

# Controlling Data Leaks from Pre-Installed Android Applications

MSc Research Project  
MSc in Cybersecurity

Rohan Singh  
Student ID: 20105606

School of Computing  
National College of Ireland

Supervisor: Niall Heffernan

**National College of Ireland**  
**MSc Project Submission Sheet**  
**School of Computing**



**Student Name:** ..... ..Rohan Singh.....  
**Student ID:** .....20105606.....  
**Programme:** .....MSc in Cybersecurity..... **Year:** ..2020-21....  
**Module:** .....Academic Internship.....  
**Supervisor:** .....Niall Heffernan.....  
**Submission Due Date:** .....11<sup>th</sup> November 2021.....  
**Project Title:** Controlling Data Leaks from Pre-Installed Android Applications  
**Word Count:** .....5648..... **Page Count:**.....20.....

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

**Signature:** .....Rohan.....  
**Date:** .....9<sup>th</sup> November 2021.....

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
<b>Attach a Moodle submission receipt of the online project submission,</b> to each project (including multiple copies).	<input type="checkbox"/>
<b>You must ensure that you retain a HARD COPY of the project,</b> both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

<b>Office Use Only</b>	
Signature:	
Date:	
Penalty Applied (if applicable):	

# Controlling Data Leaks from Pre-Installed Android Applications

Rohan Singh  
20105606

## Abstract

In the modern world every organisation success is based on the quality of data that they possess. Hence, they try to extract maximum possible ways. One of the easiest ways to get useful data is to extract from a device which is too personal and in everyday use, such as smartphone. Hence, lots of big corporates and manufacturers tries to target smartphones in the form of 'free' applications. And it has been said that everything has the price to pay hence these free applications transmit lots of personal sensitive data in the background. Fortunately, if these applications are downloaded by the user, then it can easily be removed by simply clicking on the 'uninstall' button. But the actual problem arises when these applications are marked as 'system apps' because these applications are impossible to remove and are so essential that it is required by the user to perform the basic functionality of the smartphone. So, the problem arises how we can make our smartphone secure that the personal data are not transmitted by these system apps.

## 1 Introduction

We can see that the smartphone has become the central part of our day-to-day life. According to one of the surveys conducted, it was shown that an average person spent more than five hours on mobile device every day because constantly the person is downloading and uploading a huge amount of data from his device all the time. Hence, we can imagine how much the data is being transmitted every day.

### 1.1 Source of data

We tried to find out which type of applications are more responsible for leaking the data. Hence according to our investigation result we found that most of the data were originated from the free applications which were downloaded by the user. but there were few applications which were at the system level, and they were also sending data to the manufacturer server or IT firm such as Google and Apple.

### 1.2 Possible Solutions

Now our main focus was to limit the data transmission from these system applications. Current global mobile OS market share is highly dominated by two key mobile players that is Android and iOS. iOS being a closed source we cannot do any system modifications whereas Android being an open source a lot of possible solutions can be drafted based on the circumstances. Some of the possible solutions are disabling those apps which are not required by the user, other solutions are turning off those permissions which are not required by the applications and last and foremost we can use some of the ADB tool commands in the modification of system applications. But each steps requires a deep technical knowledge. Now let us see which solution gives the most optimal result.

## 2 Related Work

The following section focus on the literature done prior on controlling data leaks from pre-installed Android applications.

The research focuses on all those preinstalled applications which are present in all the Android devices because they are making huge profit to this third party companies by selling the sensitive private data's in the background. All these organisations are successful in escaping from such type of allegations because their source code which were used to build applications are black boxed. As mentioned by Yanhui Guo and Lin Yang in Beijing University of Post and Telecommunications that it is very difficult to reveal without reverse engineering and then static detection analyzes using those Android source codes (Guo *et al.*, 2016). Hence the question arises how we can ensure our sensitive data is protected in any Android device? And is there any method we can limit the transmission of sensitive data by this system applications?

### 2.1 Research Objectives

Some objectives that need to be covered are: -

**Objective 1:** enabling the “Disable” option in any system application.

**Objective 2:** Using permission manager in such a way that it causes less frequent application crashes.

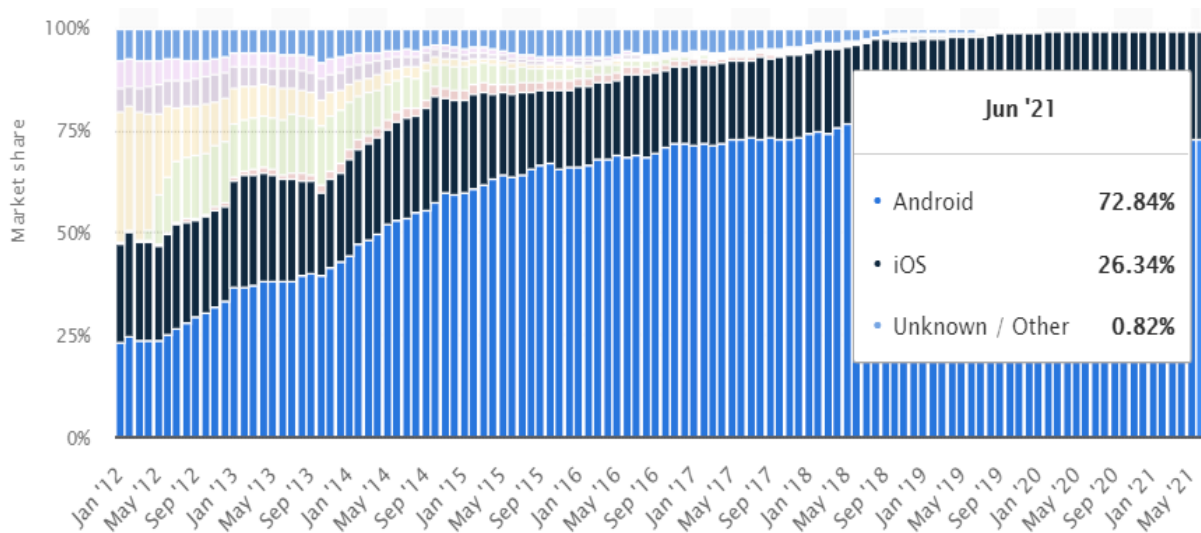
**Objective 3:** ADB command tools more transparent and simpler to use.

**Objective 4:** If possible, generate a database of those applications which are proved to be harmful for Android device in the past.

The structure of the research paper is divided into various sections and each section covers a specific topic. Section 1 deals with the abstract, introduction along with the possible solutions. Section 2 deals with the related work done in the past, literature review and the possible outcomes along with the proposed solution. Section 3 deals with the research methodology which helped in drafting a conclusion. Section 4 deals with the design specification used in each method and their outcomes in completing this research. Section 5 deals in the final implementation along with the minor drawbacks. Section 6 choose the evaluation obtained and the case study done on the final method and a brief discussion. Last but not the least, Section 7 shows the conclusion of this research and the scope of the future work.

### 2.2 A Background Overview

Before we begin this research, it is important to understand the importance of personal data for this organizations. As Thomas H. Davenport and Jill Dyche mentioned in their journal but how “Big Data” are processed and converted into profits (Thomas H. Davenport and Jill Dyche, 2013). Because of the ever-growing number of devices each year it has increase the importance of data more than ever. Therefore, companies like Google and apple started developing mobile operating system that will help in collecting data and store them directly into their server. According to the report from global mobile OS market share 2021 we can see that two companies dominate the mobile OS market and they are IOS (from Apple) which contributes 26.34% and the other one is Android (from Google) which is 72.84% and all other mobile OS contributes to 0.82% only (*Mobile OS market share 2021*, 2021).

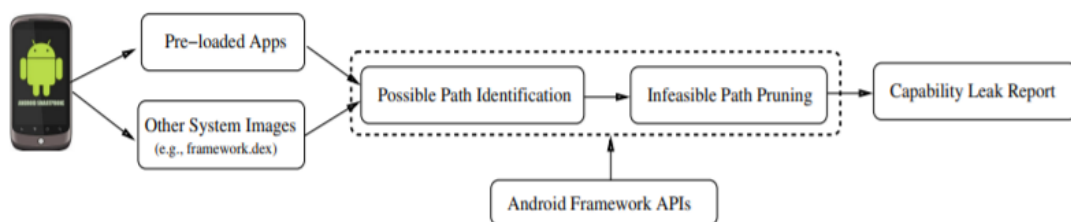


**Figure 1: Mobile Operating System Global Share 2021** (Mobile OS market share 2021, 2021)

Since the data is being collected every day hence the volume of data has increased to such an extent that it is able to reveal the human behavior pattern. These patterns are sufficient to show but what are the likes and dislikes of a particular human. And nowadays companies are publishing more of these applications in their respective App Store market (App Store market is an application from there the paid or free applications can be downloaded). Some of the free application mentions about the basic data it will transmit but they don't specify that basic data will contain their sensitive information (Almuhimedi *et al.*, 2015).

### 2.3 Sources of Data Collection

In IEEE International Conference on Social Computing, Yaniv Altshuler and Nadav Aharony revealed in MIT Media Lab that how a mobile data can accurately predict the social behavior of an individual (Altshuler *et al.*, 2012). It highlighted that various data like location, device ID, camera, contacts, call log, microphone, messages and many more were combined together and creating a meaningful pattern.

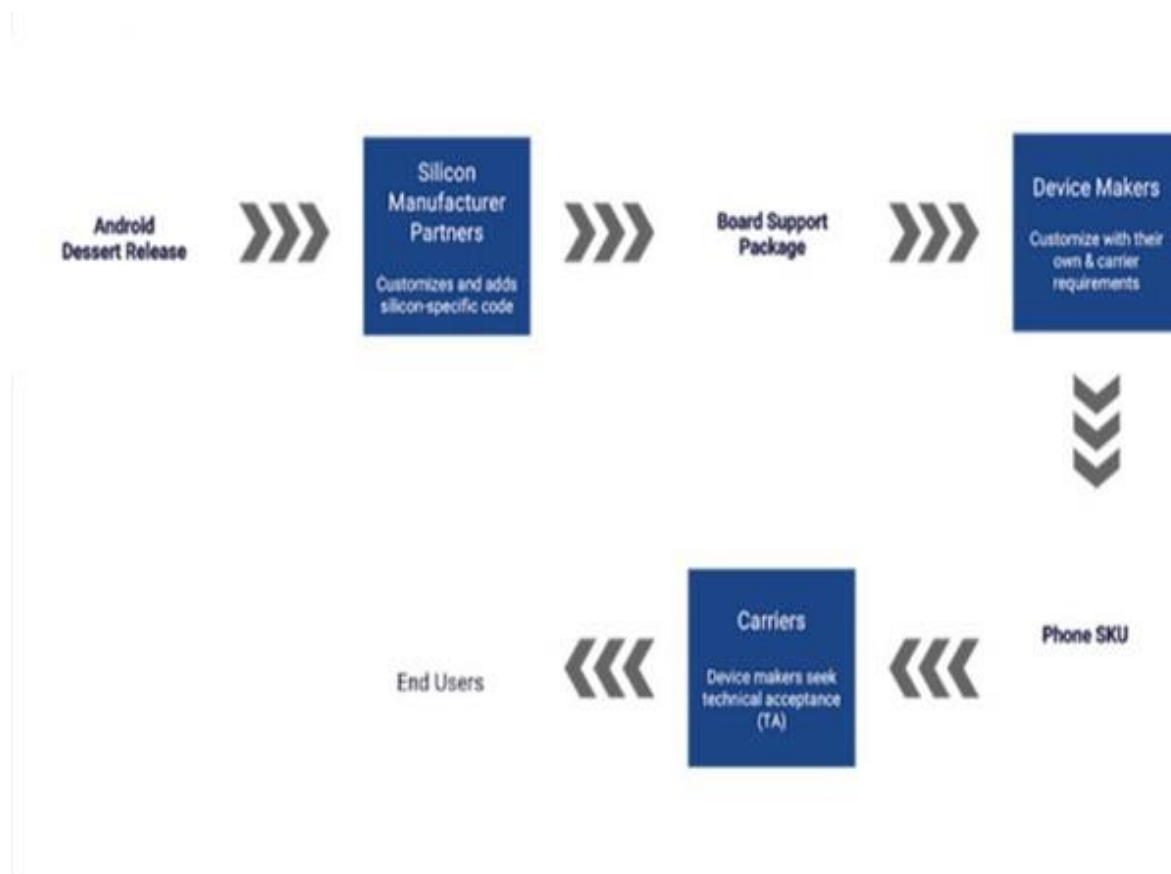


**Figure 2: Android System Design** (Han *et al.*, 2014)

Michael Grace, Yajin Zhou and Zhi Wang showed that the Google has designed on a permission based security model and each application requires the permission to access a certain personal information(Han *et al.*, 2014). These permissions are embedded at the root level and can only be modified by the manufacturers.

## 2.4 Modifications in Android by The Manufacturers

Since the beginning of Android lots of changes were made to its core system but the major changes related to privacy was introduced from Android version 4.4.4 (Famously known as KitKat version) (Khokhlov and Reznik, 2018). Android is developed as an open-source platform but the services running over the top is proprietary to Google. Because of this every time a new version of Android is developed it has to be passed from the manufacturers of the device so that they can customize the drivers which is hardware specific. During the customization, these manufacturers at their proprietary apps as in adware and then it is passed to the carrier specific companies. Again, carrier specific companies at their own proprietary apps which finally results in lots of bloatware apps.



**Figure 3: Android Customisation At Various Stages** (Sarkar *et al.*, 2019)

The functional modifications were allowed by Google so that each manufacturer can make their devices look unique and the user experience can also be enhanced by those applications. The first physical evidence was seen by Samsung when it introduced quick toggles in the notification bar. This modification was done in Android version 2.3.3 (famously known as Gingerbread) as this Android version lacked those basic functionalities. These quick toggles became so popular that from the next version of Android (which is Android 4.0) it was integrated into the framework.



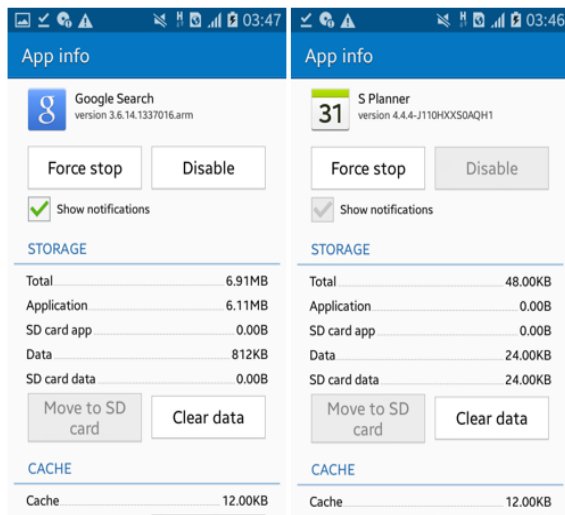
**Figure 4: Quick Toggles on Samsung Android** (Enghouse, 2016)

Slowly modifications by the manufacturers started getting popularity and this resulted in the increase in the experiments. These applications were becoming the part of the system apps and its removal were not easy (Shan, Neamtiu and Samuel, 2018). For forcefully removing of system application the device needs to be “rooted” so that the user can get admin privileges. And rooting a device needs lots of programming related knowledge (Boueiz, 2020).

## 2.5 Introduction of “Disable” In System Applications

Looking at the trend in the increase of adware applications, Google introduced few modifications which will help the general user to disable certain system applications rather than getting them uninstalled. The main benefits of disabling the applications are that it remains in the device and can be re enable by the user if required because these applications are device specific and will not be found on the play store (Ma *et al.*, 2018).

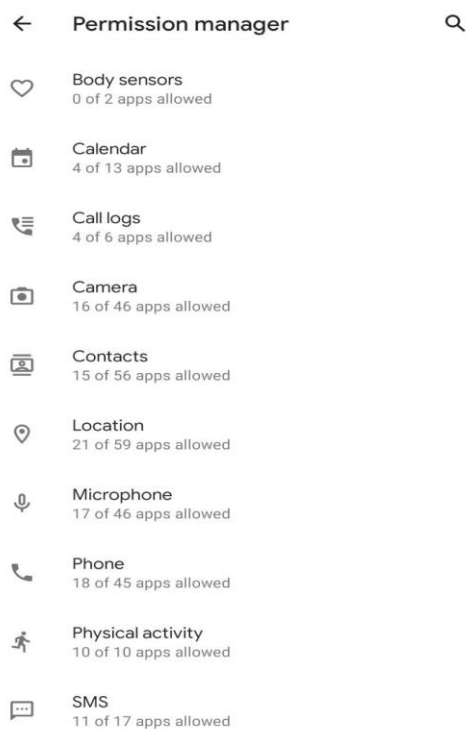
The idea introduced by Google was tremendous, but it was badly executed by the manufacturers. These manufacturers altered the applications in such a way that the disable function was removed and continued the transmission of data in the background.



**Figure 5: System Apps With Disable And The Other One Without Disable Option**

## 2.6 Permission Manager

Another major change in the Android ecosystem related to privacy was introduced in Android 6.0 (famously known as marshmallow version). A new concept “Permission Manager” was integrated in which a user was given few privileges to control the permission level of any application. Permission manager was easy to use as each applications broadcasting channels were integrated with two options and they were “Allow” or “Deny” permission rights. If a user feels to disable any of the broadcasting channel, he can deny those permissions and still continue to use the applications (Kruz *et al.*, 2017).



**Figure 6: Permission Manager User Interface**



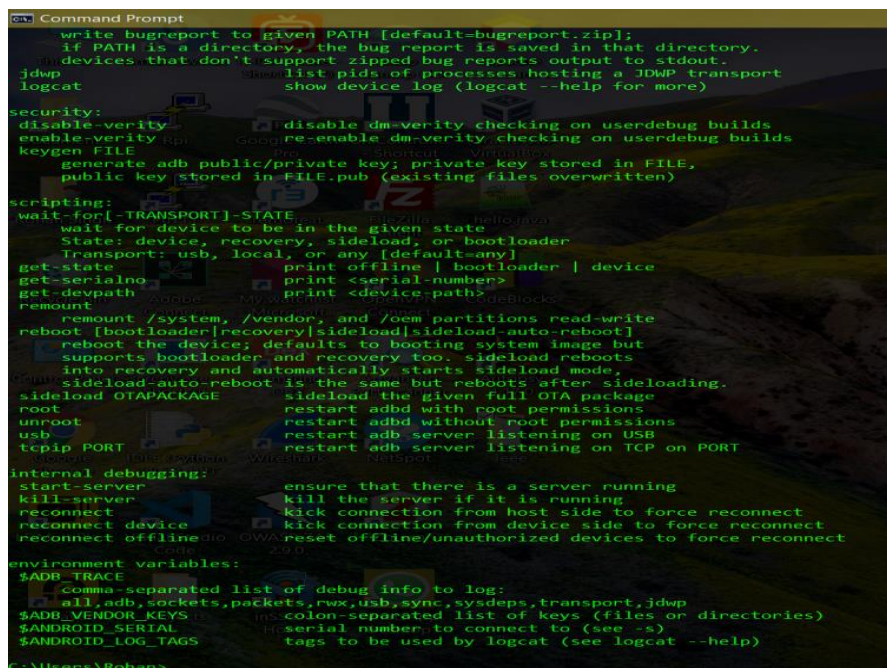
Daniel E. Krutz and Nuthan Munaiah (from 4th International Conference on Mobile Software Engineering and Systems) in his research showed that some applications were using certain permissions which were not required in their basic functionality and deactivating those permissions resulted in frequent app crashes (Krutz *et al.*, 2017).

## 2.7 Gap Analysis

A huge gap between the solutions provided and the actual transmission of data in the background can be clearly seen. Also, there are various solutions to limit the background data transmission but because of less knowledge about the importance of personal data of a user and more complexity in the Android ecosystem it is never practically implemented.

## 2.8 Proposed Solution

Some of the proposed methods can effectively reduce the background data transmission. Implementing each methods has some advantages as well as some minor disadvantages. But the most appropriate solution would be to use ADB (Android Debug Bridge) command tools('How to Install ADB on Windows, macOS, and Linux', 2021). These ADB command tools can modify any system application without the root access.



```
Command Prompt
write bugreport to given PATH [default=bugreport.zip];
if PATH is a directory, the bug report is saved in that directory.
devices that don't support zipped bug reports output to stdout.
jdwp          list pids of processes hosting a JDWP transport
logcat       show device log (logcat --help for more)

security:
disable-verity      disable dm-verity checking on userdebug builds
enable-verity      re-enable dm-verity checking on userdebug builds
keygen FILE        generate adb public/private key; private key stored in FILE,
                  public key stored in FILE.pub (existing files overwritten)

scripting:
wait-for[-TRANSPORT]-STATE
wait for device to be in the given state
State: device, recovery, sideload, or bootloader
Transport: usb, local, or any [default=any]
get-state          print offline | bootloader | device
get-serialno      print <serial-number>
get-devpath       print <device-path>
remount           remount /system, /vendor, and /oem partitions read-write
reboot [bootloader|recovery|sideload|sideload-auto-reboot]
reboot the device; defaults to booting system image but
supports bootloader and recovery too. sideload reboots
into recovery and automatically starts sideload mode.
sideload-auto-reboot is the same but reboots after sideloading.
sideload OTAPACKAGE
sideload the given full OTA package
unroot           restart adbd with root permissions
usb             restart adbd without root permissions
tcpip PORT      restart adb server listening on TCP on PORT

internal debugging:
start-server     ensure that there is a server running
kill-server      kill the server if it is running
reconnect        kick connection from host side to force reconnect
reconnect device kick connection from device side to force reconnect
reconnect offline reset offline/unauthorized devices to force reconnect

environment variables:
$ADB_TRACE       Comma-separated list of debug info to log:
                all,adb,sockets,packets,rwx,usb,sysdeps,transport,jdwp
$ADB_VENDOR_KEYS colon-separated list of keys (files or directories)
$ANDROID_SERIAL serial number to connect to (see 's')
$ANDROID_LOG_TAGS tags to be used by logcat (see logcat --help)

C:\Users\Rohan>
```

Figure 7: User Interface of ADB Command Tools

But to access ADB command tools the Android device needs to be connected to a laptop and it is based on CLI (command line interface) and hence lacks GUI (graphical user interface). if some of the basic ADB commands converted into a batch file but basic options then any normal user can get their work done easily.

## 3 Research Methodology

Since lot of applications comes with preinstalled applications hence it is difficult to determine how many applications are responsible in transmitting the data at the background. So, research

was conducted on one of the applications to find out how the data is transmitted (hence we selected browser.apk). Using ADB command tools all 3<sup>rd</sup> party cookie sessions were blocked that were responsible for sending data to a specified server and since the broadcasting channel was unknown hence all the channels were blocked (it made the application a bit unusable to those websites where third party cookies were needed). Using Wireshark a lot of IP address were recorded and but focused on which IP address were re attempting to communicate with the server and this experiment was conducted for next five days. In this experiment some values were assigned as 0 and 1 whereas 1 = data transmission was attempted and 0 = data transmission was not attempted. combining both this data a logistic linear regression was drafted to find out the actual output.

The data set was created for those application which were trying to create a communication with the server and hence the applications are the total number of sample size (which is 35). More data set can also be generated by looking at the source code with the help of reverse engineering with the help of tools like APKtool, DEX2 or Simplify. APKtool is selected because it is easy to understand and gives sufficient information. all essential information like other permission needed by the application and manifest files what extracted and saved into CSV file format.

The complete data set was divided into two part in which first 70% were used as training the data and the rest 30% were used in testing those data so that an accurate result can be drafted.

**Tests of Normality**

	Kolmogorov-Smirnov <sup>a</sup>			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
charges	.188	1338	.000	.815	1338	.000

a. Lilliefors Significance Correction

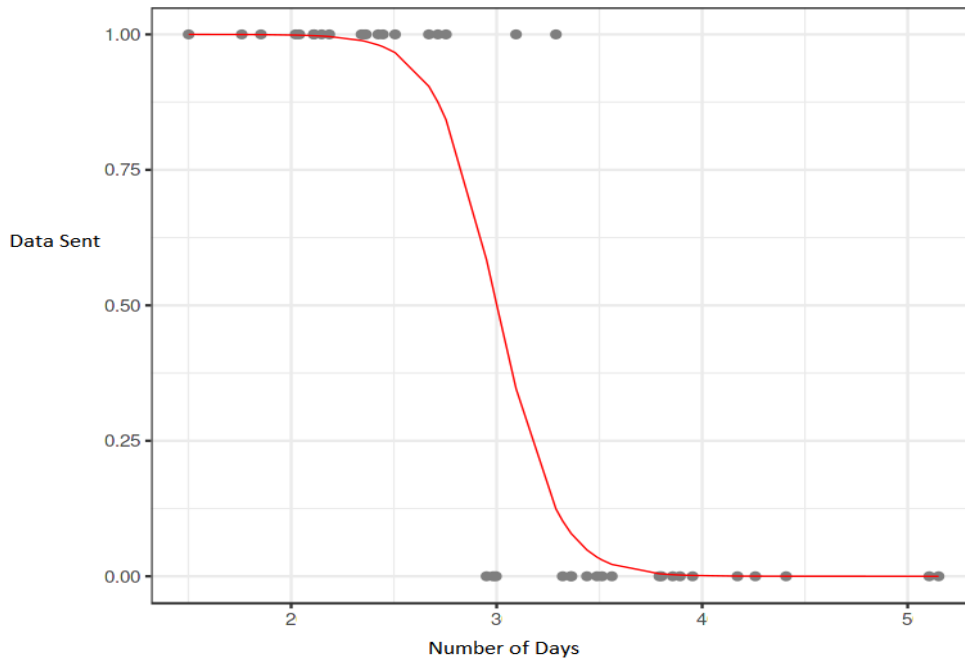
**Figure 8: Test of Normality**

Null hypothesis (h0): Data are drawn from a normal distribution.

Alternate hypothesis (h1): Data are not drawn from a normal distribution.

For this test significance level alpha is 0.05.

since Alpha value is less than 0.05 is obtained a conclusion can be made that the data are drawn from a normal distribution And binary laustic graph was created.



**Figure 9: Logistic Regression Model Result**

From the graph we can see that the following few days the applications tried to send the data in the background.

## 4 Design Specification

To conduct the experiment some of the hardware were used. A laptop (running windows 11), memory 8GB ram, processor Intel i5 (8th generation) But the minimum storage of 5 GB were used. Also a smartphone running Android ecosystem (Running Android 6.0 Marshmallow), with the least modifications done by the OEM is the most preferred device.

In this experiment instead of Android as a hardware the emulator is used named LD player (running Android 6.0 marshmallow). This emulator was assigned only two cores and the emulated device model was named as Samsung SN-M508A, IMEI was assigned as 815606224189658 and network connection was bridged.

Some of the soft trips were used to conduct this experiment were-

- APKtools- for reverse engineering the system application so source code is obtained.
- ADB command tools- these are the list of commands which can be implemented to modify the system application without gaining the root access.
- Command prompt- used to send ADB commands from the laptop to the Android hardware.
- Dev-Ops- A permission monitoring tool built in the Android core.
- SPSS- to perform a binary logistic regression.
- Android studio 3.0- to modified the Android source code.
- Notepad++ - to edit the code.

## 4.1 Enabling Disable Option

This method includes to edit the source code so that the option to disable the system application can be activated.

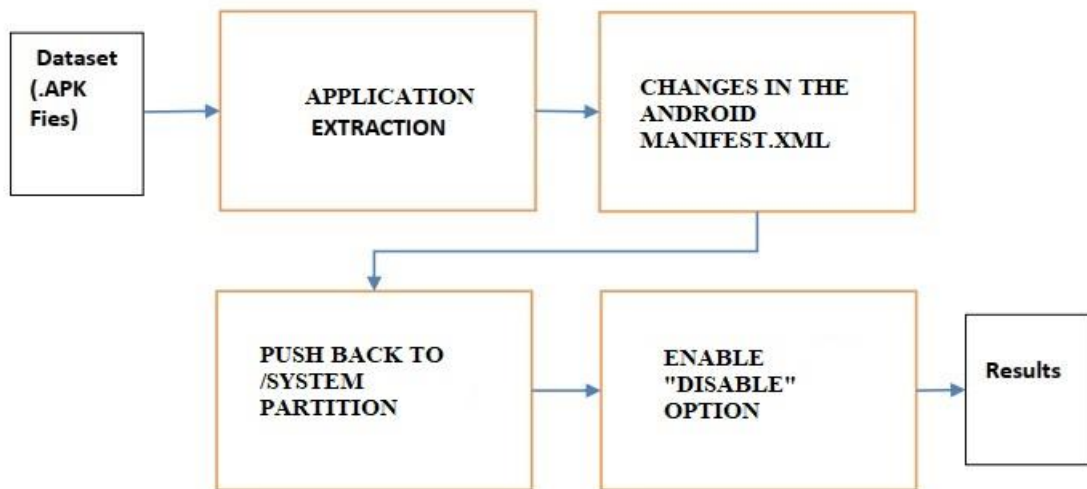


Figure 10: Disable Option Explained by Block Diagram

Any application can be identified using the system analyzer tool to figure out whether the application is sending data in the background. Each system application can be extracted using ADB command “adb pull /system/app/package.name.apk”. Once the application is extracted then using apktool.jar We look at the code under “AndroidManifest.xml” (Park, Chun and Jung, 2018). Those system application where disable option is not enabled will contain the code as shown `<category android:name="android.intent.category.DEFAULT"/>`. Then the code is changed into `<category android:name="android.intent.category.USER"/>` and re-compiled into apk package. The new modified application is pushed back into the /system partition. Thus an option to disabled system application is enabled.

## 4.2 Using Permission Manager

Enabling the disabled option is the best method to stop any background data transmission but it is not too effective because if the application is Disabled then it cannot be used by the user. Moreover, this method needs to change the source code of each and every application which is very time consuming. So, it is only useful if that system application is not at all needed by the user. But if the user needs that system application, then the alternative method would be “Permission Manager”. It is Android inbuilt feature which was introduced in Android 6.0 marshmallow version. This method is simple to use, the user has to simply turn off those permissions of broadcasting channel which is misusing the permission in sending the data in the background.

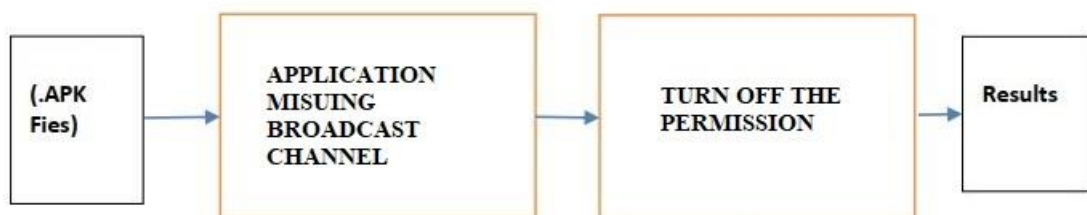


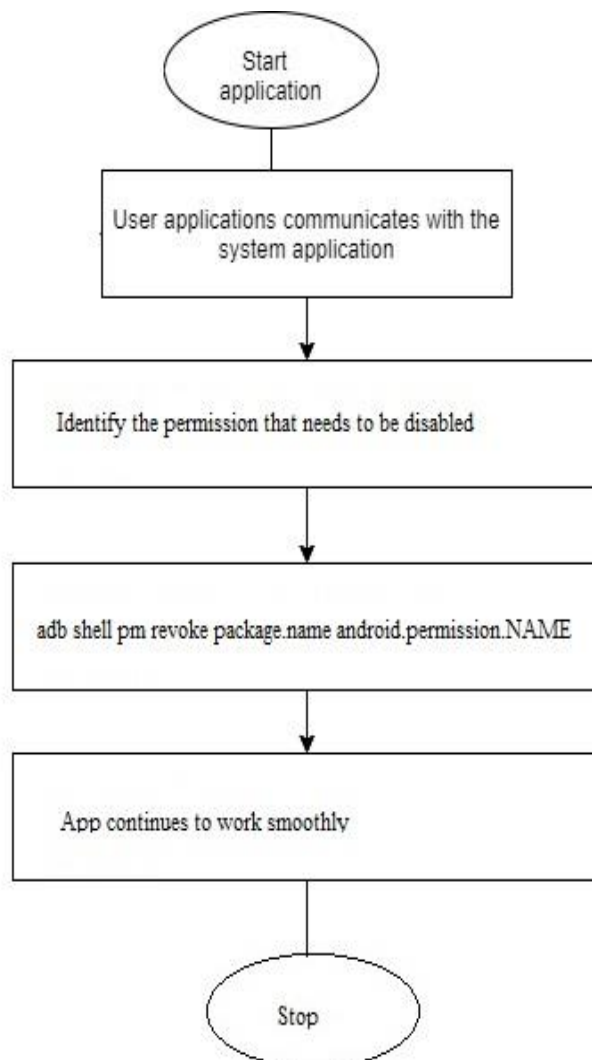
Figure 11: Permission Manager Explained by Block Diagram

It looks like a simple solution but turning some of the broadcasting channels were making the application crash more often. but the same functionality can be done with the help of Dev-Ops as those applications were substituted with the fake permissions(Sellwood and Crampton, 2013). Hence this method resulted in the less crashing of applications.

Again, this method is only applicable on those system applications where the authentication token is not required. OEM manufacturers ensure that applications are constantly verified with the token mechanism so that the data which they are receiving Are not corrupted and continuously communicating to their server.

### 4.3 Using ADB Commands

This method can be termed as the most safest method to modify any system application because it does not require any root permission and also does not causes frequent app crashes.



**Figure 12: Functionality of ADB Command Tools Explained by Flowchart**

This process requires to execute a set of commands from a laptop Running any operating system (in this case it is windows 11). The android device is connected to the Windows PC via USB cable, Then ADB driver needs to be installed (this can be obtained from the manufacturer's website because it is hardware specific). Once the device is connected then we open a Command Prompt and try to locate the device By using “adb devices” command. After locating the device another command is pushed to revoke the list of permissions using “adb shell pm revoke package.name android.permission.NAME” command which is pushed from Command Prompt.

Lots of possibilities can be done with the help of ADB command tools because these are into android core ecosystem because Android was designed to be an open-source platform for some modifications can be done without gaining a root privilege.

Some of the major challenges faced by ADB command tools are all these codes are CLI (command line interface) hence it does not contain GUI (graphical user interface). Because of non-presence of GUI, it becomes extremely difficult for a normal user to execute these commands due to less knowledge of coding. If some of the basic commands are converted into batch files, then those basic commands can easily be executed by any normal user.

## 5 Implementation

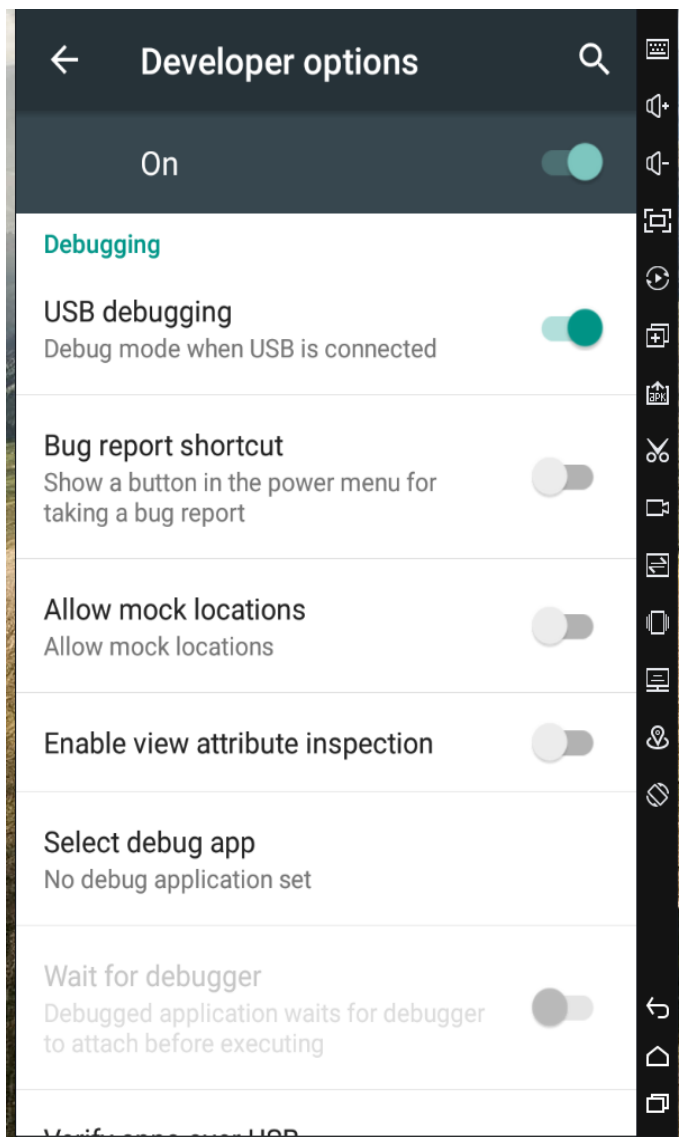
For in-depth understanding of implementation, the basics of android partition architecture need to be understood. In any Android device the storage partition is divided into multiple different partitions and they are named as /data, /system, /cache, /vendor, and many more. Apart from all these partition names two partition which are primary to any Android devices are /system and /data partition.

All those applications which are downloaded by the user from the play store are transferred to /data partition memory allocation. This memory allocation has all the permissions like read, right, modify and execute privileges. Because of this reason any manufacturer does not prefer to push their adware application because if the user get aware of the applications it can easily uninstall them(Lu *et al.*, 2013). Whereas, /system partition contains only read and execute privileges and user does not have any control over this memory allocation. Less privilege rights are given to this partition because of the presence of main operating system. It contains all pre-installed system application like calculator, calendar, clock, gallery, music player, contacts and many more. Along with this pre-installed application it also contains manufacturer adware applications like battery meter, files cleaner, browser and many more. For any type of modification in this partition root permissions are required.

To begin with the process first we will run a “system analyzer tool” which helps in highlighting those application which are sending data at the background and using those broadcasting channels which are not the main functionality of the applications. Then from the list of obtained application we then find out how many applications are user installed because user installed applications can be easily uninstalled. Then the remaining list will contains only the system applications. Now we will see the application properties of each of these system applications and if the system application disable option is present we can simply disable the application. But if the disable option is not present then we will try to extract the application from /system partition, make changes to AndroidManifest.xml and push back to the same /system partition and then we can disable the system application.

Another less popular method includes the use of inbuilt feature “Permission Manager”. This method is effective on both user as well as system application. The main advantage of using permission manager is that the user can still use the application because the broadcasting channel which was used to send the data at the background was only disabled rather than the whole application. The user has to go to the settings of a device and select the permission manager option. Then the list of all the applications present in the device are displayed. Clicking on each application will show all the broadcasting channel with the privilege rights and the user can simply turn off those channels which are not needed by the applications. But this method may cause frequent app crashing and can make the application unstable to use. The safest method to eliminate the data transmission in the background is the use of ADB commands. Certain steps are needed to ensure that the ADB commands are properly pushed from the laptop to the Android device because non proper execution may result in the bricking of device And may cause the device to malfunction in long term. Hence we will focus on all the basic steps that are needed to push the commands. These steps are as follows:-

**Step 1-** firstly we have to make sure that USB debugging option is enabled in the android device. If it is not enabled then go to settings, then enable developer options.



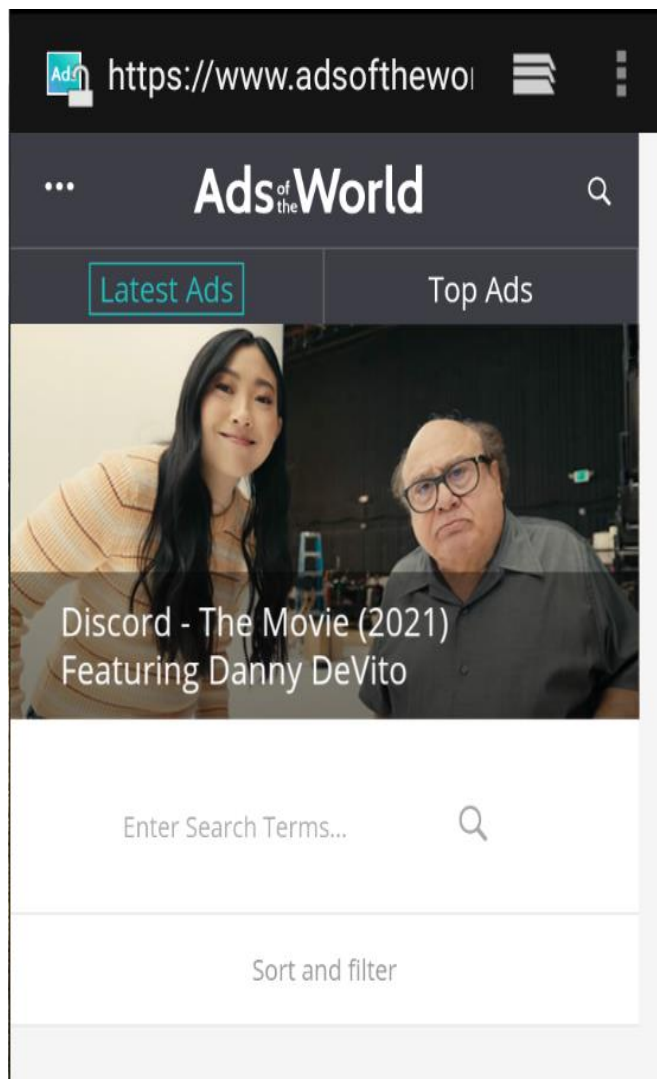
**Figure 13: USB debugging Option Enabled Under Settings Menu**

**Step 2-** Under developer options enable USB debugging. And also ensure that select debug app no debugging application is selected from the beginning.

**Step 3-** A laptop is required running any operating system (in this case windows 11 is used) Needs to be attached with the Android device via USB cable (in this case Android emulator is used).

**Step 4-** Send 'adb devices' command to ensure that the devices is interacting with the PC and when it shows the device ID in return it means that the secure connection is established between the PC and the Android device.

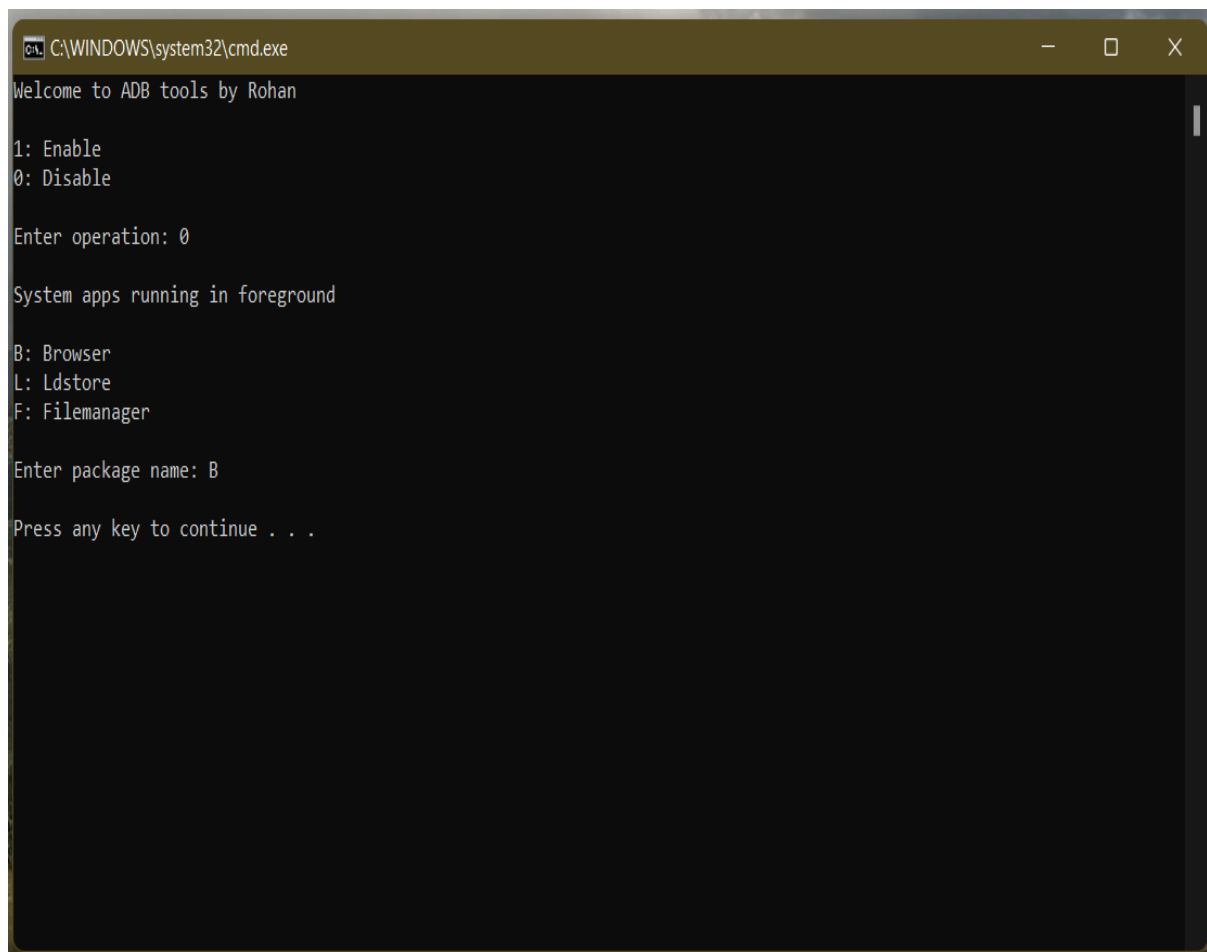
**Step 5-** Now we will select one of the system applications (in this case we selected browser.apk) and we navigated to one of the website which is loaded heavily by the advertisement (in this case we navigated to <https://www.adsoftheworld.com/>) and we can observe that the advertisement which is displaying is pulling the data from the cookies saved in the browser section.



**Figure 14: Enabled Cookies Resulted in Displaying of Ads**



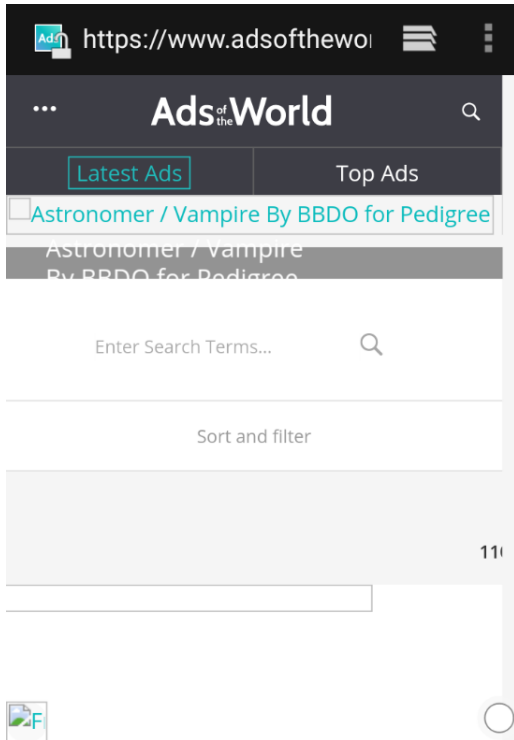
**Step 6-** Once everything is loaded, we will execute ADB tool.bat file. it contains all the basic commands in the form of batch file which will help in disabling the broadcasting channel. When initially launched it will ask from the user to select either 1 or 0. Here, 0 = disable all broadcasting channel and 1 = enable all broadcasting channel. This application will only highlight the application which is running in the foreground because all the applications including the background running are selected then the list will be very huge. And some of the applications are essential to run in order to maintain the smooth functioning of Android environment.



```
C:\WINDOWS\system32\cmd.exe
Welcome to ADB tools by Rohan
1: Enable
0: Disable
Enter operation: 0
System apps running in foreground
B: Browser
L: Ldstore
F: Filemanager
Enter package name: B
Press any key to continue . . .
```

**Figure 15: ADB Tool Displaying Options**

**Step 7-** Next we select option B for browser application from the available list. As soon as option B is selected it turns of all the background broadcasting channels and now we go to the same website we can see that the advertisements are not able to load because the cookie session has been blocked.



**Figure 16: Disabled Cookies Resulted in No Display of Ads**

**Step 8-** Hence we were successful in preventing the transmission of personal data in the background by following these steps.

## 6 Evaluation

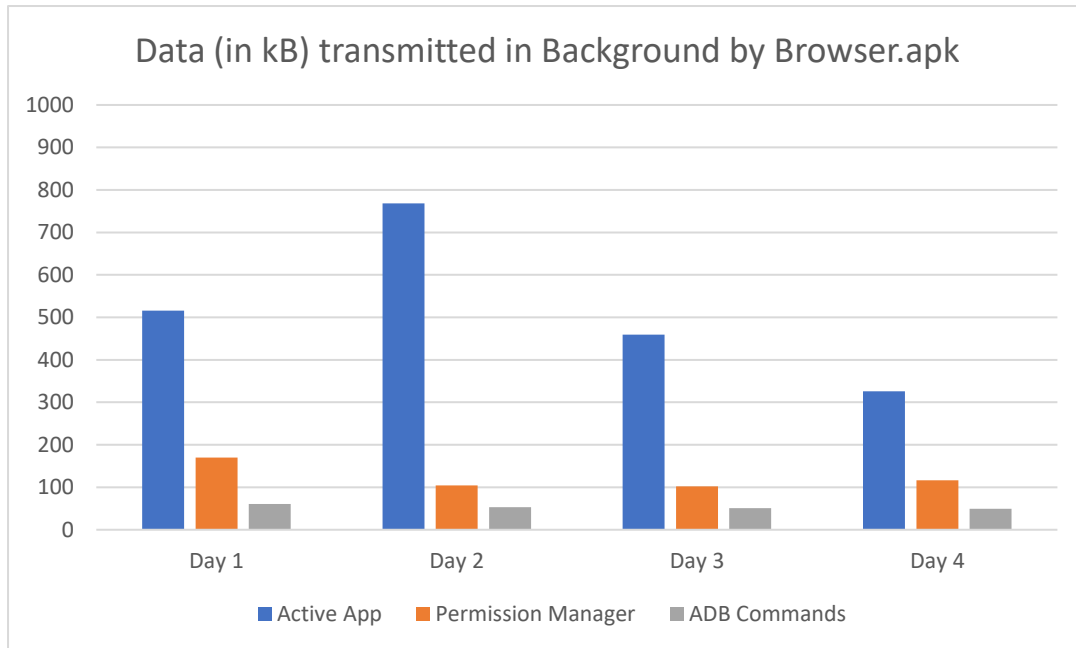
Under this section we will compare the total data leak (in kB) from two system applications by using all these methods.

### 6.1 Case Study 1

Here we selected browser.apk as an example its details are as follows: -

**Table 1**

Application Name	Brower.apk
Version	7.1.2
Package Name	me.android.browser
System Application	Yes



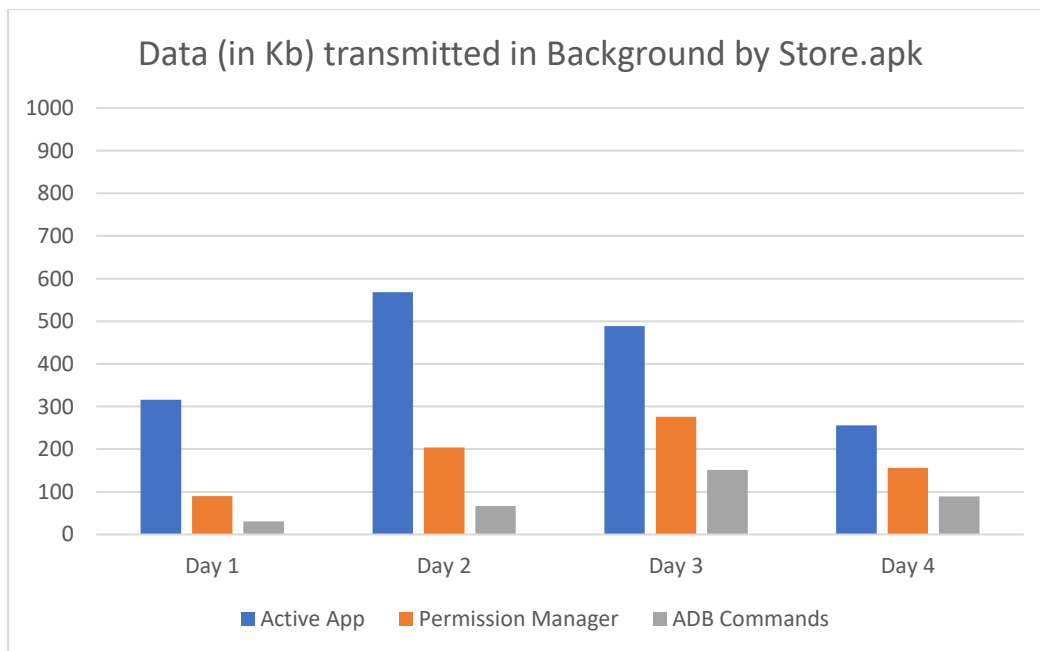
**Figure 17: Data Transmission by Browser.apk**

## 6.2 Case Study 2

Here we selected store.apk as an example its details are as follows: -

**Table 2**

Application Name	Store.apk
Version	5.1.2
Package Name	me.android.store
System Application	Yes



**Figure 18: Data Transmission by Store.apk**

From case 1 and case 2 we can observe there is a huge difference in the Total volume of data transmitted. this evaluation does not contain the “disable” method because this method doesn't allow the application to run hence the background data transmission will always be 0 As the main objective of this research is to prevent the transmission of data in the background without turning off the application.

### **6.3 Discussion**

After evaluating the case 1 and the case 2 we can observe that the transmission of data in the background was not fully prevented. In fact, after blocking all the broadcasting channels still some data were able to transmit. But the experiment was successful in preventing almost 80% transmission of data. theoretically it seems like the privacy of data can be fully implemented but practically a lot of work needs to be done from Google itself. For example- Google needs to ensure that certain guidelines are drafted for the manufacturers and must be forcefully implemented like if the manufacturer claimed certain apps as system applications then it needs to follow the limit of data transmitted in the background. If the data contains too much of privacy then it must give explanations in details that why those data are needed. Also any system apps can be disabled by the user because those applications causes the greater security threat and increases the chances of malware attacks.

Some more changes like Privacy Manager should be given more user functionality like if a certain broadcasting channel is turned off by the user the application should still function smoothly (that means not causing the application to crash frequently). Also the free applications under the “Play Store” should give the greater details about the type of data that will transmit in the background So that the user will be well aware about their privacy.

Fully blocking the data transmission in the background is not possible because some data are needed by the free applications to function. therefore, the ADB command tools need to be optimized further so that it can identify based on the data type that it should be transmitted or prevented.

## **7 Conclusion and Future Work**

Therefore, we can conclude that Android ecosystem has the great flexibility (for this reason it has captured the maximum percentage of mobile OS market share of 2021) it can be made more secure if the mentioned solutions are implemented. Also, ADB tool.bat is based on Android 6.0 that means any Android version which is greater than Android 6.0 these commands may not give accurate result. Any changes to the Android SDK will break the functionality of these codes hence the codes need to be updated whenever the Android SDK is upgraded. So the objective is to make this ADB tool.bat to contain more Android versions and based on the Android version certain codes will be auto implemented. Hope this research will make the Android ecosystem more secure and give user more privacy control.

## References

- Almuhimedi, H. *et al.* (2015) ‘Your Location has been Shared 5,398 Times! A Field Study on Mobile App Privacy Nudging’, in. doi:10.1145/2702123.2702210.
- Altshuler, Y. *et al.* (2012) ‘Incremental Learning with Accuracy Prediction of Social and Individual Properties from Mobile-Phone Data’, in *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Conference on Social Computing. 2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Conference on Social Computing*, pp. 969–974. doi:10.1109/SocialCom-PASSAT.2012.102.
- Boueiz, M.-R. (2020) ‘Importance of rooting in an Android data acquisition’, in *2020 8th International Symposium on Digital Forensics and Security (ISDFS). 2020 8th International Symposium on Digital Forensics and Security (ISDFS)*, pp. 1–4. doi:10.1109/ISDFS49300.2020.9116445.
- Enghouse (2016) *Set up Internet, Set up Internet - Samsung Galaxy S2*. Available at: <https://www.helpforsmartphone.com/public/en/samsung/galaxy-s2/android-4-1/guides/22/Set%20up%20Internet> (Accessed: 8 November 2021).
- Guo, Y. *et al.* (2016) ‘The static detection analysis technology of Android source codes’, in *2016 IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC). 2016 IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC)*, pp. 288–292. doi:10.1109/ICNIDC.2016.7974582.
- Han, Z. *et al.* (2014) *Systematic Analysis and Detection of Misconfiguration Vulnerabilities in Android Smartphones*. doi:10.13140/2.1.2383.2009.
- ‘How to Install ADB on Windows, macOS, and Linux’ (2021) *xda-developers*, 28 July. Available at: <https://www.xda-developers.com/install-adb-windows-macos-linux/> (Accessed: 14 August 2021).
- Khokhlov, I. and Reznik, L. (2018) ‘Android system security evaluation’, in *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC). 2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pp. 1–2. doi:10.1109/CCNC.2018.8319325.
- Krutz, D.E. *et al.* (2017) ‘Who Added That Permission to My App? An Analysis of Developer Permission Changes in Open Source Android Apps’, in *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft). 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, pp. 165–169. doi:10.1109/MOBILESoft.2017.5.
- Lu, L. *et al.* (2013) ‘A study of Linux file system evolution’, in *Proceedings of the 11th USENIX conference on File and Storage Technologies*. USA: USENIX Association (FAST’13), pp. 31–44.
- Ma, Y. *et al.* (2018) ‘A Tale of Two Fashions: An Empirical Study on the Performance of Native Apps and Web Apps on Android’, *IEEE Transactions on Mobile Computing*, 17(5), pp. 990–1003. doi:10.1109/TMC.2017.2756633.

*Mobile OS market share 2021* (2021) *Statista*. Available at: <https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/> (Accessed: 17 April 2021).

Park, J., Chun, H. and Jung, S. (2018) ‘API and permission-based classification system for Android malware analysis’, in *2018 International Conference on Information Networking (ICOIN)*. *2018 International Conference on Information Networking (ICOIN)*, pp. 930–935. doi:10.1109/ICOIN.2018.8343260.

Sarkar, A. *et al.* (2019) ‘Android Application Development: A Brief Overview of Android Platforms and Evolution of Security Systems’, in *2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*. *2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pp. 73–79. doi:10.1109/I-SMAC47947.2019.9032440.

Sellwood, J. and Crampton, J. (2013) ‘Sleeping android: the danger of dormant permissions’, in *Proceedings of the Third ACM workshop on Security and privacy in smartphones & mobile devices*. New York, NY, USA: Association for Computing Machinery (SPSM ’13), pp. 55–66. doi:10.1145/2516760.2516774.

Shan, Z., Neamtiu, I. and Samuel, R. (2018) ‘Self-Hiding Behavior in Android Apps: Detection and Characterization’, in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pp. 728–739. doi:10.1145/3180155.3180214.

Thomas H. Davenport and Jill Dyché (2013) *Big Data in Big Companies*. Available at: <https://www.iqpc.com/media/7863/11710.pdf> (Accessed: 27 October 2021).