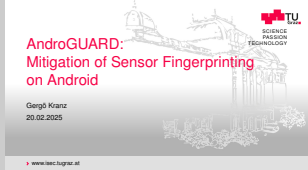
AndroGUARD: Mitigation of Sensor Fingerprinting on Android

Gergö Kranz

20.02.2025

SCIENCE
PASSION
TECHNOLOGY



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

Outline

Outline

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

android

Lets have a quick look at what we will talk about. First we will introduce the topic of fingerprintng different devices and mention some methods used in general. Then 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. In the end we will discuss the limitations of our implementation.

Introduction

- Misuse of the Android API
- Used for targeted advertisements
- Does not require user permission

Introduction
- Misuse of the Android API
- Used for targeted advertisements
- Does not require user permission

Fingerprints are created by misusing the android api in a way it way not intended to. The android api is used to query different data from the device. Individual data can be requested through the api like it was intended to and used harmlessly. But when a multitude of information about the device is requested and is combined it can be harmful. The combination of these easily accessed information can be unique to a user and his device.

Introduction

- Misuse of the Android API
- Used for targeted advertisements
- Does not require user permission

Introduction
- Misuse of the Android API
- Used for targeted advertisements
- Does not require user permission

This unique fingerprint can be used to track users and their habbits. This data then can be used for example to create targeted advertisements for individual users.

Introduction

- Misuse of the Android API
- Used for targeted advertisements
- Does not require user permission

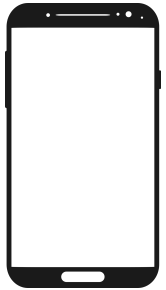
Introduction

- Misuse of the Android API
- Used for targeted advertisements
- Does not require user permission

Many of the required information that are 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 permission less fingerprints create a significant privacy threat.

Smartphone Fingerprinting

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

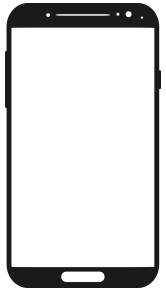


Smartphone Fingerprinting
- Similar to browser fingerprinting
- Not as known as browser fingerprinting
- Zero permission identifiers

Smartphone fingerprinting is very similar to browser fingerprinting. Browsers can be identified and fingerprinted, by gathering various informations about the screen, installed fonts and extension.

Smartphone Fingerprinting

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

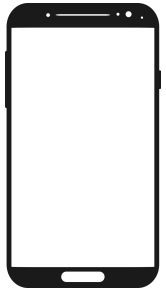


Smartphone Fingerprinting
- Similar to browser fingerprinting
- Not as known as browser fingerprinting
- Zero permission identifiers

Even though browser fingerprinting is similar to browser fingerprinting and just as a privacy threat it is not as widely known. Because it is not as widely known there are currently also a lot less of protections on devices.

Smartphone Fingerprinting

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

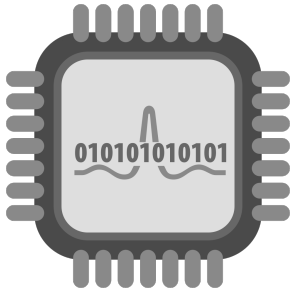


Smartphone Fingerprinting
- Similar to browser fingerprinting
- Not as known as browser fingerprinting
- Zero permission identifiers

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.

Fingerprinting Sensors

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

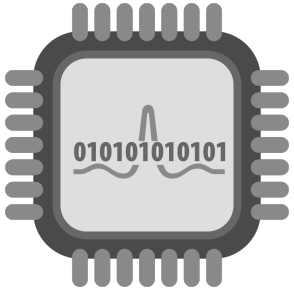


Fingerprinting Sensors
- Measurement inaccuracy of sensors
- Simple to fingerprint via machine learning algorithmus
- Constant over the sensors lifetime

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 and we will focus on these sensors in the following.

Fingerprinting Sensors

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

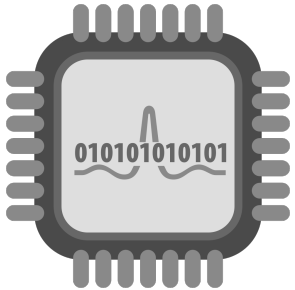


Fingerprinting Sensors
- Measurement inaccuracy of sensors
- Simple to fingerprint via machine learning algorithmus
- Constant over the sensors lifetime

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 algorithmus
- Constant over the sensors lifetime



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

It has been already proven by multiple pappers, that these fingerprints based on the builtin sensor error are constant over the lifetime of a sensor. These fingerprints 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 the problem of sensor fingerprinting. 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, which each having a different approach. The first we will look at is calibration. We can mitigate the builtin factory imporfections 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

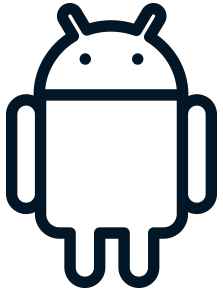
Calibration
- Requires user awareness and interaction
- Requires precision

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

Both of the proposed solutions have their respective challages. Calibration requires user awareness and interaction. They have to actively and precisely calibrate their sensors to get rid of and not just change the sensor imperfections. 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

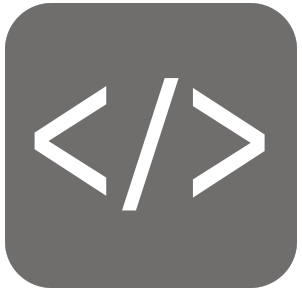


Our Methodology
- Noise Generation
- Patch application vie A2P2 framework

Our nethodology 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 then to rely on the users precicion to calibrate their device to perfection. To introduce our custom code responsible for the noise generation we are using the android application pipeline, short a2p2, which 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 in order to let our masking class recieve the sensor values instead of the original calling method. Also we will need to implement the onSensorChanged method to pass the recieved sensor value from the system after adding noise 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 original value with their error. Due to applied noise there will be a loss of precision in the sensor values degrading 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 tunctionality of our patch. The first is the intercept method.

Intercept Method

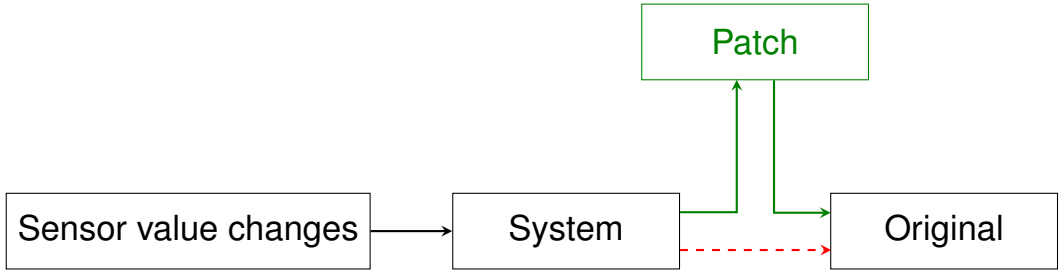


Figure: The function calls from the system are intercepted by our patch and forwarded after modification to the original function.

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 this 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

An other important part of our implementation is the noise generating function.

Noise Generating Function

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

Noise Generating Function

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

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 nosie.

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 generation function, which ensures the randomnes of the values used for the noise generation to prevent newly created fingerprintable features.

Application of Patch

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

Application of Patch
- Straightforward application
- Only requirements are
  - JAVA JRE
  - A2P2
  - APK to be modified

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 command.

Testing

- Functionality
- Effectiveness
- Usabilty

Testing
Functionality
Effectiveness
Usability

To determine if our patch mitigates fingerprintability we have to perform some tests. To check the functionality of our patch and controll that sensor values are intercepted and modified before passing them to the original function.

Functionality

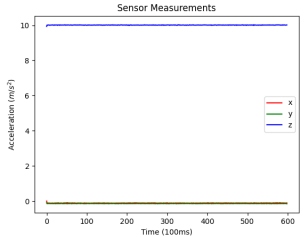


Figure: recorded values before the patch

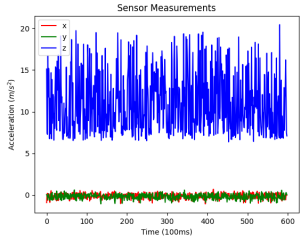


Figure: recorded values after the patch

We do this by recording sensor values over a small period of time with an android app. Then we apply our patch to the app we used to record the values and compare the recorded results. As we can see the recorded sensor values were constant before the patch but got inconsistant after the application of our patch.

Functionality

Figure: recorded values before the patch

Figure: recorded values after the patch

Testing

- Functionality
- Effectiveness
- Usabilty

Testing
Functionality
Effectiveness
Usability

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

Effectiveness

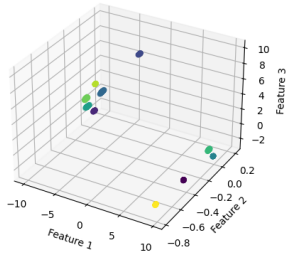


Figure: knn decision boundaries before the patch

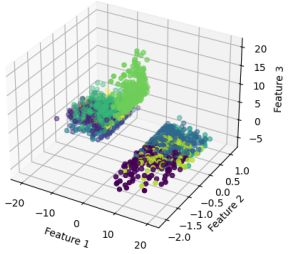


Figure: knn decision boundaries after the patch

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

To test this we trained a knn machine leaarning 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 prominant and easily differentiable. This lead to 100% accuracy in the predictions of the model. After our patch the decision boundaries were not that easily distinguisable and the trained models accuracy decreased noticably.

Testing

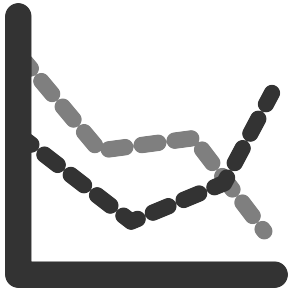
- Functionality
- Effectiveness
- Usabilty

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 were still able to play the game without significant problems.

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. Increasing the nosie we noticed it decreased the fingerprintability, but also decreased the apps functionality significantly. We have to balance the level of the applied nosie in order to have the most effective mitigation of fingerprinting but stil retaining the apps functionality.

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

Our test was done with a limited number of devices in a controlled environment. But we are confident that based on our findings and other pubished papers our patch would also mitigate sensor fingerprintability on a larger scale in an everyday environment.

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 nosie we mitigate sensor fingerprinting on android.