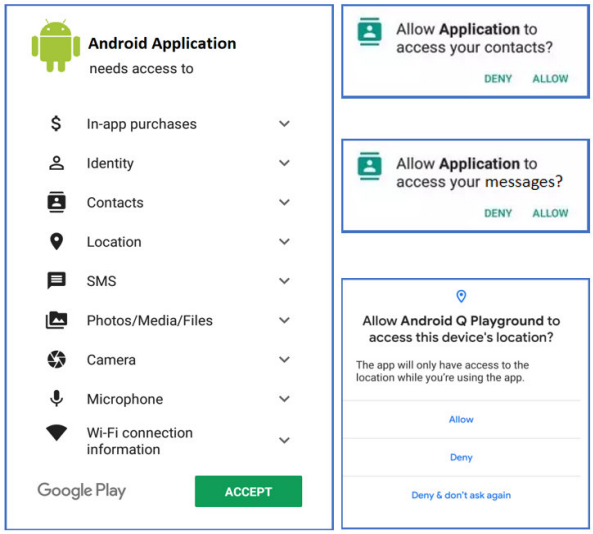

Figure 2 - Android permission dialogue

There are four different categories of permissions in android. Each category of permission specifies the range of restricted action the app can perform or the range of restricted data of user that an app can access after the system has granted the permissions. [3]

2.2.1 Install Time Permissions

These types of permissions have very low to almost no access to sensitive and restricted data. Once an application with such permissions is installed, the system grants the access to application automatically because these can only perform actions that affect system or other apps to the minimal. In an App Store, install time permissions can be viewed on an app's

detail page. Examples of such permissions could be access network state, Bluetooth etc. It has further two subcategories that are explained below.

2.2.1.1 Normal Permissions

Such permissions do not require runtime prompts. System examines the manifest file of the app and goes through the list of all permissions and allows these at runtime automatically. With such permissions, an app can perform actions such as location access, creating app's shortcuts, killing background processes etc.

2.2.1.2 Signature Permissions

Every application that is developed has a signed certificate by its developer. If there are two applications with the same developer, both applications would have the same signed certificate, which means that both of them will have same level of permissions.

2.2.2 Dangerous Permissions

These types of permissions have capability to affect user's private data or device's operations. Therefore, a runtime prompt is mandatory before accessing these permissions. Android apps are not allowed to use these permissions until the user himself agrees and explicitly grant these permissions. Some of the examples could be read/write contacts, messages, make or answer phone calls, accessing location, gallery, external storage, call logs etc.

2.2.3 Special Permissions

These permissions are related to certain application operations. They can be defined by only Original Equipment Manufacturer. Moreover, these are defined only when it is intended to protect access to mainly powerful actions, such as drawing over other apps. The Special app access page in system settings contains a set of user-toggleable operations. Many of these operations are implemented as special permissions.

Presently, the procedure for applications to access sensitive and restricted data or perform actions on it is that, first the developer of the application must define permissions in application's manifest file, once the application is installed, the system goes through manifest file looking for install time permissions and allows those permissions automatically as discussed above. In case there are further runtime(dangerous) or special permissions present, A custom prompt message appears asking the user for accessing restricted data or to perform certain actions. If the permissions are legitimate, then the user allows those permissions else they are simply denied. Considering this, the scenario depicts that no

application can access user's data without his consent and therefore no malware can enter in a device, nor any attack should take place. But since android attacks and android malwares are so sophisticated today, it is not always easy to detect or prevent from them. Malicious applications are disguised as benign ones and ask for permissions to access user data and perform restricted actions. After the permissions are granted, they perform illicit actions at the backend such as spying, monitoring user actions, and sending all the recorded data to attacker. These applications can further install malwares at the backend. And as the user is unaware of this whole situation, such malicious applications and malwares remain undetected meanwhile all of the user data is being sent to attacker which can lead to even bigger attacks. Some malicious applications and malwares are so refined that they can easily evade smartphone's security and authentication process. Once a malware evades the security and authentication barrier, they can convincingly perform any malicious attack without letting the user have any idea of it.

2.3 ANDOIRD ATTACKS:

Android is most aimed at platform for attackers due to its open-source nature. The purpose behind every attack is ultimately to compromise or steal data from mobile devices for various reasons such as sending out the data, signing up users for services without user's knowledge, locking device to demand ransom etc. [5] There are numerous kinds of attacks and malwares in Android that puts user's data at danger in different ways. [6]

2.3.1 SMSHING

This attack involves a malicious website link, designed by attackers to gain unauthorized access to victim's smartphones to make calls, send texts and even send sensitive information to malicious websites without user's knowledge. This link is distributed among target phones. The user is unaware that his phone is under attack

2.3.2 PHISHING

Phishing is similar to smshing attack, the only difference is that the malicious website link is propagated through emails and once the user clicks on this link, all his sensitive information is sent to attacker.

2.3.3 ROOTING

Rooting is done to unlock the OS in order to install unapproved apps, replace firmware, update OS etc. However, this process makes smartphones vulnerable to various kinds of malware and give access to attackers.

2.3.4 UNTRUSTED APKS

Users are convinced to install applications from third parties and untrusted sources. Once the untrusted APK is installed, it can perform several malicious activities such as spying and sending reports to attackers, sending out data, perform illegal actions etc. The installed APKs could also possibly contain malicious software, giving remote access to the attackers.

2.4 ANDROID MALWARES

Malwares are malicious software designed by hackers to intrude into smartphones in order to steal information, monitoring activities spying, gain control over user's smartphone etc. With the passage of time as smartphones evolved, malwares became more sophisticated and their methods to evade smartphones' security and authentication processes have improved as well. Discussed below are different ways and types of malwares that are most common and threatening to smartphone's security. [7]

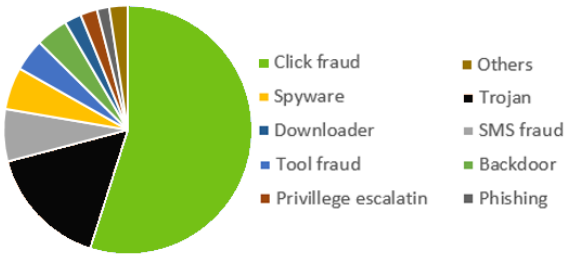

Figure 3 - Android Malware Classificatio

2.4.1 BACKDOOR

A way to breach smartphone's security and authentication processes and then covertly execute unwanted, malicious codes and perform harmful activities. It allows attackers to gain access to user's data. It can also be used to establish further communication channels for other malwares and attackers to intrude in smartphone.

2.4.2 CLICK FRAUD

Hackers are paid when a user clicks on adds. They generate fake clicks by overlaying buttons, images, and test layouts over advertisements.

2.4.3 RANSOMWARE

A type of malware with which an attacker prevents a device or a block of data from user access. Attacker then demands a ransom from user or asks to perform certain actions in order to unblock the device or data from access.

2.4.4 SPYWARE

It is a type of malware that allows attackers to monitor and record user's sensitive information without his/her permission. It can also covertly transmit all the recorded data from user's device's hardware.

2.4.5 ADWARE

Adware is also a malware that hides on a device and displays advertisements. It operates in the background and generates a popup window to display fake ads and sell fake products. Adware also monitors user action

2.4.6 TROJAN

An application that is designed by attackers and is disguised as legitimate but once downloaded, it starts to function maliciously and executes malicious code hidden from user. Attackers can gain backdoor access, steal, or spy sensitive information from user's device with the help of such malwares

2.4.7 DOWNLOADERS

In Android, a downloader can be defined as a malicious application that, when gets an access to internet, downloads further malicious malwares without user's knowledge.

Just like how smartphones are getting faster, better and improved every day, android malwares are also getting repackaged and more advanced with an intend to infect Android devices on a bigger level. Initially any attack or malware created was eventually detected due to suspicious behavior of device such as slowed processes, unidentified files or software downloaded etc. But since more refined and sophisticated malwares and attacks have been taking place, it becomes difficult to detect and identify them because most of them are performing their actions at the backend. Different research has been conducted on tools and solutions for malware detection have been submitted, each study focused on a certain behavior of an application based on which they were identified as either malicious or benign.

In 2019, research presented a method to detect malware applications in a device by collecting mobile traffic generated by the applications when they get connected to the internet. The traffic was then evaluated, and each URL visited by applications were examined. URLs were divided into several segments and analyzed by applying algorithms. [8] Though, this method is not effective because it only focuses on the URLs. Plus, there is a chance of false positive result as well. Another limitation of this method is that it only focuses on HTTP URLs traffic which means this method is not applicable on any non-HTTP protocols or even HTTP encryptions. In the same year, another research was conducted, and a new method was designed to determine malicious applications. This method focused on a

concept of every application in order to use system's services, depend on Android APIs. This is because each Android API supports a unique system service such as I/O management, graphic processing, memory management etc. Consequently, it means that an application's main objective and characteristics can be determined by looking at the list of APIs it has. In this research, two classified ranked lists were constructed benign_api_list and malicious_api_list. Benign_api_list contained all the commonly used APIs among benign applications. Malicious_api_list contained APIs that were usually found in malicious applications. Then for any suspicious application, its APIs were analyzed. For all the benign APIs present in that suspicious application, sum of the inverse values of the ranked APIs were calculated and similarly sum of the inverse values of the ranked APIs for malicious APIs present in that application were calculated as well. As a result, if the sum of inverse values of benign APIs were greater than that of malicious APIs, then the app was determined as benign else malicious. [9] However, this method also fails to serve the purpose since 1000s of applications are developed, and new malicious APIs are constructed, it is impossible to keep a track of all of them and updating the API list every now and then. Plus, it is not a strong solution to completely rely on APIs for malicious app detection since there is a possibility that malicious apps could use benign APIs also. During the same year another research focused on android permissions to detect malicious apps by acquiring datasets of malicious and normal applications (from 2010- 2014 and 2014-2018) and extracting permission pairs from their respective manifest files. A graph of all the permission pairs is structed and an edge weight is assigned to permission pair based on the number of malicious application it is present in. [10] This approach has limitations and some grey areas in it. Firstly, this approach requires all the permission pairs for the detection of malicious application. Secondly, there aremany malicious applications containing very few normal permissions and they can evade this detection approach. Moreover, this approach cannot work on any app with no permissions. Thirdly, many social media applications require dangerous permissions such as access gallery, access contacts etc. This approach identifies normal social media applications as malicious applications based on the permissions in manifest files. Hence, this method also has false positive rate. A survey discussed how attackers are always ahead of anti-malware groups and how important it is to keep a track of all the malicious apps, their working, and what tools have been presented for their detection. It further explains all the timely various static and dynamic approaches that have been proposed to counter the advance malicious applications. This study helps to open new directions for future research. [11] Later in the year, another systematic survey was done on 236 papers (from 2011-2019) about android malicious application detection. It discussed all the limitations of the detection tools and solutions. With that, the paper

also provided recommendations on how those solutions could improve. Moreover, it also stated why detection of malicious applications are difficult and what are the possible solutions to overcome the challenges faced meanwhile. [12]

In the early 2020, A host-based intrusion detection system (HIDS) was introduced, this model was incorporated with statistical and semi supervised machine learning algorithms. It required only benign applications' behavior as features with a few malicious ones for tuning. Rather than supervising each application individually for suspicious behavior, this model worked on dynamic analysis and constantly looked for suspicious activity at the device level based on incorporated set of features. Afterwards, to define an application's run time behavior, the model applies a machine learning or a statistical algorithm and classifies it either benign or malicious. [13] This model and its working approach seemed comparatively better than most of the solutions provided prior, yet it had some restrictions as well. Firstly, in order for this model to work effectively, it requires richer data sets for feature learning of both malicious and benign applications working. But since every day new means to evade malicious application detection are being made by the intruders, it becomes a challenge to create a dataset of all the malicious features. Secondly this model could be more useful if it had a prevention mechanism integrated in it as well because it only detects the malicious applications, a mitigation mechanism could make this model more efficient. Later, in the mid of the same year, MADFU (Malicious Application Detection on Features Uncertainty) was introduced. MADFU works on the basis of logistic regression function to describe the relationship between permissions and labels. The analysis found out that there are some uncertainties in the features of android that effects the detection of malicious application. As a solution MADFU was presented that solved the uncertainties of dangerous permissions. [14] However, further studies revealed several limitations of MADFU, because it only classifies malicious applications on the basis of dangerous permissions, but that may not be the case always because there are various root-exploit level malicious applications that do not have any permissions to use during the analysis. Therefore, it is impossible to carry out malicious application detection based on dangerous permissions only. Another paper that is important to include in this research is basically a comparative study about how malicious app works differently from a benign application and what are their similarities. It studies both static and dynamic analysis of each set of applications and brings into light important factors to consider on detection of malicious applications based on their run time behavior. [15]

It can be observed that in the past years, every research on malicious application detection has focused on a particular feature of android application. A table below gives a concise description of what each paper focused on and what it contributed.

Analysis	Year	Title	Description
Dynamic	2019	Deep and broad URL feature mining for android malware detection	Focuses on traffic generated by apps and analyzes URLs visited by applying algorithms in order to determine whether app is malicious or not
Dynamic	2019	Detecting malicious android apps using the popularity and relations of APIs	Constructs two APIs list i.e., malicious, and benign and compares the list for any suspicious app by calculating its sum of inverse values. If the sum value of benign API is greater than malicious then the app is benign else malicious
Static	2019	Group wise classification approach to improve android malicious application detection accuracy	Uses drebin benchmark malware dataset in order to explain how malicious app detection can be enhanced by analyzing the apps after grouping the collected data based on the permissions
Static	2019	Permpair: Android malware detection using permission pairs.	Extracts permission pairs from dataset of malicious apps and compares the permissions for any suspicious application
Survey	2019	A survey on the detection of android malicious apps	Discusses potential features any malicious application could contain and how they could be detected
Survey	2019	Constructing features for detecting android malicious applications: issues, taxonomy, and directions	A systematic review off 236 papers from 2011 to 2018 of all the tools presented for malicious app detection and provides recommendations on their limitations.
Dynamic	2020	An autonomous host-based intrusion detection system for android mobile devices. Mobile Networks and Applications	Presents a model HIDS that is trained by a set of features and detects malicious apps on a device level dynamically based on app's activities.
Static	2020	MADFU: An Improved	Solves the uncertainties while detection of

		Malicious Application Detection Method Based on Features Uncertainty	malicious applications and classifies any app to be malicious or benign based on dangerous permissions accessed by them.
Literature review	2020	A study of run-time behavioral evolution of benign versus malicious apps in android	A comparative study between malicious apps and benign apps and what their similarities and differences are in working. This study helps to understand more about malicious apps and how a better solution for detection can be provided.

Table 1 - Static and Dynamic analysis

In a paper [12], it discussed various challenges that are faced during detection of malicious applications, some major challenges are mentioned bellow:

- Threat of application collusion is neglected
- Applications that contain malware may have their code obfuscated through several techniques and complicated program comprehension.
- Malicious applications could contain encrypted code which could have gone unnoticed during detection mechanism.
- Datasets included in research studies do not include datasets of malwares like clones, adware, data miners etc.
- During static analysis of malicious applications, dynamic loading and reflection call are still a challenge that requires more work.
- Metamorphic malware modifies its code itself by rewriting for example renaming methods or classes in an application. Detecting such apps that contain such malwares is a major challenge.
- The process of extracting features can be time consuming due to the increase in size and highly complicated behaviors of Android Package which results in a non-effective detection.
- Extraction of well discriminated static features is also a challenge because the behaviors of android apps have become progressively more polymorphic and sophisticated.
- During malicious application detection, dynamic analysis cannot track all the possible paths of execution which could result into false negatives results.
- Extracting app's features could go up to a million, major issue is how to process the sparse vectors.

- Dynamic features extraction is difficult if an app is protected at runtime security mechanisms

Any malicious application could potentially contain a malware or a piece of code that performs harmful activities on a user's device. Out of all the existing challenges confronted in detection of malicious application, this study focuses on the first point mentioned, application collusion. It is an attack where two or more applications group together to share data and perform malicious activities. Though not much of the research has been done on it because it is comparatively a new attack and there are still many considerations that need to be discovered in order to address this attack properly. The complication of this attack is that these malicious applications are developed in such a way that they seem like a legitimate application. Therefore, no detection method could detect them either. It could have all the features that a benign application contains and at the same time it could take advantage of android vulnerabilities to access user's sensitive data and perform malicious activities at the backend without user having any idea of it. In order to detect application collusion, a model named FUSE was proposed in 2014, this model first analyzes every single app and stores relatable information and then combine all information to detect collusion based on some restricted policy engine. The issue with this model is that this policy is not publicly available. [16] Another research performed a number of experiments on all the models proposed as solution for detecting application collusion and indicated that none of them could work for the detection of application collusion, because all the solutions offered prior had some level of limitation or false positive results. This research then further presented a map of all the possible communication channels among Android apps in order to statically characterize inter-app ICC. The presented map even though does not provide any solution to application collusion detection, but it can identify which applications could possibly be communicating. Hence the motive to present this static ICC map was for future researchers to identify communication channels and apply potential security policies for prevention from application collusion. [17] A number of researchers tried to develop a novel analysis method to detect ICC. Their research proposed a model that statically analyzes each application and retrieves applications' communication at component level. This communication retrieved is then further represented in the form of a state machine in order to detect collusion. [18] However, this method only works to detect collusion between two applications and that too only at component level.

Even after above mentioned research, application collusion still remained a challenge for Android security because attackers always found a way to evade authentication procedures by

masquerading as a benign application and then accessing user's restricted data by sharing resources of other applications. A paper in 2017, analyzed over 10,000 Android applications and discussed most important factors in occurrence of application collusion. First It presents a tool that analysis applications statically on a large level and is named as called AppHolmes. This tool first extracts two things (i) manifest file, for learning application's component, intent files and permissions, (ii) smali code, to carry out static analysis. After gathering all the required information and performing analysis, the too AppHolmes categorizes the information. This study then further discusses the root causes of application collusion and categorizes the potential causes into four main categories i.e., push service by third parties (77%), functional SDK (15%), shared resources (5%), miscellaneous (3%). As all the research discussed above had some ambiguity along.

Therefore, along with other studies and research being done on android collusion, it was identified that Android OS provided some of the features itself that enabled applications to access data that was only allowed for limited and highly privileged applications such as READ_WORLD_MODE, WRITE_WORLD_MODE. [20] Using these two flags in any application installed in a device, allows all other applications to read its content and even modify it. Further in year 2020, new research represented a model that analyzed and detected only potentially collusive applications which helped to reduce the time consumed to detect all the benign applications as well. Then a function was used that tracked the flow of sensitive information. If any flow of sensitive information through an application ended up in a shared resource, then that specific application was marked as collusive. The function used in method works only on detection of any two applications. [21] Another paper presented somehow same thing but used K-means algorithms and some linear of SVMs in order to learn about behavior of malicious and benign applications. It then uses vector parameters with the concept of potential colluding applications will pose same threats as malwares. Along with this, a simple decision function was used to detect collusion between applications. [22] Furthermore, in 2020, a study showed that detecting or analyzing single application at a time does not exhibit any collusive property. Therefore, in order to detect malicious collusion between applications, it is important to create pairs based on their possibly collusive features. In this way the study further carried out an experiment by creating pairs of applications based on the feature of SharedPreferences (), this object in an application enables it to view key-value pairs in other applications and provides a simple way to read and write over those values with the help of GET and PUT method. This study really benefits on detecting colluding applications.[23] Similarly another paper that benefits this study was presented this year, the study presents an architectural design and

implementation for ContentAnalyzer to detect sensitive information leakage and prevention in Android devices. It performed static and dynamic analysis of suspicious applications for collusion. Then with the help of these static and dynamic combinations, illegal data leakage was discovered. [24] Even though the approach and analysis in ContentAnalyzer appeared to be effective, some of the drawbacks of this implementation were slowing down over all system's performance. Moreover, it required pre-installation of the application on the target device which means that ContentAnalyzer could not detect or prevent information leakage between applications that were installed prior to ContentAnalyzer. [25]

A brief explanation of all the papers studied for detection of malicious applications and colluding applications are presented in the table below

Title	Description	Year
A survey of malware detection in Android apps: Recommendations and perspectives for future research	Provides a detail of all the tools presented to detect malwares in android apps and discusses drawbacks of each of them. Moreover, this paper also provides recommendations as solutions for their drawbacks for future research	2021
Multi-app security analysis with fuse: Statically detecting android app collusion	Proposed a model that analyzes and stores relatable information of every single application installed then combines all information together and detects collusion between apps based on a policy.	2014
On the need of precise inter-app ICC classification for detecting Android malware collusions	Discusses why a detection solution is required for application collusion and further presents a map of all the possible communication maps of android apps that could benefit to statically track and characterize app collusion based on ICC	2015
Intersection automata-based model for android application collusion	Proposes a model that tracks all components communication and presents it in a form of state machine to detect application collusion.	2016
Appholmes: Detecting and characterizing app collusion among third-party android markets	Introduces a model AppHolmes that extracts smali code and information from manifest file and then combines all related information to analyze all applications statically to classify them either collusive or benign.	2017
Malicious Collusion Detection in Mobile Environment by means of Model Checking	Introduced a model that analyzed and detected only potentially collusive applications. A function was used that tracked the flow of sensitive information. If any flow of sensitive information through an application ended up in a shared resource, then that specific application was marked as collusive	2020
Hybrid classification model to detect android application-collusion	This paper used K-means algorithms and some linear SVMs in order to learn about behavior of malicious and benign applications. It then uses vector parameters with the concept of potential colluding applications will pose same threats as malwares. Along with this, a simple decision function was	2020

	used to detect collusion between applications	
Android Collusion: Detecting Malicious Applications Inter-Communication through SharedPreferences	Detects app collusion by creating pairs of applications based on the feature of SharedPreferences (), this object in an application enables it to view key-value pairs in other applications and provides a simple way to read and write over those values with the help of GET and PUT method.	2020
Implementation of contentanalyzer for information leakage detection and prevention on android smart devices	Presents an architectural design and implementation for ContentAnalyzer to detect sensitive information leakage and prevention in Android devices. It performed static and dynamic analysis of suspicious applications for collusion.	2021

Table 2 - Application Collusion detection tools

CHAPTER 3

EXPERIMENTAL SETUP AND SCENARIOS

3.1 INTRODUCTION

All the solutions or tools that were presented in research previously focused on external elements i.e., code designed by the attackers in malicious applications. These codes were then extracted for further info to detect application collusion. However, there are various internal features i.e., features supported by Android OS itself that could be used potentially by attackers or intruders to perform application collusion or any malicious activity. As discussed in chapter 2, READ_WORLD_MODE is a flag that was supported by Android OS. Whenever a file is created in an application with SharedPreferences (), openFileOutput (), openOrCreateDatabase (String, int, SQLiteDatabase.CursorFactory), adding this flag along enabled all the other packages to read its content. Similarly WRITE_MODE_WORLD is a flag that enabled all the other packages to write or modify file content though the file is still owned by the application to which it was created in but due to these flags the file becomes global for all other applications installed to read or write its content. Thus, there are several features presented by Android which can be misused by attackers to tackle authentication and security in mobile phones.

Therefore, this study focuses on exploring particularly Android features that help attackers to evade mobile authentication to perform application collusion attack. Furthermore, a preventative solution is introduced that could assist future researchers for developing tools from opposing apps to collude. This chapter is divided in two main parts Attack methodology and defensive measures. Attack methodology discusses Android features that are favorable for intruders to gain unauthorized access to user's sensitive information. Moreover, it includes how to perform application collusion attack with such features. The other section offers a probability framework on how to identify potential colluding applications in a mobile device.

This research presents an attack based on a feature provided by Android i.e., SharedUserId. This research aims to develop two android applications and perform application collusion among them. During the literature review stage, it was observed that a study conducted research in 2020 on locating potential collusive apps on the basis of using SharedPreferences () object. In consideration to that, this study targets on a different shared object feature in android which is sharedsuerid (). A userid in an application is simply a unique identifier. With this userid the OS identifies the application to share resources and services accordingly. In Android an application is assigned a userid by default, but it can also be assigned explicitly by

the developer in the manifest file while developing the app in Android Studio. After an application is installed in a device, the device's OS looks up in the manifest file for a userid to classify that application in order to perform activities, share resources and allow permissions.

3.2 ANDROID ARCHITECTURE

This Linux based OS architecture is stacked with software components and is divided into four main layers. The five software components are discussed below.

- Linux kernel
- Libraries
- Android runtime
- Application Framework
- Applications

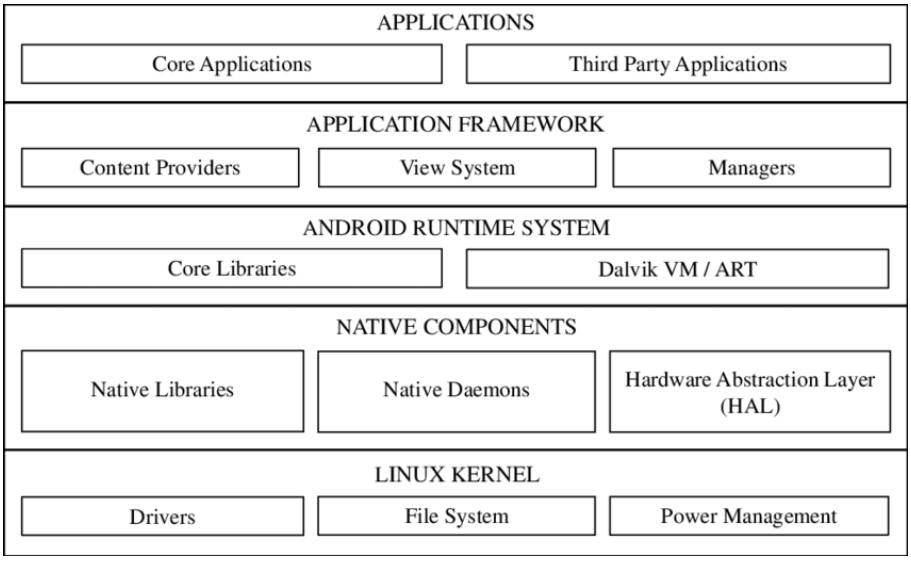

Figure 4 - Android OS Architecture

3.2.1 LINUX KERNAL

Linux kernel is responsible for managing input and output requests from the software. It also manages basic functionalities of the system such as process management, memory and device management for camera, keypad, display etc.

3.2.2 LIBRARIES

Above Linux Kernel, there is a set of libraries consisting open-source web browsers for example WebKit, library libc. These libraries are used for playing audio and video recordings. The SQLite is a database which is used for storing and sharing application data. Additionally, SSL libraries are responsible for internet security.

3.2.3 ANDROID RUNTIME

This component is responsible to provide Dalvik Virtual Machine (DVM) which is a virtual machine-like java, specially designed and optimized for Android. It has to process the virtual machine in Android OS so that the apps can run on Android devices. Dalvik VM utilizes the core features of Linux like memory management and multithreading. It also enables apps to run in their own processes.

3.2.4 APPLICATION FRAMEWORK

This component is in the next layer which provides several high-level services to apps such as windows manager, view system, package manager and resource manager etc. The app developers are permitted to use these services for their applications.

3.2.5 APPLICATION

This layer is on top, and all the applications are written and installed on it such as contacts, browsers, services books etc. Each of these applications perform a different type of role in over all applications.

The basic architecture and flow of how an application after installation in a device is identified by the OS through user id and how the application is allowed to access user data based on permissions mentioned in manifest file, is explained below with the help of sequence diagram.

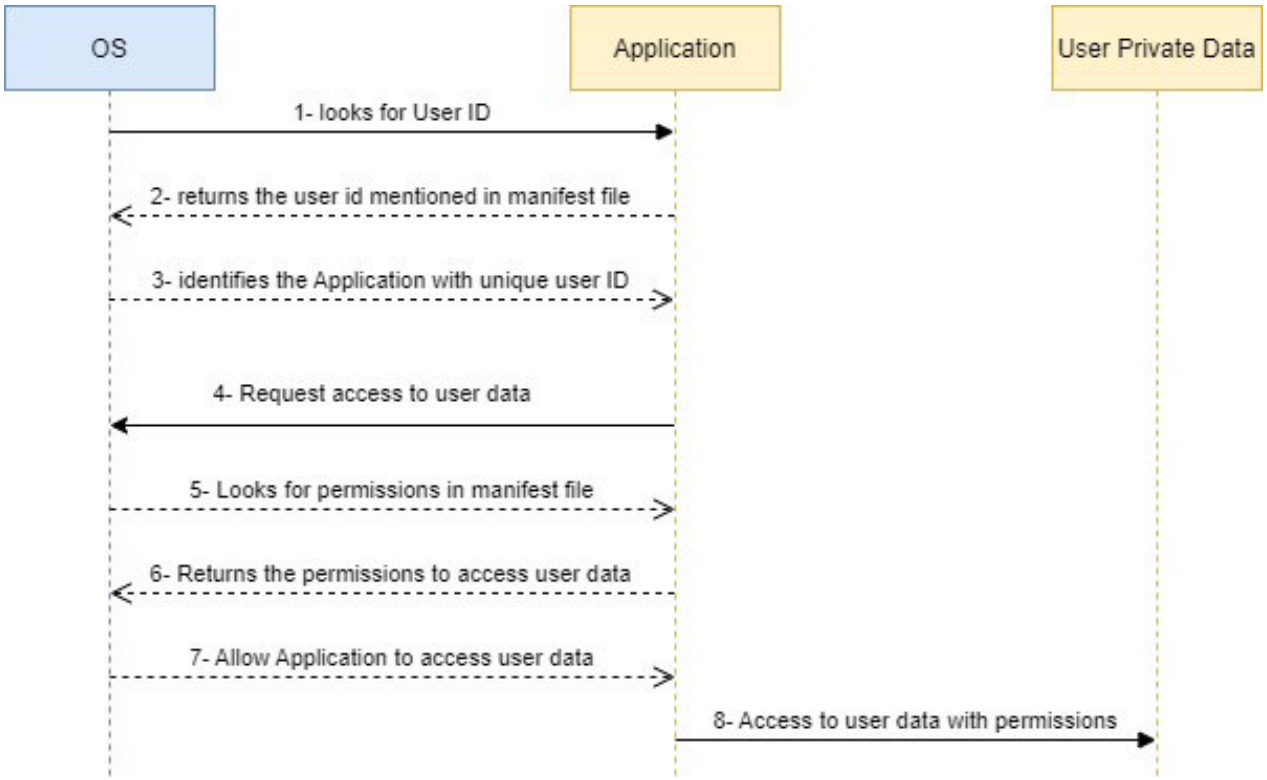

Figure 5 - application access to user data in a sequence

The diagram illustrates the process of identifying an application when installed in a device. Each step is described below

1- Once the app is installed, the OS looks up for the unique identifier also known as userid.
2- The userid is declared in the manifest file of the app.
3- After the userid is discovered, the app is identified by OS.
4- Since the app needs to access user data to perform some functions in order to provide services to user, it requests the access in form of permissions.
5- OS looks up for the permissions in the manifest files
6- Data access permissions are stated in manifest file of the app.
7- After the user allows these permissions, OS allows the app to access user data.
8- App accesses user data to perform actions.

3.3 ATTACK METHODOLOGY

Every android application is assigned a user id, this user id is a unique identifier for an application. The motive behind assigning each app with its own unique user id is that no application could use resources of any another application, nor they can run on each other's processes or activities. Therefore, when an activity is in running and a new activity of another

application is called, the control is passed to the new activity and both activities run on different processes. However, Android gives option for developers to create applications with same user id. When applications have same user id, they can share each other's resources, permissions, fields etc. Moreover, applications can run data of other applications in their processes as well. This can be done by explicitly declaring same user id using Android feature sharedUserId in both applications' manifest files, provided that both applications are developed by same developer and have same signature certificate. After mentioning sharedUserId, both apps can share permissions, resources, data, fields and much more.

Tools/Setup	
Android Studio	2021.2
Mobile Model	Huawei P30 Lite
Android OS Version	20.0.5
Firebase Database	10
App 1 SharedUserID	com.ncsael
App 2 SharedUserID	com.ncsael

Table 3 - Tools and software used

Proposed attack methodology work is consisted of several steps. Since our research is based on android, we used Android Studio tool as a simulator. Android studio is an open source and easy-to-understand tool with minimal human efforts required. For developing the two apps, programming language Java is used, it's a high-level programming language and is easily understandable. The aim is to develop two android apps with different group of permissions but design them in such a way that these applications when installed on any device together, communicate to share each other's permissions to perform collusion attack. It is necessary to make sure that the targeted user installs both of our developed apps in his device. After both apps are installed only then android application collusion attack can be accomplished. The process and series of steps included while performing application collusion attack within two applications in Android is explained below with the help of block diagram.

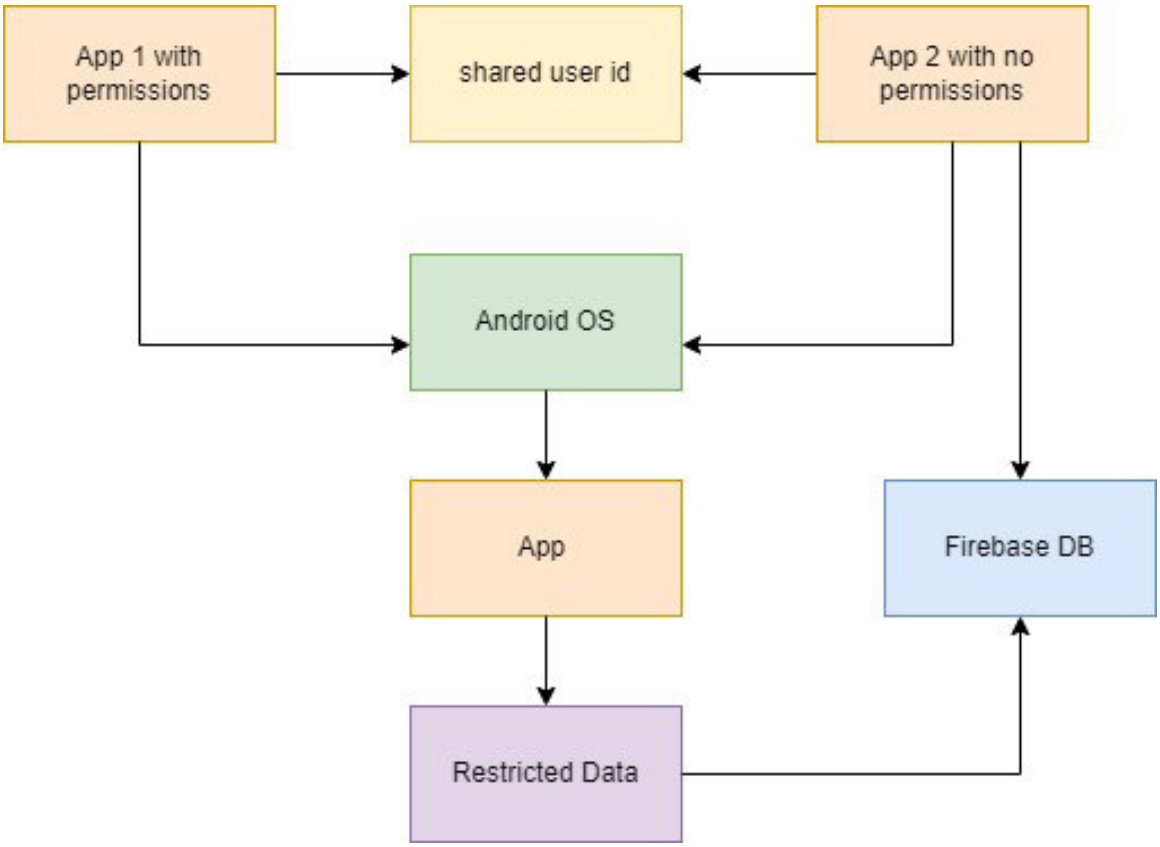

Figure 6 - Attack Methodology Flow Chart

The steps taken in this proposed attack are explained below:

1- App 1 and App 2 are developed with same userid using shareduserid feature of Android.
2- App 1 is given permissions to access user data.
3- App 2 has no permission assigned but is connected with firebase.
4- After both apps are installed on the device, the OS fails to distinguish them as two different apps and instead identifies them as generic individual App due to same user id.
5- This generic App has permissions of App 1 and connectivity to firebase of App 2.
6- Hence App 2 can access permissions of App 1 and access restricted data of user illegally.
7- Lastly App 2 uses the connectivity with firebase to send this accessed data to store and maintain for malicious purpose.

To perform this attack, the applications were developed in Android Studio using high level programming language java and were named as App1 and App2. Both apps contain an object SharedUserId in their manifest files and are given the same user id i.e., “com.ncsael”. App1

has number of permissions assigned such as to read user's contacts, read call logs, get all accounts info stored in user's device etc.

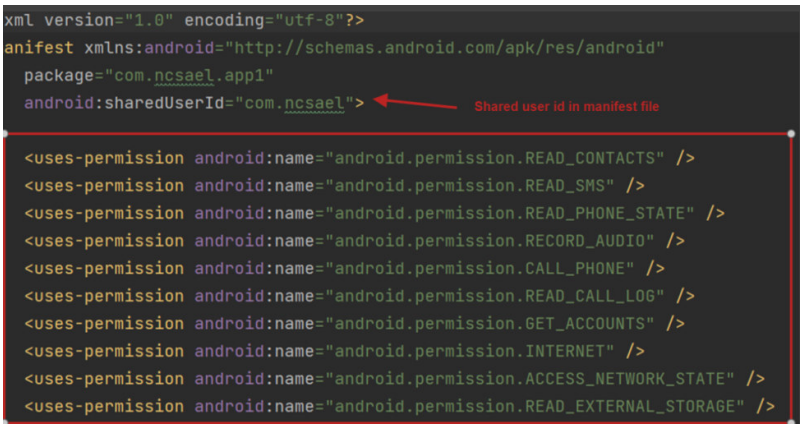

Figure 7 - App 1 Shared user id with permissions in manifest file

However, App2 is not assigned to any permission and in its manifest file it only has the object SharedUserid with user id same as of App1 "com.ncsael"

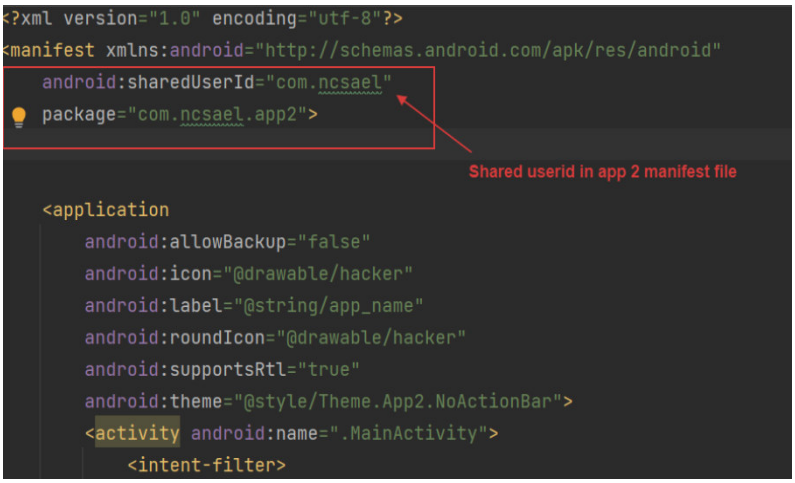

Figure 8 - App 2 Shared user id with no permission

But at the same time, app 2 is connected to an external realtime DB to send out call log info, contacts and accounts info and store them in Firebase even though app 2 has no permission to access any of this user info.

```
public class MainActivity extends AppCompatActivity {
    TextView contactview, calllogs, accounts;
    String uname;
    private DatabaseReference mDatabase;
    private DatabaseReference contactsDatadbasereference;
    private DatabaseReference callogsDatadbasereference;
    private DatabaseReference accountsDatadbasereference;
    private DatabaseReference permissionsDatadbasereference;
    private DatabaseReference tagCloudEndPoint;

    private ProgressDialog loading;
    private ImageView imgCongrats;
    List<Contacts> contactsList =new ArrayList<>();
    List<CallLogs> callLogsList =new ArrayList<>();
    List<Accounts> accountsList =new ArrayList<>();
    List<String> permissionsList =new ArrayList<>();
```

Figure 9 - App 2 Connection to Database

When a user downloads app1, The UI of app1 contains a message and a button. The message shown says that a famous sports brand Nike on the occasion of their 55th anniversary is giving away free cash prizes and rewards, to avail the chance click the button below to download official app. Once the button is clicked, it generates a link that seems like a legitimate NIKE site URL, but it redirects user to a different website directing to download App 2.

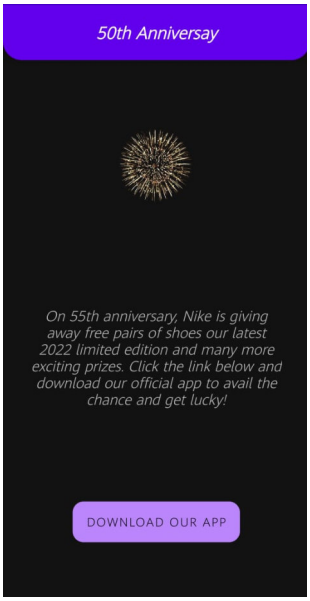

Figure 10 - App 1 UI with a message

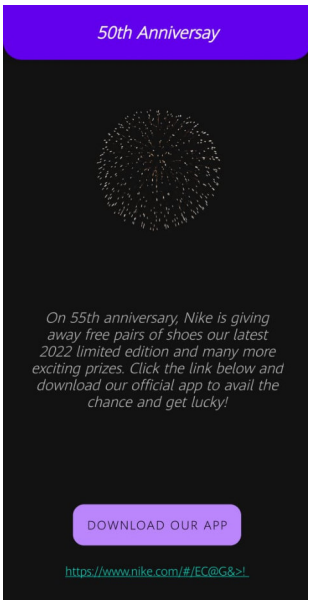

Figure 11 - App 1 Link generation after clicking button

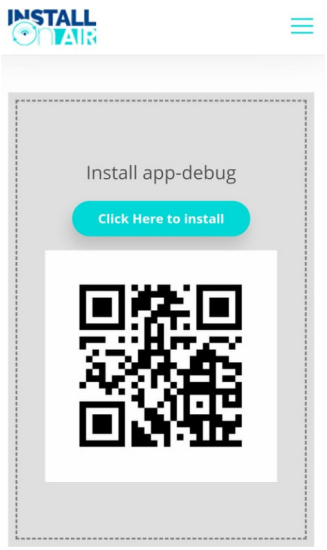

Figure 12 - Redirection to a site with app 2 apk

App2 has almost a similar UI consisting of a message and a button. It is not assigned any permission; however, it is connected to a firebase DB in the backend. Since both app1 and app2 are given the same user id. On the frontend, it only contains a button and a message saying “Congratulations, you are our 50th lucky user and we are happy to declare you that you have won 50,000 USD. That’s not all, we are giving you 25 of our limited editions 2022 Nike shoes. We hope that you are prepared for great changes that will come to your life soon. Enjoy and stay safe. Click below to provide your account information and address so that we can wire out your cash prize and your reward”. When the button was clicked, on the backend, it has been coded in a way that when user presses the button, it uses permissions of app1 to read contacts, call logs and accounts residing in the device and uses the connection to firebase

to send out all the accessed data to be stored. Attached below is the screenshot of app 2 and the message it shows.

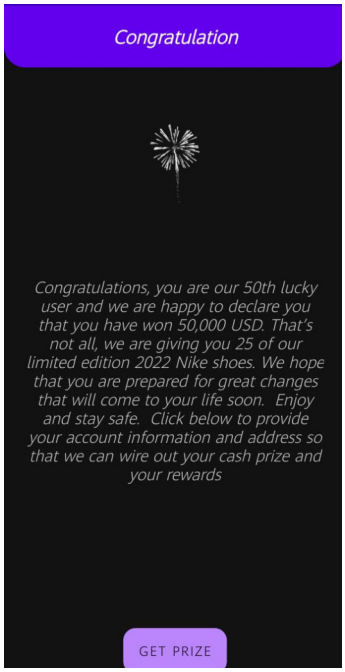

Figure 13 - App 2 UI with a button having firebase connectivity

The table below displays what resources were shared with each application

App Name	Permissions	Firebase connectivity
App 1	• Contacts • Call logs • Accounts	-
App 2	-	Connected

Table 4 - Resources for each apps

CHAPTER 4

RESULTS AND ANALYSIS

4.1 INTRODUCTION

Our evaluation is based on the accuracy and effectiveness of presented work. The presented research provided an efficient method to evade Android OS authentication and gaining unauthorized access to restricted data using Android built-in feature. The efficiency of this work can be evaluated by developing two Android applications one with access permission to user's restricted data and the other with no permission to access at all. Once launched in an Android device, the OS gives permissions of first app to another app which had no permissions defined. This clearly proves that Android authentication has been compromised with sharing permissions and applications could gain unauthorized access to user's restricted data. The restricted data being revealed and being accessed by apps without any authorized permissions assigned, leaves a huge question mark on Android security measures.

4.2 EFFECTIVENESS

In this proposed work, it is shown how an application collusion attack is performed using nothing, but Android built- in feature of SharedUserId to gain unauthorized access to user's restricted data with invading Android OS authentication process. Furthermore, it is also shown how the accessed data is sent out from compromised device to a firebase Db without user's knowledge. The effectiveness can be calculated by examining the stored data residing in Firebase. Firstly, as both apps App1 (with permissions defined) and App2 (with no permissions defined but connected to Firebase at backend) are installed in a device. Installation of both applications in a same device can be achieved by luring user through phishing emails, adds on malicious websites. Usually, malicious applications or applications that are developed by attackers cannot be found on Google Play Store. Attackers make these applications available on third party sources and trap users by promising to provide them such services which are not usually found free on any platform for example Tube Mate (YouTube videos downloader). After the user downloads and installs such malicious applications from third party sources it is quite possible that attacker gains some access to user's data and perform illegal actions.

In our presented attack user can download App1 from any phishing mail by being trapped by a message saying Nike is giving away cash prizes. After app1 is installed on user's device, the app asks user is shown a URL to visit to download official app i.e., app 2. Both app1 and app 2 share the same user id and due to that Android OS allows App1 and App2 to share resources

and permissions etc. Now App 2 is designed in such a way that when a user clicks the provided button, all the accessed data using permissions of App1 is sent to firebase. The data sent to firebase includes call logs, contacts, accounts info stored in device.

4.3 EVALUATION AND ANALYSIS

The data accessed is stored in Firebase which is a cloud-based real time database to store and sync data between users in real-time. It enables to store apps data and maintains data storage such as syncing and query app data at a global scale. The picture below explains how user's data is stored in three categories Accounts, Call Logs and Contacts.

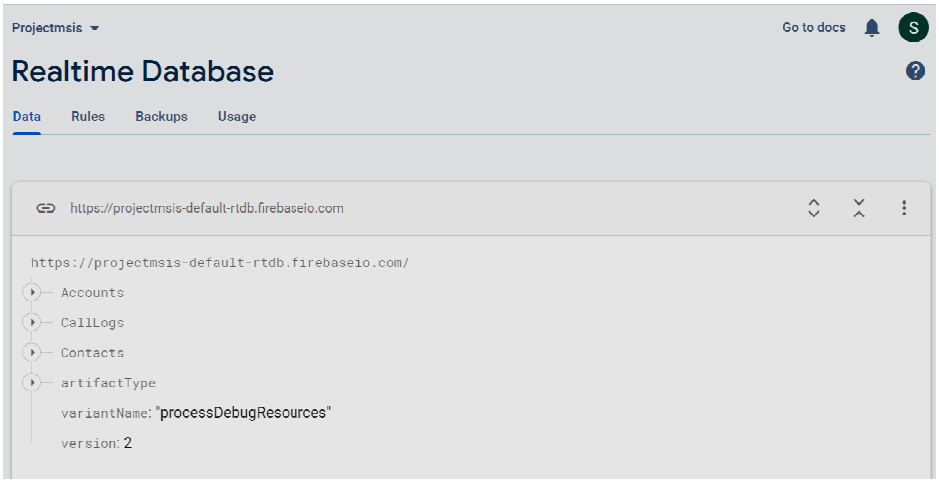

Figure 14 - Data stored in firebase in 3 different categories

The Accounts category has three things included when stored i.e., full email, id with which it was saved in the device and name of the user.

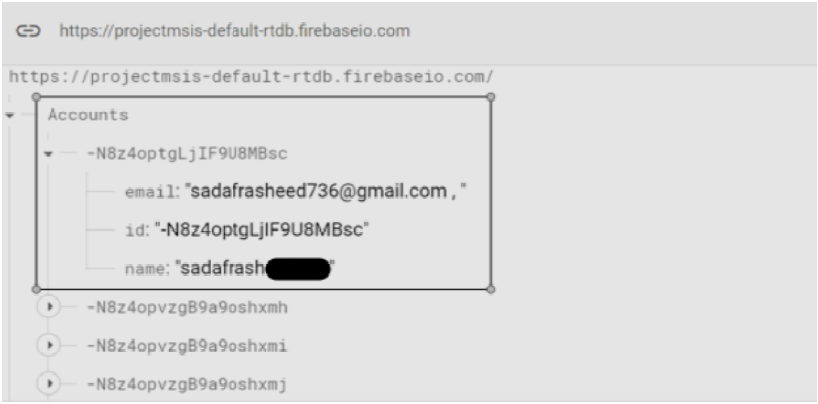

Figure 15 - realtime account ids saved in device

Second category is Call logs, Firebase stores date of call, call duration, caller id, name, phone number and call type of each entry stored in call log. There are three types of call types

outgoing, incoming, and missed. Incoming call type is assigned a value of 1 while storing entry in firebase, whereas outgoing calls are assigned value of 2 and calls that are missed are assigned a value of 0.

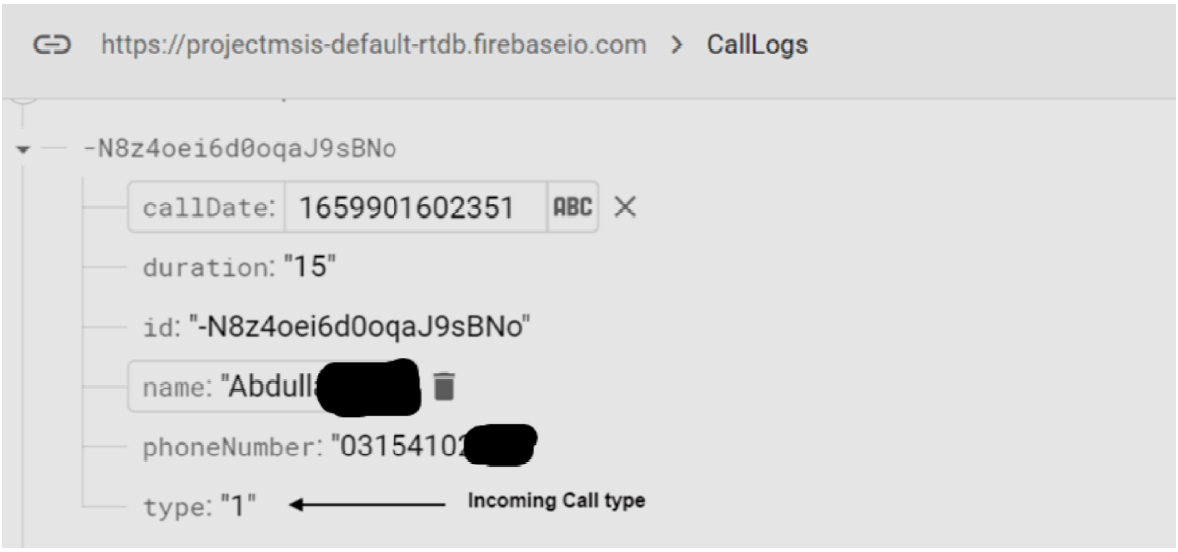

Figure 16 - call log of incoming call

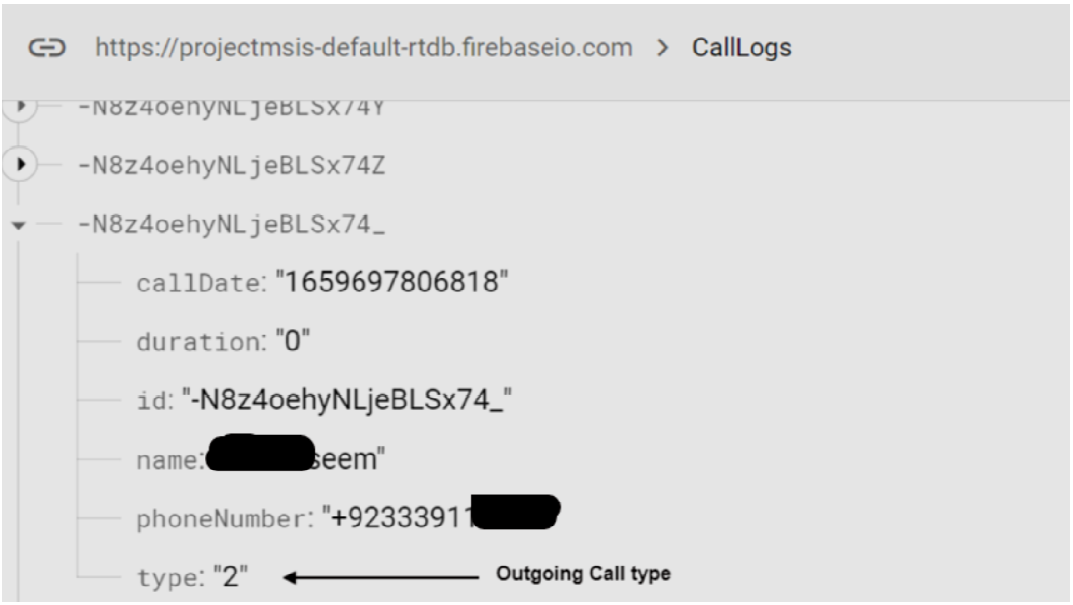

Figure 17 - Call log of outgoing call

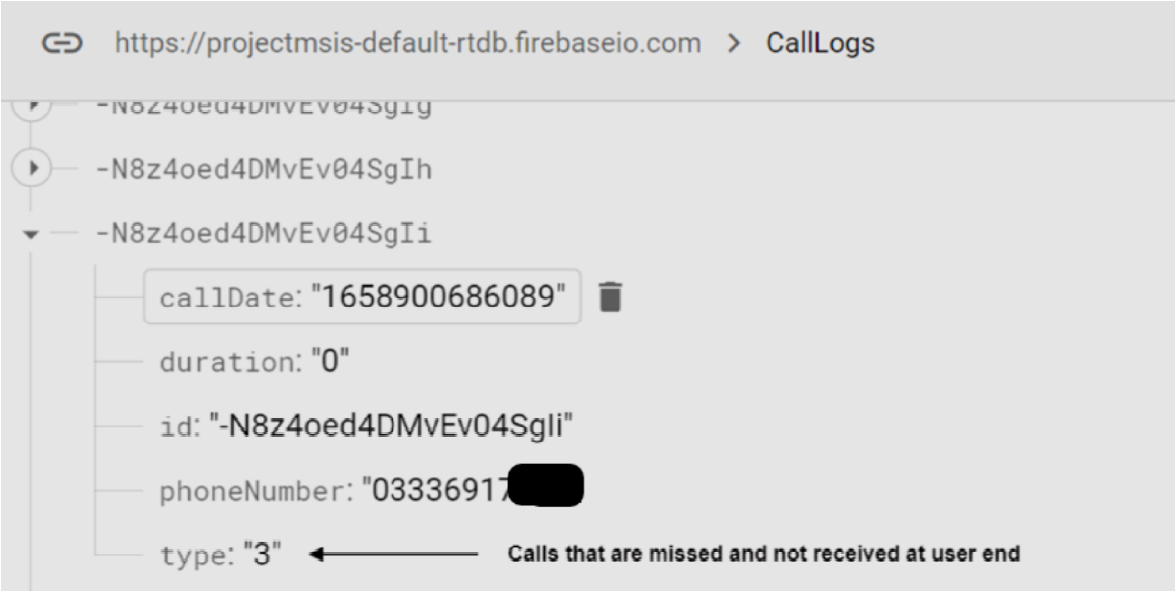

Figure 18 - Call log of missed calls

Duration is saved in seconds and Call date is stroed as String date that can be converted into timestamp.

Third category of data stored in firebase is Contacts with storing contact number, contact ID and contact name.

Figure 19 - Saved contact in device

4.4 DEFENCE MECHANISM

After the proposed attack methodology, we present two defense solution in respect to that. As we have seen it is difficult to detect android collusion if applications are sharing user id. By storing set of permissions of each app into firebase, we can prevent from occurrence of our proposed attack methodology. The diagram below explains each step taken to prevent from this attack.

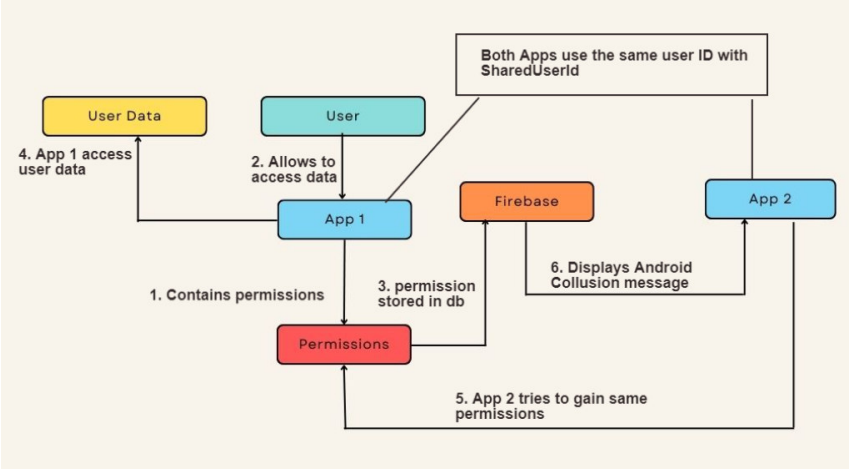

Figure 20 - Defense solution Workflow

1- Both apps share the same user id and app 1 contains some permissions to access user data.
2- User allows those permissions to app 1.
3- After user allows these permissions, they are stored in firebase.
4- App 1 accesses user data based on the allowed permissions.
5- App 2 tries to gain access to same set permissions
6- It will show a warning message about app 2 using permission set of app 1.

While it is seen in the proposed attack that when both applications app 1 and app 2 are installed in the device sharing same user id, their permissions, resources, and data everything is shared between them. As a defense we have proposed a solution that on installation of every application, their set of permissions can be stored in firebase and whenever any other application tries to access those specific set of permissions, it shows a warning message of application collusion. Below attached screenshots represent how permissions are stored in firebase and what warning message it shows.

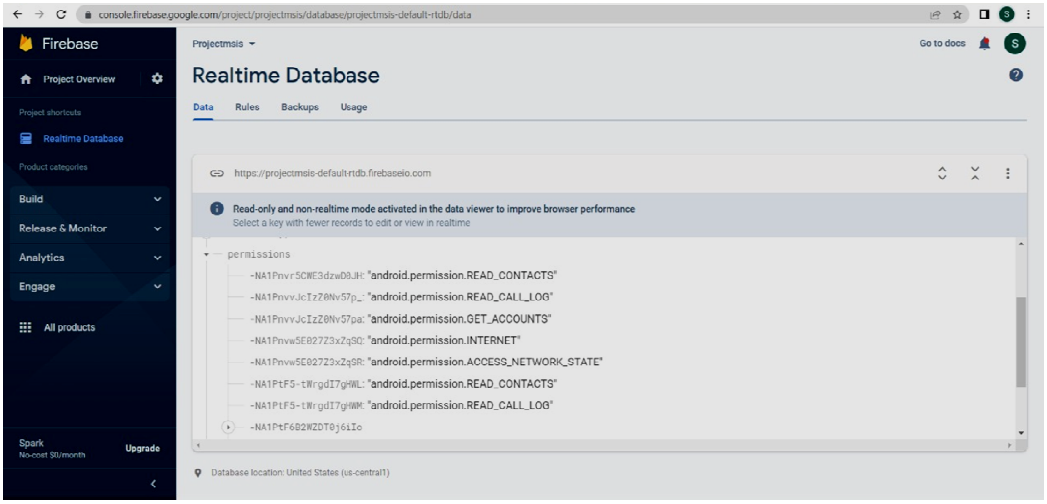

Figure 21 - Set of permissions stored in firebase

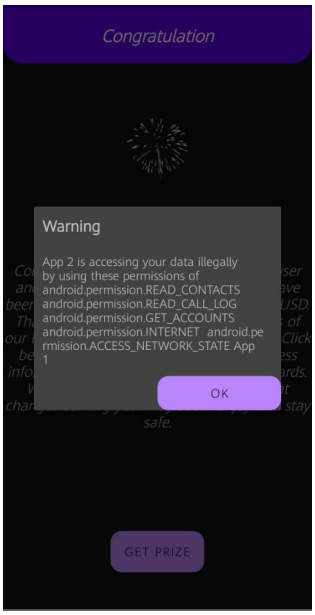

Figure 22 - Warning message when app 2 is opened

Another preventative solution that we have come up with is probability framework, since it is difficult to detect colluding apps because of some legitimate features of Android such as shared preferences, shareduserid etc. We have grouped together all the potential colluding features that can be used by attackers while developing the apps to evade Android security and perform application collusion. After grouping these features, we have assigned each one of them an absolute value and represented a theoretical probability framework to calculate their collusion degree. As we are focused on collusion attacks with same developer, signature, and user id. That is why, X in the table is kept as constant

Features	Assigned value
Same developer + Same signature certificate + Shared user id	X
IF (App has → demystified permission)	1 (Assign)
IF (App Uses → Shared preference)	2 (Assign)
IF (App Uses → DB access flag)	3 (Assign)
IF (App creates → Data backup)	4 (Assign)
IF (App sends → out data)	5 (Assign)

Table 5 - Potential Colluding Features in Android

After a comprehensive scan and acquired results, tested application is further classified into 4 major categories based on their probability. These categories are Vulnerable, Suspicious, Risky, Critical, and Under Attack.

Probability Equation	Values	Percentage	Probability (P) (0-1)	Probability Value	Application Status	Collusion Degree
P = X+1	Demystified permission	20%	0.2	Low	Vulnerable	Minor
P = X+2	Shared preferences	40%	0.4	Average	Suspicious	Slight
P = X+3	DB access flag	60%	0.6	Moderate	Risky	Dangerous
P = X+4	Data backup	80%	0.8	High	Critical	Dangerous
P = X+5	Send out data	100%	1	Max	Under Attack	Unsafe

Table 6 – Colluding probability of each feature set

4.5 DISCUSSION

After the evaluation of stored data, it can be proved that stored information in firebase is accurate and effective. With just one feature of sharing same user id, apps were allowed to share permission and with this, any malicious app residing in mobile device can take advantage of it. As it is mentioned in previous chapters that application collusion attack can only take place when there is more than one app included, which means that a combination of at least two apps is required in our proposed work, and it must have the same user id. In our methodology we used one app to show message and lure the user into believing that he may have a chance to win a cash prize and rewards by downloading App2. But there are many

other ways through which Apps can be downloaded into targeted device. Since Google Play store only includes apps that are developed by trusted developers and publishers, such malicious apps as they have been developed by attackers and intruders, cannot be found in Google play store. However, there are several ways to get such apps with illicit intent to targeted user such as phishing mails. Through phishing mails attackers can make the mail look reliable and offer something appealing for user so that he can be convinced into believing that this mail is authentic These mails can simply ask user to download either both apps or single app at first by offering a service or feature in app that cannot be easily found in google store for example free downloading of YouTube videos. After the first app is downloaded, user grants permissions and avails the service of the app. Meanwhile advertisements and messages to download app2 can be displayed through app1. As the user has gained trust through app1, it is quite predictable for user to download app2. After app2 is downloaded with a unique service to offer, both the downloaded apps now share the same userid which means application collusion attack is performed effortlessly without user knowing anything. Likewise, such apps can be delivered through third party sources as well. There are number of sites where applications are distributed providing services that are interesting for user but in the backend have malicious purpose to gain deep level access of user's device. Even user can be asked to download both apps at once by conditioning that functionality of app1 depends on app2.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

The use of smartphones has become a necessity today. The number of its users has grown to an indefinite level. People have started switching towards smart phones for their financial businesses as well as educational and social matters. Therefore, smartphones are comprised of most of user's private data and sensitive information and for this reason, smartphones are the primary target for attackers these days. Even though smartphone manufacturers are trying their best to design an effective security model that prevents from all possible threats and overcomes its vulnerabilities, but it seems like attackers are always one step ahead of what has been presented. Same is the case in application collusion. Every application that requires to access some level of user's private data to function has to request a permission from user to access it. If permission is rejected, the application is not allowed to access user's data. This permission mechanism is provided for making users aware that their applications are accessing sensitive information. Moreover, this mechanism gives a choice for users to deny applications from accessing their personal data. However, in application collusion attack, malicious applications can take benefit of other application's vulnerabilities and use it as a tool to access user's private data without user's knowledge. Quite a lot of research have been done and many solution tools were presented previously with the aim of detecting colluding applications in Android though they all had certain limitations or conditions in order to detect or prevent application collusion efficiently. The foremost issue that is faced while constructing a solution for such an attack is there are no datasets available of colluding applications, and as a result there are no exact properties defined of how this attack works. In addition to this, some applications are comprised of actual legitimate nonthreatening code but are exploited by attackers to evade authentication. Due to this lack of information, it becomes challenging for researchers to present a solution for detecting or preventing application collusion without proper direction.

This study, therefore, starts off first with focus on discovering and analyzing in depth of most likely vulnerable features supported by Android OS that can be exploited by attackers to perform this attack and till now no solution has considered to detect such features since they are supported by Android itself. As a result, it was founded that Android allows developers to create applications by assigning them same user id. Basically, a user id is a unique identifier of an application with which they are identified by the OS. However, under a condition of same developer developing multiple applications with same keystores, Android allows

developers to assign these applications same user id. The concept of same user id was created to assist developers for making their correlated applications communicate once installed in the device but there are high chances of occurrence of application collusion attack with it. This study has shown how this same user id attribute in an application's manifest file can lead to make other applications collude with it. For this experiment, two applications were created, one was assigned a permission to read contacts while the other application was assigned no permission at all. Both of these applications however had sharedUserId in their manifest files. Once these applications were installed in a device, the OS due to their same user id, treated them as a single application and hence permissions were shared. In this way it is shown how an application with no permission assigned can still access to unauthorize sensitive data of user by misusing permissions of other applications due to having same shareduserid. Next stage of this attack was to send out the data which is accessed illegally. This was done by connecting the application to firebase. In this attack we have sent out the data with an assumption that firebase could be compromised, and user's sensitive data could be exposed to attackers. The purpose behind this was just to demonstrate how easily an application could access data it is not authorized of and how simple it is to send out the sensitive information to any malicious ip or network etc. Hence with this attack, it was determined that there are various vulnerabilities present in Android that needs to be addressed before long.

Later, concern was shifted to security, raising a question of how to detect and prevent from such colluding applications. Thus, this study then presents two defense solutions, first solution works with a concept of storing each application's set of permissions on connected database and whenever any other application tries to access those group of permission, a warning message of android collusion would pop up. Second solution presented is a hypothetical probability framework that can assist in future research to provide a guideline on what android features to look for when designing a solution for detecting these attacks. It can further provide a direction for researchers of what has and hasn't been discovered yet. Each feature in the framework is assigned a value and then different combinations are generated, after this, sum of each combination is calculated through which value of probability of an application to be collusive is calculated.

5.2 FUTURE WORK

In future this research can expand and surpass in detecting further properties of android applications and built-in features that can be exploited and misused by attackers to perform malicious activities, gain unauthorized access, or invade Android authentication process. Additionally, this attack can further be enhanced if through performing reverse engineering we get user id of any famous apps like WhatsApp, Facebook or snapchat. Since these

applications are present in almost everyone's mobile device, it would only require creating one application with same user id as of those famous applications to perform collusion attack. Plus, the hypothetical probability framework provided can be implemented practically to detect application collusion attack.

BIBLIOGRAPHY

1. What is android-2 https://www.androidauthority.com/what-is-android-328076/
2. Android OS openness -2 https://en.wikipedia.org/wiki/Android_(operating_system)
3. Android applications and permissions https://developer.android.com/guide/topics/permissions/overview
4. "Permissions on Android," *Android Developers*. https://developer.android.com/guide/topics/permissions/overview
5. Android malware and attack main objective https://searchmobilecomputing.techtarget.com/definition/mobile-malware
6. Android Attack types https://www.greycampus.com/opencampus/ethical-hacking/types-of-android-attacks
7. Android Malware types https://developers.google.com/android/play-protect/phacategories
8. Wang, S., Chen, Z., Yan, Q., Ji, K., Peng, L., Yang, B. and Conti, M., 2020. Deep and broad URL feature mining for android malware detection. Information Sciences, 513, pp.600-613. -2 (deep URL feature mining)
9. Jung, J., Lim, K., Kim, B., Cho, S.J., Han, S. and Suh, K., 2019, June. Detecting malicious android apps using the popularity and relations of apis. In 2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE) (pp. 309-312). IEEE. (Popularity APIs)
10. Arora, A., Peddoju, S.K. and Conti, M., 2019. Permpair: Android malware detection using permission pairs. IEEE Transactions on Information Forensics and Security, 15, pp.1968-1982. (Permission Permpair)
11. Sahay, S.K. and Sharma, A., 2019. A survey on the detection of android malicious apps. In Advances in Computer Communication and Computational Sciences (pp. 437-446). Springer, Singapore. (survey)
12. Wang, W., Zhao, M., Gao, Z., Xu, G., Xian, H., Li, Y. and Zhang, X., 2019. Constructing features for detecting android malicious applications: issues, taxonomy, and directions. IEEE access, 7, pp.67602-67631. (App collusion Wala)
13. Ribeiro, J., Saghezchi, F.B., Mantas, G., Rodriguez, J., Shepherd, S.J. and Abd-Alhameed, R.A., 2020. An autonomous host-based intrusion detection system for android mobile devices. Mobile Networks and Applications, 25(1), pp.164-172. (HIDS Wala)
14. Yuan, H. and Tang, Y., 2020. MADFU: An Improved Malicious Application Detection Method Based on Features Uncertainty. Entropy, 22(7), p.792. (MADFU)

15. Cai, H., Fu, X. and Hamou-Lhadj, A., 2020. A study of run-time behavioral evolution of benign versus malicious apps in android. Information and Software Technology, 122, p.106291.

16. Ravitch, T., Creswick, E.R., Tomb, A., Foltzer, A., Elliott, T. and Casburn, L., 2014, December. Multi-app security analysis with fuse: Statically detecting android app collusion. In Proceedings of the 4th Program Protection and Reverse Engineering Workshop (pp. 1-10).

17. Elish, K.O., Yao, D. and Ryder, B.G., 2015, May. On the need of precise inter-app ICC classification for detecting Android malware collusions. In Proceedings of IEEE mobile security technologies (MoST), in conjunction with the IEEE symposium on security and privacy.

18. Bhandari, S., Laxmi, V., Zemmari, A. and Gaur, M.S., 2016, March. Intersection automata-based model for android application collusion. In 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA) (pp. 901-908). IEEE.

19. T. Chen, Q. Mao, Y. Yang, M. Lv, and J. Zhu, "TinyDroid: A Lightweight and Efficient Model for Android Malware Detection and Classification," Mob. Inf. Syst., vol. 2018, pp. 1–9, Oct. 2018, doi: 10.1155/2018/4157156.

20. "Dynalog: an automated dynamic analysis framework for characterizing android applications," in 2016 International Conference On Cyber Security And Protection Of Digital Services (Cyber Security), London, United Kingdom, Jun. 2016, pp. 1–8, doi: 10.1109/CyberSecPODS.2016.7502337.

21. K. Xu, Y. Li, and R. H. Deng, "ICCDetector: ICC-Based Malware Detection on Android," IEEE Trans. Inf. Forensics Secur., vol. 11, no. 6, pp. 1252–1264, Jun. 2016, doi: 10.1109/TIFS.2016.2523912.

22. K. A. Talha, D. I. Alper, and C. Aydin, "APK Auditor: Permission-based Android malware detection system," Digit. Investig., vol. 13, pp. 1–14, Jun. 2015, doi: 10.1016/j.diin.2015.01.001.

23. E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "MalDozer: Automatic framework for android malware detection using deep learning," Digit. Investig., vol. 24, pp. S48–S59, Mar. 2018, doi: 10.1016/j.diin.2018.01.007.

24. S. Hassan, C. Tantithamthavorn, C.-P. Bezemer, and A. E. Hassan, "Studying the dialogue between users and developers of free apps in the google play store," in Proceedings of the 40th International Conference on Software Engineering, Gothenburg Sweden, May 2018, pp. 164–164, doi: 10.1145/3180155.3182523.

25. W. Y. Lee, J. Saxe, and R. Harang, "SeqDroid: Obfuscated Android Malware Detection Using Stacked Convolutional and Recurrent Neural Networks," in Deep Learning

Applications for Cyber Security, M. Alazab and M. Tang, Eds. Cham: Springer International Publishing, 2019, pp. 197–210.

26. K. Xu, Y. Li, R. Deng, K. Chen, and J. Xu, "DroidEvolver: Self-Evolving Android Malware Detection System," in 2019 IEEE European Symposium on Security and Privacy (EuroS&P), Stockholm, Sweden, Jun. 2019, pp. 47–62, doi: 10.1109/EuroSP.2019.00014.

27. J. Xu, Y. Li, R. Deng, and K. Xu, "SDAC: A Slow-Aging Solution for Android Malware Detection Using Semantic Distance Based API Clustering," IEEE Trans. Dependable Secure Comput., pp. 1–1, 2020, doi: 10.1109/TDSC.2020.3005088.

28. M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "DL-Droid: Deep learning based android malware detection using real devices," Comput. Secur., vol. 89, p. 101663, Feb. 2020, doi: 10.1016/j.cose.2019.101663.

29. "Firebase." https://firebase.google.com/ (accessed Feb. 11, 2022).

30. "Developer Policy Center." https://play.google.com/about/developer-content-policy/ (accessed July 20, 2022).