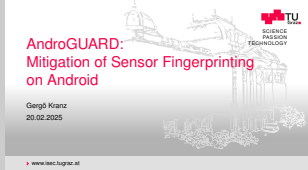
AndroGUARD: Mitigation of Sensor Fingerprinting on Android

Gergö Kranz

20.02.2025

SCIENCE
PASSION
TECHNOLOGY



Welcome to my presentation of AndroGUARD: Mitigation of Sensor Fingerprinting on Android.

Outline

1 Introduction
2 Background
3 Sensor Fingerprinting
4 Methodology
5 Approach
6 Implementation
7 Evaluation
8 Discussion & Limitations

Outline
1 Introduction
2 Background
3 Sensor Fingerprinting
4 Methodology
5 Approach
6 Implementation
7 Evaluation
8 Discussion & Limitations

Lets have a quick look at what we will talk about. First we will introduce the topic of fingerprinting different devices and mention some methods used in general. Then we will go into detail about sensor fingerprinting, and look at the methodology and the approach we applied. We will then present our implementation and show the evaluation steps we took to check if our work is functional. At the end we will discuss the limitations of our implementation.

Introduction

- Combination of information
- Used for targeted advertisements
- Does not require user permission

Introduction
- Combination of information
- Used for targeted advertisements
- Does not require user permission

Device fingerprints are created by using the given api to collect data about the device and its usage. The collected data can be for instance, the hardware information or the user configuration. This data is harmless and they are not unique and can not be used to identify and track users. But when multiple information is requested through the api and combined, they build a trackable id. This trackable identifier can be misused to track users. The combination of all the collected data is a unique fingerprint.

Introduction

- Combination of information
- Used for targeted advertisements
- Does not require user permission

Introduction
- Combination of information
- Used for targeted advertisements
- Does not require user permission

This unique fingerprint can be used to track users and their habits. This data then can be exploited for financial gain, like creating targeted advertisements for individual users, hurting the users privacy.

Introduction

- Combination of information
- Used for targeted advertisements
- Does not require user permission

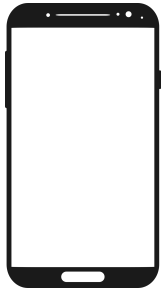
Introduction

- Combination of information
- Used for targeted advertisements
- Does not require user permission

Much of the required information that is used for the unique fingerprint can be requested through the android api without additional user-granted permission. Due to the absence of explicit consent and knowledge of the user, these permissionless fingerprints create a significant privacy threat.

Smartphone Fingerprinting

- Similar to browser fingerprinting
- Not as known as browser fingerprinting
- Zero permission identifiers

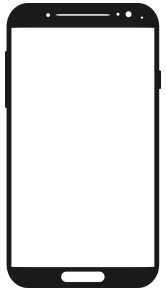


Smartphone Fingerprinting
- Similar to browser fingerprinting
- Not as known as browser fingerprinting
- Zero permission identifiers

Smartphone fingerprinting can be compared with the more widely known fingerprinting method applied in browsers. Browsers can be identified and fingerprinted by gathering various informations about the screen, installed fonts and extension. Smartphone fingerprinting requests similar characteristics, to create a fingerprint of the device ot is being run on.

Smartphone Fingerprinting

- Similar to browser fingerprinting
- Not as known as browser fingerprinting
- Zero permission identifiers

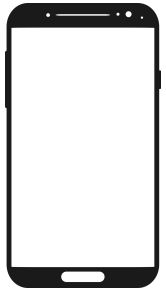


Smartphone Fingerprinting
- Similar to browser fingerprinting
- Not as known as browser fingerprinting
- Zero permission identifiers

Today there are already many well established papers and projects to mitigate browser fingerprinting techniques. Even though smartphone fingerprinting is similar to browser fingerprinting and also a privacy threat, it is not as widely known. Because it is not as well known, there is currently also a lot less protection for the devices.

Smartphone Fingerprinting

- Similar to browser fingerprinting
- Not as known as browser fingerprinting
- Zero permission identifiers

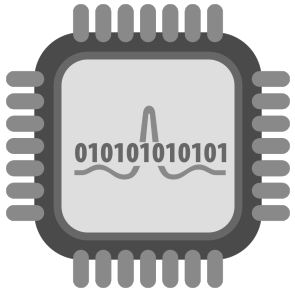


Smartphone Fingerprinting
- Similar to browser fingerprinting
- Not as known as browser fingerprinting
- Zero permission identifiers

To create a fingerprint of a mobile device in many cases zero permission identifiers are used. These can be information about the device, configurations and different sensors, which do not require elevated permissions to access.

Fingerprinting Sensors

- Measurement inaccuracy of sensors
- Simple to fingerprint via machine learning algorithm
- Constant over the sensors lifetime

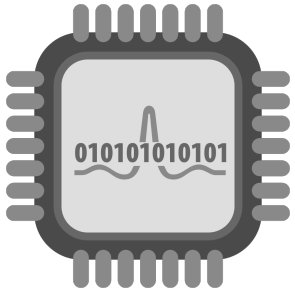


Fingerprinting Sensors
- Measurement inaccuracy of sensors
- Simple to fingerprint via machine learning algorithm
- Constant over the sensors lifetime

The main topic of ours is to focus on sensor fingerprinting. These fingerprints are created by reading the sensor values and determining the measurement inaccuracy. This measurement error is created due to the not-perfect manufacturing processes of the sensors. Multiple sensors can be used for fingerprinting, such as gyroscopes and accelerometers. These are not the only sensors, that can be used but are already present in nearly all of the mobile devices today. This is why we focused our patch on these sensors.

Fingerprinting Sensors

- Measurement inaccuracy of sensors
- Simple to fingerprint via machine learning algorithm
- Constant over the sensors lifetime

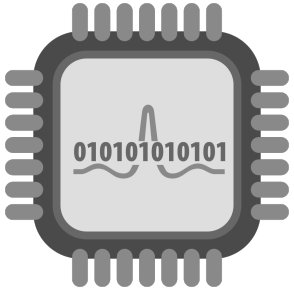


Fingerprinting Sensors
- Measurement inaccuracy of sensors
- Simple to fingerprint via machine learning algorithm
- Constant over the sensors lifetime

Fingerprinting sensors can be done in multiple ways. The most simple one is to train a machine learning algorthim to match the recorded sensor values to the device they were recorded from.

Fingerprinting Sensors

- Measurement inaccuracy of sensors
- Simple to fingerprint via machine learning algorithm
- Constant over the sensors lifetime



Fingerprinting Sensors

- Measurement inaccuracy of sensors
- Simple to fingerprint via machine learning algorithm
- Constant over the sensors lifetime

It has been already proven by multiple papers, that these fingerprints based on the builtin sensor error are constant over the lifetime of the device. These fingerprints based on the measurement error of sensors are also unique enough to be used to identify the device they were recorded from.

Main Question

How to protect against sensor fingerprinting

Main Question
How to protect against sensor fingerprinting

The main question of us is: How to protect against sensor fingerprinting? There have been already some papers that focused on this problem. Some of these also had some proposed solutions to mitigate the privacy risk.

Proposed Solutions

Calibration

- Systematic adjustment of sensor readings
- Correcting the sensor data

Noise Generation

- Introduces variability into the sensor data
- Masks the original values

Proposed Solutions

Calibration
- Systematic adjustment of sensor readings
- Correcting the sensor data

Noise Generation
- Introduces variability into the sensor data
- Masks the original values

There are mainly two proposed solutions. Calibration and noise generation with each having a different approach. The first one we will look at is calibration. We can mitigate the builtin factory imperfections of a sensor by recalibarting it to correct the error. Noise geration on the other hand makes the error larger and random by introducing variability into the sensor data to mask the original value.

Challenges

Calibration

- Requires user awareness and interaction
- Requires precision

Noise Generation

- Degrade the functionality of applications
- Code has to be modified

Challenges

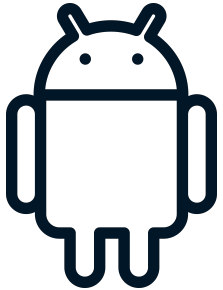
Calibration
- Requires user awareness and interaction
- Requires precision

Noise Generation
- Degrade the functionality of applications
- Code has to be modified

Both of the proposed solutions have their challages. Calibration requires user awareness and interaction. Users have to actively and precisely calibrate their sensors to get rid of the sensor imperfections and not just change them. Noise generation does not require user interaction. But due to the noise and decreased accuracy of the sensor data, applications would also decrease their functionality. Also in order to introduce the masking noise into the application the code would need to be modified.

Our Methodology

- Noise Generation
- Patch application vie A2P2 framework

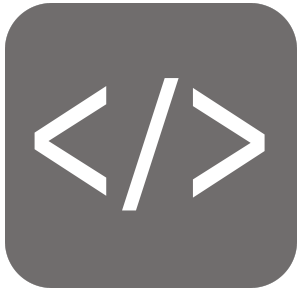


Our Methodology
- Noise Generation
- Patch application vie A2P2 framework

Our methodology uses the proposed noise generation, to mask the builtin error of sensors. We choose this because we believe it to be much easier to scale up than to rely on the users precision to calibrate their device to perfection. To introduce our custom code responsible for the noise generation we are using the android application patching pipeline, short a2p2. This enables easy integration into a number of android apps.

Modifying the Sensor API

- Intercept calls to registerListener method
- Provide modified values to onSensorChanged method



Modifying the Sensor API
- Intercept calls to registerListener method
- Provide modified values to onSensorChanged method

To replace the original sensor values with our masked ones we have to modify the sensor api. We have intercept calls to the registerListener function. We do this in order to redirect the sensor values sent by the system to our noise generating function. Also we will need to implement the onSensorChanged method to pass the masked sensor value back to the original calling method.

Noise Generation

- Adds random gain and offset to every value
- Masks values
- Loss of precision

Noise Generation
- Adds random gain and offset to every value
- Masks values
- Loss of precision

In our approach, the selected noise generation adds a random gain and offset to every single sensor value, masking the built-in factory error. However, we have to keep in mind, that due to the applied noise there will be a loss of precision in the sensor values, degrading the app functionality.

Implementation

- Intercept Method
- Noise Generating Function
- Random Value Generation Function

Implementation
- Intercept Method
- Noise Generating Function
- Random Value Generation Function
JAVA

Our implementation has some fundamental methods which ensure the functionality of our patch. The first one is the intercept method.

Intercept Method

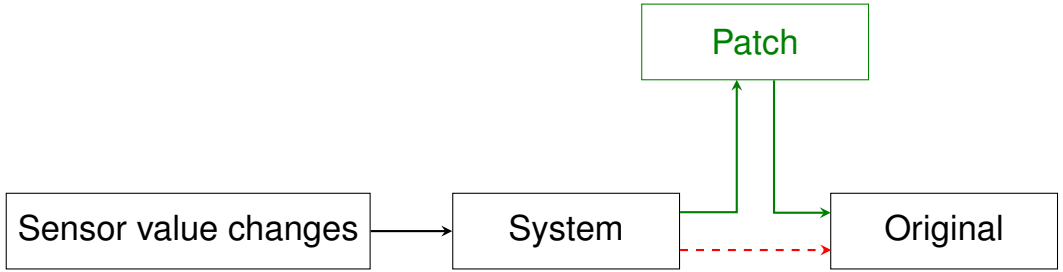


Figure: The function calls from the system are intercepted by our patch and forwarded after modification to the original function.

The intercept method is responsible for intercepting the sensor values from the system before reaching the original method and forward them to our patch. Our patch then generates and applies noise to these recieved values and passes them to the original method.

Implementation

- Intercept Method
- Noise Generating Function
- Random Value Generation Function

Implementation
- Intercept Method
- Noise Generating Function
- Random Value Generation Function
JAVA

Another important part of our implementation is the noise generating function.

Noise Generating Function

$$value_{new} = \frac{(value_{old} - offset_{sensor})}{gain_{sensor}}$$

Figure: Function from the paper: Tracking Mobile Web Users Through Motion Sensors: Attacks and Defenses

Noise Generating Function

$value_{new} = \frac{(value_{old} - offset_{sensor})}{gain_{sensor}}$

Figure: Function from the paper: Tracking Mobile Web Users Through Motion Sensors: Attacks and Defenses

By subtracting a random offset and divinding a random gain from the original value we genarate the new obfuscated sensor value. The random values of the offset and gain are generated from a carefully selected range responsible for the amplitude of the noise.

Implementation

- Intercept Method
- Noise Generating Function
- Random Value Generation Function

Implementation
- Intercept Method
- Noise Generating Function
- Random Value Generation Function
JAVA

The most important function is the random value generator function. Ensuring the randomnes of the used values for the noise generation is important to prevent creating new fingerprintable features.

Application of Patch

- Only requirements are
  - JAVA JRE
  - A2P2
  - APK to be modified

```
java -jar a2p2.jar <app>.apk !  unpack !  apply patch.zip
                    static !  pack !  sign !  ./
```


Application of Patch
- Only requirements are
  - JAVA JRE
  - A2P2
  - APK to be modified
java -jar a2p2.jar <app>.apk ! unpack ! apply patch.zip
static ! pack ! sign ! ./

The application of our patch is very straightforward. We only need to install JAVA runtime environment and download the latest a2p2 release and the apk of the app the we want to patch. If all these reqirements are met, the app can be patched by executing a simple single line command.

Testing

- Functionality
- Effectiveness
- Usabilty

Testing
Functionality
Effectiveness
Usability

To determine if our patch mitigates fingerprintability we had to perform some tests. First we had to determine the functionality of our patch.

Functionality

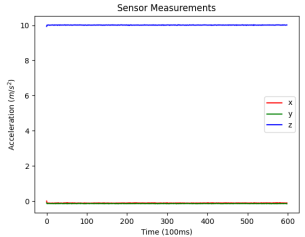


Figure: recorded values before the patch

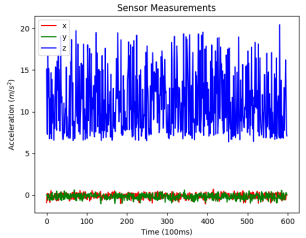


Figure: recorded values after the patch

Functionality

Figure: recorded values before the patch

Figure: recorded values after the patch

To check the functionality of our patch we had to prove that sensor values are intercepted and modified before passing them to the original function. We did this by recording sensor values over a small period of time with an android app. Then we applied our patch to the app we used to record the values and compare the recorded results. As we can see on the left figure, the recorded sensor values were constant before the patch. After the introduction of the patch as seen in the right figure, the values got inconsistant.

Testing

- Functionality
- Effectiveness
- Usabilty

Testing
Functionality
Effectiveness
Usability

Then we had to check the effectiveness of our patch in mitigating fingerprintability.

Effectiveness

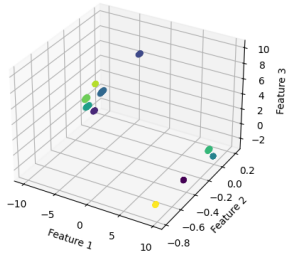


Figure: knn decision boundaries before the patch

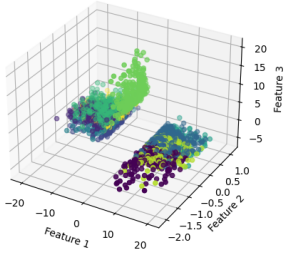


Figure: knn decision boundaries after the patch

Effectiveness
Figure: knn decision boundaries before the patch
Figure: knn decision boundaries after the patch

To test this we trained a k-nearest neighbour machine learning model with the recorded sensor values before and after the application of our patch. We can observe that before our patch the decision boundaries of the trained model were very prominent and easily differentiable, as it can be observed in the first figure. This lead to 100% accuracy in the predictions of the model. After our patch the decision boundaries were not that easily distinguisable as shown in the second figure. This led to a noticable decrease in accuracy of our trained model.

Testing

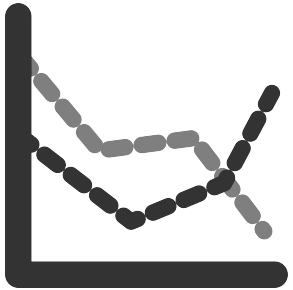
- Functionality
- Effectiveness
- Usabilty

Testing
- Functionality
- Effectiveness
- Usabilty

We also tested if the patched apps retained their functionality and usabilty. To test this we patched a motion controlled game and played for a while. We were still able to play the game without significant problems. There was only a slightly noticable shaking of our motion controlled character.

Noise Level Adjustment

- Increasing noise decreases fingerprintability
- Increasing noise decreases functionality



Noise Level Adjustment
- Increasing noise decreases fingerprintability
- Increasing noise decreases functionality

We also evaluated the level of the applied noise, by testing different noise levels. Increasing the nosie we noticed it decreased the fingerprintability, but also decreased the apps functionality significantly. In order to retain functionality of the patched apps we have to be carefull in the selection of the noise level. We have to balance the level of the applied noise in order to have the most effective mitigation of fingerprinting but keeping the apps usable.

Discussion & Limitations

- Limited amount of test devices
- Could not be done sufficiently due to limited access to supported hardware

Discussion & Limitations
- Limited amount of test devices
- Could not be done sufficiently due to limited access to supported hardware

Due to the range of our testing we only have limited proof of our patch currently. Our test was performed only with a limited number of devices in a controlled environment. But we are confident that our patch would also make sensor fingerprinting more complicated on a larger scale in an everyday environment. Our confidence is based on our findings and other pubished papers.

Conclusion

- Easy application of the patch
- Masking the sensor values decreases fingerprintability

Conclusion
- Easy application of the patch
- Masking the sensor values decreases fingerprintability

We believe that our implementation of androguard is a possible solution to lower privacy violations. By easily applying our patch to applications and masking the builtin error of sensors with added noise, we mitigate sensor fingerprinting on android.