# An Exploration of Informational Capitalism of freeware apps within Android

William Black, Amirhossein Sefati, Zeinab Erfanmanesh

## 1 Introduction

Nowadays, sensors are an essential part of smartphones, sensors are the 'smart'. They can be used in several different areas like taking pictures with a camera sensor, and locating and finding the best routes using radio signal receivers. The list of sensors is ever-expanding including the likes of an accelerometer, gyroscope, thermometer, and barometer. More sensors in your phone collect more data about you and your environment. Information gathered by mobile sensors has become priceless since they can be used in many different ways such as helping you when you are in danger after you have fallen off a cliff face. Also, vital for scenarios like advertising, marketing, or stealing your sensitive information.

Today's computing is primarily driven by mobile phones, with 1.5 billion devices sold in 2021 [6]. With this shift to smartphones, we are presented with an ever-expanding surface for exploitation of the users. A useful feature of a smartphone is its versatility allowing for any application to extend the base features. Applications have elevated access to the user, including user interaction, stored data, and sensors. Each poses a risk to the user's privacy and security. This is why there are numerous checks and balances to help protect users of mobile phones. Principally the 'App' stores Google Play and App store for android and iPhone Operating System (iOS) respectively. These repositories aim to give their users a safe place to source applications for their smartphones.

Even though software repositories provide an improvement over sourcing software from a place on the Internet. Malicious applications are distant to make it through the security gates that are present to be added to one of the 'App' stores. More problematic than the blatantly evil applications, are the apps that are incentivized to profit off the user indirectly. This is the free application model where apps provide their services in exchange for ads and all of the interactions that the user has with the application's service. The canonical example of a company employing this model is Facebook now named Meta, a company able to make some profit without charging its users.

The mobile phone's computing devices have all this rich sensor data, available to applications on that mobile phone. Now likely that these applications are incentivized to compromise this trust to get compensation for their service.

## 2 Prior Work

We will first take this time to discuss sensors and some vulnerabilities within. These attacks fall into two categories the first of which is the classification type, these attacks aim to infer the user's actions. Location tracking is the second concern, by using a collection of sensors to identify where the user frequents. We will all discuss some background on the Android permission system, its functionality, and its limitations.

### 2.1 Sensors Data

Some sophisticated attacks can be run on users of smartphones. First, we will talk about the pioneering work by Cai et. al, and Aviv et. al [7-8]. These attacks use machine learning and deep learning techniques to classify users' touch input on sensitive views, including payment and logins. It is hoped that these views are protected from possible malicious applications. The construction of these attacks is not within the scope of this review. The surface of this attack is what is most concerning this side channel that is ever-present and can infer security pins and possibly more sensitive data.

The two attacks above use only the gyroscope and accelerometer, and from our study of 100 popular android applications 98% and 100% access this data from the operating system. Asking the question why is this data used so freely by applications in the android system? These pose some useful User Interface (UI) benefits, such as screen rotation from portrait to landscape, or simulating a lightsaber (an old application I once had). It is clear to see that even in this simple example of basic sensors inside smartphones there is both a threat to user privacy and help for users, with better UI and more rich application environments.

Android has built-in support for well over 15 sensors, the camera, microphone, location, and activity sensors have start time permissions to allow an application to use these sensors. Each of these sensors possibly has data that will have the potential to leak sensitive data about the user. We believe that this data that is collected to make the user experience better poses issues.

### 2.2 Android Permission System

The permission system of the Android operating system is the heart of its security and privacy. Since Android API updates regularly each year, there are many kinds of research

into different API versions and their permission system. The authors of [1] have performed an early investigation on the permissions system to determine how the permissions have grown. The study covers API levels 3 to 15 on a set of 237 Android applications. They concluded that the permissions system increased in complexity for both the Android platform and its applications until 2012.

Following that, in newer Android versions the permission system added another feature called "run-time" permission. It is about asking for the granting of permission the first time the application is using that operation requiring privileged access. The authors of [2] investigated the evolution of the permissions system until the Android platform version 23 in 2015. The rising consideration was about accessing sensitive information by privileged third-party applications without user awareness, which drove them to investigate run-time permissions deeply. They discovered various security concerns regarding the complexity of the run-time Android permissions system that the community has to address. [3]

The authors of [4] investigated the security issues of the Android permission system for API 23 and higher. They concluded an attack called Transformation Attacks which targets other applications installed on a phone.

Our view of the Android permission system is that the access is given to the application based on some factors, what the application's features require, and the user's consent for permissions that are considered dangerous. The usage of this sensor data is not regulated by the operating system, if an application can access sensors the actual usage of this data is not governed. This concept takes the forum in many ways, a one-time granting of microphone permission to Spotify will allow Spotify to access the microphone indefinitely when even this app is executing. This allows Spotify to possibly record arbitrary data from the microphone. This is a technical ability, not a legal ability or an insinuation that Spotify records you without your knowledge.

## 3 Android Sensors

The sensor subsystem within AOSP works by providing a hardware abstraction layer interface that must be achieved for a sensor to be functional within android. Most interfaces are straightforward, supporting polling and flushing sensor data to a memory address. Other interfaces are more contention for example, in flagship smartphones multi-camera is a novel sales feature. Where these cameras are more contention is should there be a more complex camera interface that can support added sensors, or should each camera sensor have its unique interface. We decided that looking at the raw data entering the operating system at these interfaces would be too cumbersome to collect. We took the opposite approach to collect data on sensor data being used within the Android ecosystem by logging the sensor data being passed into an application. In this section, we briefly discuss what changes we did to collect the logging data.

### 3.1 General Debugging

The first change that we did within AOSP is enabled verbose debugging in the sensor hardware layer. This debugging data was used to identify when sensor components were being accessed by the kernel when we begin testing applications. Adding debugging data to the HAL interfaces was the full extent of our instrumentation to the kernel level sensor access. This is required to identify some sensor access as some applications choose to develop application callback and sensor access for a variety of reasons, performance, battery, and control.

### 3.2 Sensor Event

Most sensors within android get their sensor data to a process or application by launching that application's callback with a sensor event object. Within this object are a timestamp, sensorId, and numerical array of data. This Sensor Event Is a class with one constructor, so It could be assumed that some code will call the contractor of this class to instantiate this object, but no this is android you allocate raw memory cast it to a pointer to a sensor object, and memcpy your sensor data into this memory so it is magically formed. We do not guess why this is done like this. This occurs in the systemSensorManager class, defined as a sensorEventDispater method. We logged the PID in which the callback was being executed along with the sensorId info and the raw data into the logcat system.

A sensor event will cover the data for the following sensors; Accelerometer, Gyroscope, Proximity, Light, Steps, rotation, gravity, etc. This list is not exhaustive but gives the theme of the sensors that employ this method of data delivery.

### 3.3 Microphone and Camera

Most applications that want to access this data, require some encoding to process this data. Whether photos are presented to a webView as a portable network graphic (PNG), or some other codec. What is similar about the camera and microphone is that the audio or visual data must be encoded before being presented to a user. A developer can stream the raw data into their App and use libraries to encode this data into something usable, for example, the default 'camera app' provided by the system vendor does exactly this. The next option that is used for access to the camera or microphone is to have your application call the default as a sub-process. Both of these methods we only can see due to our debugging data from the HAL.

Our method of logging this rich data is to instrument the mediaRecorder module provided by the AOSP, this is the recommended method of accessing these sensors allowing for a start/stop recording interface. We log the process and the Linux file descriptor and filename, if available, to where the sensor data will be written to. We do not log the actual data being accessed, we use this as evidence that data was collected.

### 3.4 Location and Touch

We did not consider Touch as data we were collecting in this survey of the AOSP. Furthermore, the location was also ignored for two main reasons, the first of which is the control of this data the operating system has logged all the locations this data could be accessed proved infeasible. The second was the plethora of articles looking into the practices of managing this data.

## 4 Testing Phase

In this section, we tried to test our edited version of Android in the hope to find some malicious freeware apps. First, we found considered 7 categories as follows:

- Health and Fitness: Applications in this category are often free but with in-purchase options. The popularity of this category has increased in the past year due to COVID19 forcing people to be at home and closing fitness clubs. So, this category can be a good target for attackers and fraudulent publishers.
- Maps and Navigation: In an era of Google Maps, other applications have no chance to overcome the market. But, we observed that there are plenty of applications providing the same service like Google Maps. They are free to use and many of them even don't have an in-purchase option. So, we collect applications of this category to test since there is a good chance to have something suspicious about them.
- Personalising: There is some funny application in Google Play with a huge number of downloads/installs. They are theme changers for everything. In our project, we collected some applications which change the theme of the android phone when the battery is charging. Another type was the application that change the keyboard layout and make it different than the default one. There is a good chance that the users of these applications are not professionals, they are probably children or teenagers. So, with a high chance, they don't know a lot about permissions, security, and privacy in Android. Therefore, they will grant any permissions that the application needs. So, they can also be a good target for fraudulent publishers.
- Remote Controls: This category is targeting people who don't know about hardware limitations. They advertise "Turn on your TV with your phone no matter what is their brand!" which is not possible knowing there should be many different hardware specifications to be able to do that.
- QR Code Scanners: The camera of an Android phone can scan QR codes itself. Knowing this fact, there is a surprisingly high volume of applications in Google play that provide QR code scanning services with a high number of downloads/installs. They even don't provide any other services. So, this was suspicious for us and we include them in our testing targets.
- Speech-to-Text: There are numerous applications with the same goal, they convert voice to text. Although we tested some of them manually and they did their job done but there is a good chance that these applications are recording your voice all the time. So, we add them to our database.
- Equalizer and Hearing Aid: Applications in this category work with the sound output of Android phones in a way that is not doable in a normal way with default Android applications. Some of them make the volume of the phone louder than the normal highest possible. That is why they called this feature of their apps "Hearing Aid". However, we tested some of these applications manually and they couldn't work as well as they claim.

We picked 15 applications from each of the abovementioned categories. The way that we picked them was random, taking into consideration the number of downloads and installs of that application. The installs should be higher than average so we won't end up testing some applications with really low usage. Also, if there is a high number of downloads/installs while the app is not very useful, it is more suspicious to be fraudulent. Finally, we created a file with 105 lines each line including the package name of the application. We got the package name of the applications from their URL of them when we tried to open their install page on the Google Play website.

### 4.1 Downloading Desired Applications

Since we tested the applications on our edited version of Android, we should use APK files alongside Android Debug Bridge (ADB). So, we need to have the APK files to install them in our target phone which was a rooted Google Pixel 3a. Throughout the project, we tried to find an automation tool to download all the desired APK files from the package name file we created in the previous section. But we were not successful to find a proper working code for automation. So, we end up downloading all the applications manually. Table 1 shows the package names of applications that we look for their APK files in this project.

Finally, we end up downloading 101 applications from the 105 package names. We couldn't find the other 4 applications from our source. The source to download the APK file from the package name was an extension for chrome which is called "APK-Downloader" which is developed by "apkcombo.com". It was easy to use and also it was completely free. However, it has an APK file for 101 applications in its database and it was enough for us to test our edited version of Android.

### 4.2 Implementing Automation Process

Since it was impossible to test 101 applications manually and one by one, we decided to write a script to do it for us. We use the following tools to help our script run:

- Android Debug Bridge: it was used for installing/uninstalling the app, running the test process, getting process id, getting task id, pinning/unpinning the app to the screen, and most important, capturing the logs.
- Monkey Test Tool: it is integrated with ADB, we use this to generate 3000 random calls to the app to capture the logs from app activity.

Table 4 shows what was included in our automation process with an actual code to run that task. First, it installs the app. Then, it gets the package name which is mapped to the number in the file we created in the previous section. Now, it tries to open the app and it will wait for it to finish the opening process. Then, it gets two IDs. First, process id (PID), which helps us to filter the logs which are generated from that specific app. Second, task id (TID). During our investigations on using Monkey, we found a

TABLE 1
Category and package name of desired applications

| Category | Package name |
|---|---|
| Fitness | bodyfast.zero.fastingtracker.weightloss<br>losebellyfat.flatstomach.-.fatburning<br>menloseweight.-.weightlossformen<br>loseweightapp.loseweightappforwomen<br>women.workout.female.fitness<br>loseweight.weightloss.workout.fitness<br>homeworkout.homeworkouts.noequipment<br>pedometer.steptracker.-.stepcounter<br>sixpack.sixpackabs.absworkout<br>armworkout.armworkout.armexercises<br>increaseheightworkout.increaseexercise<br>splits.splitstraining.-.splitsin30days<br>steptracker.healthandfitness.-.pedometer<br>buttocksworkout.-.forwomen.legworkout<br>absworkout.ab.-.fatburningworkout |
| Navigation | com.baladmaps<br>org.rajman.neshan.traffic.-.navigator<br>menloseweight.-.weightlossformen<br>com.joulespersecond.seattlebusbot<br>om.ec.evowner<br>org.eurail.railplanner<br>com.savvy.navvy.android.app<br>com.what3words.android<br>com.salebug.truckstop<br>com.tranzmate<br>com.calimoto.calimoto<br>com.eatsleepride.esrandroid<br>gr.talent.kurviger<br>se.app.detecht<br>com.reverllc.rever |
| QR Code Scanners | qrcodescanner.—.qrscanner.qrcodereader<br>com.google.zxing.client.android<br>com.scanteam.qrcodereader<br>appinventor.ap_progetto2003.SCAN<br>net.qrbot<br>de.gavitec.android<br>la.droid.qr.priva<br>com.ScanQR<br>com.qrscanner.qrreader.qr.-.maximustools<br>com.talukder.qrcodescannerfree<br>com.ScanLife<br>com.application_4u.-.-.flashlight<br>com.ebooks.ebookreader<br>voicedream.reader<br>com.bookfunnel.bookfunnel |
| Speech to Text | com.ktix007.talk<br>com.fahmtechnologies.speechtotext<br>com.cliffweitzman.speechify2<br>com.mrlabs.voicetexter<br>com.bongappstore9.voice_typing<br>org.eurail.railplanner<br>com.texttospeech.-.-.-.tts.audio.converter<br>nl.dtt.voicemail<br>com.langogo.transcribe<br>com.hyperionics.avar<br>hesoft.T2S<br>de.townkult.notizblock<br>com.khymaera.android.listnotefree<br>com.nuance.dragonanywhere<br>com.maruar.voicetotext |

TABLE 2
Cont. Category and package name of desired applications

| Category | Package name |
|---|---|
| Personalising | com.aoemoji.keyboard<br>kika.emoji.keyboard.teclados.clavier<br>com.simejikeyboard<br>com.syntellia.fleksy.keyboard<br>com.jb.emoji.gokeyboard<br>com.smarttechapps.emoji<br>com.battery.charging.animation.Effect<br>com.mobilefastcharger.-.flashing.-.-.chargingapp<br>com.batterycharginganimation.-.-.Batterycharging<br>com.charging.fun<br>com.batterycharging.-.-.animationeffects<br>com.malam.color.flashlight<br>com.rvappstudios.Flash.-.-.-.-.Flashlight<br>call.sms.flash.alert<br>flashalerts.flashlight.calls.messages |
| Equalizer | com.it4you.petralex<br>com.main.amihear<br>com.lifeispurpose.BestHearingAid<br>com.ronasoftstudios.hearmax<br>com.superhearing.earspeaker<br>com.audiofix.hearboost<br>com.superhearing.soundamplifier<br>com.earcare.apps.hearingaid<br>com.oticon.remotecontrol<br>com.everydayapps.-.-.-.volumebooster<br>music.volume.equalizer.-.-.gold_style<br>com.kalay.equalizer<br>bass.eq.music.player.theme<br>sound.volume.volumebooster<br>com.smartandroidapps.equalizer |
| Remote Control | com.fineart.universal.tv.remote.control<br>com.adi.remote.phone<br>smart.tv.wifi.remote.control.samcontrol<br>com.universal.remote.ms.tv<br>codematics.wifitv.-.smarttv.-.tv.-.control<br>sensustech.universal.tv.-.control<br>com.roku.remote<br>com.tekoia.sure.activities<br>sensustech.android.tv.remote.control<br>com.universal.tv.-.-.all.tv.controller<br>com.universal.remote.ms<br>com.quanticapps.remotetvs<br>com.universal.tv.remote.control.irremote<br>com.cetusplay.remotephone<br>com.kraftwerk9.remotie |

way that prevents the monkey tool to use system calls which are called screen pinning. For pinning the screen we needed the TID of the app to pin it to the screen. So, after it pins the app to the screen of our target, it starts logging the output with ADB "logcat" command. Then, it starts sending 3 thousand requests to the app randomly. it stops for the operation to end. After finishing the process, it unpins the app. Finally, it uninstalls the application from our target and will move on to the next APK file.

TABLE 3
Automation Process

| Process Description | Actual Code |
|---|---|
| Install the app | adb install filename.apk |
| Open the app | monkey -p pkgname -v 1 |
| Wait until it opens | wait |
| Get the Process ID (PID) | adb shell pidof packagename |
| Get the Task ID (TID) | adb shell am stack list |
| Pin the app to the screen using tid | adb shell am task lock TID |
| Capture logs | adb logcat |
| Generate 3000 random call to app | monkey –p pkgname -v 3000 |
| Wait until the process is finished | wait |
| Save the log output | $\geq\geq$ *packagename_log.log* |
| Unpin the app | adb shell am task lock stop |
| Uninstall the app | adb uninstall packagename |

We ran the script and it finished the entire process within 2 hours of working. The environment set up for the testing process had the following configuration: Ubuntu version 20.04 LTS, 16 GB RAM, 512 GB SSD PCIe, and MX 150 as

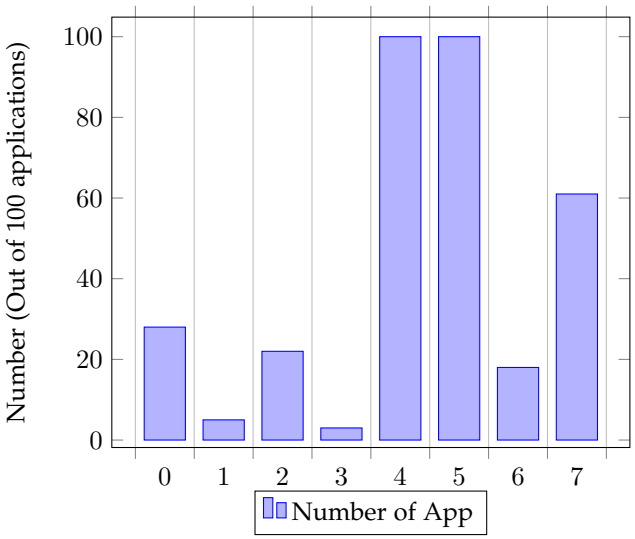


Fig. 1. Number of applications when just the app is captured.

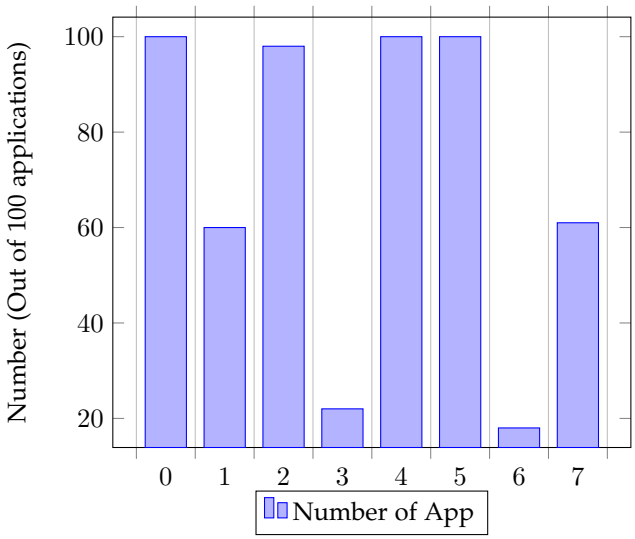


Fig. 2. Number of applications when app and OS are captured.

TABLE 4
Code for each sensor

| # | Name of Sensor |
|---|---|
| 0 | Accelerometer |
| 1 | Magnetometer |
| 2 | Gyroscope |
| 3 | Step |
| 4 | WiFi |
| 5 | Ads |
| 6 | Camera |
| 7 | Microphone |
| 8 | Barometer |
| 9 | Light |
| 10 | Proximity |
| 11 | Gravity |
| 12 | Acceleration |
| 13 | Rotation |

GPU. The script successfully generates log files as we wish. Then we create a program to filter log files and found out what sensors and tools each application used during the test. Note that We used two different approaches for our tests. First, we just captured the logs that were generated by the specific application with a specific package name. In addition, as the second approach, we tests applications while we were capturing all the logs (including the logs from specific applications and the logs from the OS itself. This generates two different logs for each application. Therefore, we processed two different outputs for each application. The next section shows the final result from the logs.

### 4.3 Result of Our Test

We processed all the logs and got the result. Note that 101 applications, one of them, couldn't be installed without having Google play installed. So, we delete that from our results. The application package name was "com.eatsleepride.esrandroid" which prevents the user to use a rooted device, with a basic test. Figure 1 shows how applications were using sensors during the test.

Note that because the number of applications that use barometer, light, proximity, gravity, acceleration, and rotation sensors was lower than 5, we didn't include them in the Figure 1 and Figure 2.

As Figure 1 and Figure 2 show, we have almost all the apps using an accelerometer (100 out of 100) and gyroscope (98 out of 100). It was surprising and we got deeper into the reason that the Android is using these sensors whenever our applications were running. The answer was these sensors are responsible for the auto-rotation of the screen and view on the screen whenever a phone is rotated. As every phone has two different working types, horizontal and vertical, these sensors are useful for developers to know how the user is using the phone in a real-time environment. So, the next interesting thing was they didn't appear in the first results but they were on the second one when we capture OS activities as well as the specific app. One probable reason is they don't get information from these sensors directly, instead, they use a callback function. It means they are notified whenever the user changes the orientation of the phone. So, instead of developers bothering to handle this information, the Android itself gets the information in each period of time, process the data, and notifies developers through an API if the orientation change. That is why we get the activity from OS and not the specific application.

The next observation was the fact that all the applications were using WiFi and they were all advertising to make a profit. However, this is usual behavior for all freeware. So, we didn't look deeper into it.

Another interesting point in the results was that 18 applications use a camera and other related things to media. But, in our database, we had just 15 applications for QR code scanning which is normal behavior for them to use a camera. So, the issue was "What are the other three?". We get deeper into that and we concluded that the three following applications might be using camera-related things which they are not supposed to do:

- com.baladmaps: Balad, is an Iranian application that does nearly the same job as Google Maps. Since the government of Iran banned Google to access data in Iran, they built an application with the same

usage. However, it is a common thing in some Asian countries to do that. For instance, China has its own Google Maps too. It is called Baidu Maps which was launched in 2005. But, we can not claim any issues about Balad yet, since it should be tested again and with different methods too.

- nl.dtt.voicemail: Voice Memos SpeechNotes VoNo, this is an app that converts speeches or voicemails to texts. Although the usage appears to not do any camera-related processes, when we got deeper into the application, we find out why it has a log about accessing the camera, they put another ”extra but free” feature which is taking a picture and extracting the text from it. This feature is new in Android while it is a built-in feature in iOS. So, it is out of our list of suspicious free apps. Also, there were many positive comments about how it works and it seems that this application has something to offer in real. Another thing was the app got updated during our project periodically and we couldn't be sure whether the QR code scanning feature remained in the newer version.
- com.roku.remote: This one was really strange. The app seems to be updated and also there are no camera-related things in this application as well as the first one. So, it should be tested manually or by other tools to get confidential about whether it is a real issue or not.

The last observation was about the apps that use the microphone for different purposes. We observed 63 applications using the microphone sensor or speakers. They were mostly in the 3 following categories in which we clarified why the logs showed microphone usage for each of them.

- Health and Fitness: Most of the apps in this category which were developed by ”leap fitness group” were suspicious to use the microphone. However, we got deeper into the logs of the applications of this group and it seems to be a bug in our script to capture a wrong number as app id which causes this problem for just this type of application. But, this issue helps us to know a lot more about how this group is dominating the fitness online applications for both Android and iOS.
- Personalising: In this category, we observed several times that the applications which change the keyboard theme got the microphone access at the startup. After taking a look at the reason for doing so, we end up with the fact that because they are completely separate from the default keyboard, they should have permission to access the microphone as the microphone icon might appear sometimes. So, they get permission to ensure the usability of their theme in all different situations and modes.
- QR Code Scanners: It is pretty unusual for an application that scans QR codes to ask permission to use the microphone as well. So, we went deeper into the logs. The result was that many of the QR code scanners asked for microphone permission. So, we searched for the probable reason and it seems that it is a bug forcing developers to ask permission for the microphone when they want to use the camera as a QR code scanner. However, it should be tested more manually as well as other suspicious applications.

Finally, all the results are available for a deeper look at a request.

## 5 Future Work

There can be some developments after this project, it is included but not limited to
1- Adding the logging system to the main hardware layer. So, if the malicious application is using a side-channel (As it is mentioned in the limitations section), we can detect all of the usages of the sensors from any applications whether they are using them in a good way or not.
2- Add machine learning methods to see how we can wisely detect malicious applications which are using sensors without just cause.
3- Compared to the new version of iOS (Version 15), Android doesn't get the user to have the options like ”asking app not to track” and ”Allow while using”. The first one will prevent applications to collect the user behavior for future usage and the second one is like an active permission model which allows the application to use functionalities just in the case application is running in the foreground and not in the background. Adding these features can be future work for Android OS to improve its security and privacy.

## 6 Conclusion

This project was enlightening for numerous reasons, the first and for most was how surprised we were at the house of cards nature of the AOSP. This software is an abstraction on top of other abstractions, with almost no rhyme or reason why certain design choices were made. That fact alone becomes almost terrifying at the scale of deployment of Android. Further, we found that a majority of applications have the ability to run sophisticated attacks on touch input as most apps have access to sufficient sensor data to run a crude classifier. The last observation that we would like to share is that our rather unsophisticated approach to finding bad actors inside the app market found some unusual activity. We would to explore more just what percentage of bad actors are lurking in plain sight collecting data on their users because it is easy and valuable.

## 7 REF

[1] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, “Permission evolution in the Android ecosystem,”, 28th Annual Computer Security Applications Conference, 2012, pp. 31–40.
[2] Y. Zhauniarovich and O. Gadyatskaya, “Small changes, big changes: an updated view on the Android permission system,” Int. Symp. Res. Attacks, Intrusions, Defenses. Cham, Switzerland: Springer, 2016,pp. 346–367.
[3] Iman Almomani, Aala Al Khayer, “A Comprehensive Analysis of the Android Permissions System”, IEEE, November 2020.
[4] E. Alepis and C. Patsakis, “Unravelling security issues of runtime per-missions in android,” J. Hardw. Syst. Secur., vol. 3, no. 1, pp. 45–63, Mar. 2019.

[5] Google https://source.android.com/ The Android Open Source project.
[6] https://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/
[7] Adam J. Aviv, Benjamin Sapp, Matt Blaze and Jonathan M. Smith, “Practicality of Accelerometer Side Channels on Smartphones”, ACSAC, 2012 [8] Liang Cai and Hao Chen. “Touchlogger: inferring keystrokes on touch screen from smartphone motion.”, HotSec’11, 2011