Information Inference based on Barometer Sensor in Android Devices

by

Alireza Hafez

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Software Engineering and Intelligent Systems

Department of Electrical and Computer Engineering

University of Alberta

Abstract

Most modern smartphones are equipped with barometer sensors. Accessing barometers does not require any permission or user notification, as it is deemed to be harmless. We show, however, that by simply reading low-rate barometer's samples, any background application can detect user's finger taps on the device's touch screen, and infer information about the positions of the taps. This can reveal sensitive information to background applications that, for security reasons, do not have access to the touch screen input. In addition, we show that barometers' samples reveal information about the smartphone's speaker activity. Specifically, using simple machine learning and classification methods, we detect with high accuracy whether the speaker is silent or playing a given sound such as a ringtone.

“Knowledge, like air,

is vital to life. Like air,

no one should be denied it, ” said V

- - Alan Moore, V for Vendetta

To my dearest ***Dorsa****,*

without whom it would not have been possible.

Acknowledgements

I would like to thank my supervisor **Dr. Majid Khabbazian** who has helped me throughout my studies. I am also thankful to my supervisory committee members **Dr. Masoud Ardakani** and **Dr. Hai Jiang** who have given me valuable comments and suggestions on my work. I should also thank **Dr. Mahdi Tavakoli Afshari** for chairing the examination. Finally, I like to thank all those whose actions, directly or indirectly, have helped me in this journey.

Contents

List of Tables

List of Figures

List of Abbreviations

Notation	**Description**
ANSI	American National Standards Institute
API	Application Programming Interface
APK	Android Application Package
App	Application
ARM	Advanced RISC Machine
CRT	Cathode-Ray Tube
ePTFE	Expanded Polytetrafluoroethylene
GPS	Global Positioning System
Hz	Hertz
IEC	International Electrotechnical Commission
IP	Ingress Protection
NDK	The Native Development Kit
NEMA	National Electrical Manufacturers Association
OS	Operating System
RIG	Runtime Information Gathering
SDK	Software Development Kit
SVM	Support Vector Machine
TEE	Trusted Execution Environment
UI	User Interface

Chapter 1

Introduction

1.1 Motivation

Smartphones have become an integral part of many people's lives. In fact, there are now more than 3.5 billion smartphone users in the world [1]. We are increasingly relying on them for various tasks including, communications, navigation, fitness tracking, gaming, social networking and banking. To support this wide set of tasks, smartphones have been equipped with a large collection of sensors ranging from microphones and cameras to GPS, accelerometers, gyroscopes and barometers.

Barometers are now found in most mid-range to high-range smartphones. In addition to measuring air pressure, barometers can be used as altimeters as pressure data can be translated to altitude. The sensitivity of barometer samples allow detection of altitude changes as little as one meter [2]. As such, many applications, including floor level detection in indoor tracking, rely on barometer sensors. Barometers can also be used in low-power outdoor tracking as they consume significantly lower power than GPS.

The abundance and sensitivity of data collected by sensors in smartphones is a major source of privacy concern. To mitigate these privacy concerns, smartphones have put in place permission mechanisms that allow users to control apps' access to many of the smartphone's sensors. For example, in Android, apps must get permission from users either at install time or at runtime to access sensitive sensors. In addition, access to sensitive inputs may be taken away when the app goes to the background. For instance, when an app goes to the background, its access to the touch screen input is stopped, as this input can have sensitive information such as typed password or text that belongs only to the foreground app. However, there are sensors such as barometers whose access requires no permission or user notification because their input data is considered non-sensitive. These sensors may be used as side-channels to gain sensitive information.

In this work, we are interested to know if barometers can be used as side-channels to gain sensitive information. A major challenge in doing so is the low sampling rate of smartphone's barometers, currently limited to 25Hz. Nevertheless, we show two possible side-channel information leaks based on internal barometer readings. First, we show that in smartphones with ingress protection, any app in the background can detect touch screen user's finger taps (gentle user touch on the screen with a finger), and to some extent their positions. Second, we show that any app can gain information about earpiece speaker activity without permission.

Thesis Organization. The remainder of this thesis is organized as follows. In the remainder of this chapter, we provide background information, and cover related work. In Chapter 2, we present our results, which include our initial observation, and data gathering methods. We conclude, and mention possible future work in Chapter 3.

1.2 Background

In this section, we cover the background information deemed to be necessary for understanding the concepts and terms we used throughout this thesis.

1.2.1 Side-Channel Attacks

Side-channel attacks infer sensitive information by exploiting information leakage of computing devices or implementations [3]. The source of this information leakage is typically an implementation flaw in hardware or software.

Side-channel attacks are not new. The idea of side-channel attacks started decades ago. In 2007,the National Security Agency of the United States declassified a paper [4], which shows that these attacks started to emerge in the early 1960s. In this period, Britain was trying to join the European economic community.They were worried about France blocking their entry, so they asked their intelligence agency to gather intelligence about France's stance on this matter. In pursuing this, the agency discovered that the enciphered french communication signal carries a weak signal that is extractable. The extracted signal was discovered to be the actual communication in plain text, which leaked through the cipher machine [5]. Another example of early side-channel attacks is a proof of concept (presented by Van Eck Phreaking in 1985) which showed that CRT displays leak their contents through electromagnetic emanations [6].

By emerging new types of devices such as personal computers and smartphones, the attack surface has expanded, and further side-channel attacks appeared. Some of them exploit the physical characteristics of the device, such as Backes et al. [7] work on recovering the words printed by the dot-matrix printers using acoustic analysis. Some others exploit the human interactions with the target device, like Aviv et al. [8], which infers the lock screen patterns on Android smartphones by analyzing the smudges on the screen as a result of human interaction with touch screens. In this thesis, we exploit the physical characteristics of the touch screen alongside the side effects of human interaction on internal barometric pressure to infer the user input. We also investigate the side effects of internal speakers' activity on the barometric pressure to determine the internal earpiece speaker activity.

Spreitzer et al. [3] divide side-channel leaks in mobile devices into two categories: intended and unintended information leaks [3]. Intended information leaks are those that are either considered harmless (e.g. data consumption) or accessible through OS APIs, SDK and NDK in Android OS, to other third-party apps to support their functionalities. In contrast, unintended information leaks result from exploiting characteristics of hardware such as power consumption, electromagnetic and acoustic emissions [3].

According to Spreitzer et al. categorization, our proposed attacks fall into the first category (i.e., intended information leaks). In addition, our attacks can be categorized as Runtime Information Gathering (RIG) attacks. RIG attacks refer to the attacks where a malicious app collects or uses any data provided to or produced by the app under attack during its lifecycle [9]. In our case, the malicious app listens and profiles the barometric pressure sensor data alongside the other apps running on the device.

1.2.2 Android Permission Model

Permissions are an essential part of the Android security model. Each permission controls access to a set of Android or Java APIs. In this security model, using any of these controlled APIs without special permission will result in a security exception [10]. Such an exception indicates a security violation and as a result the access to the controlled API will remain blocked.

The permission model in Android devices [11] is a gateway for third-party apps to gain access to different resources such as internal/external storage for data read and write, Internet and various sensors. Android defines four permission levels: normal, signature, dangerous and special permissions [12].

- Normal permissions control the access to APIs that are not harmful to the user but can damage the user experience. For example, normal permissions control the access to SET_WALLPAPER, which can set the phone's wallpaper.

- Signature permissions control access to the system level and dangerous operations, such as deleting the installed apps and packages. To operate at this level, the app must be signed by the device's manufacturer.

- Dangerous permissions control the access to potentially harmful APIs and resources, like access to fitness data, cameras, microphones, or sending text messages.

- Special permissions are a subcategory of dangerous permissions. However, the process of granting these permissions to third-party apps is different. Users have to go through the settings of the device and confirm the access to such capabilities upon request. Accessing device admin apps, battery optimizations, display over other apps are some examples that require this type of permission.

Before Android Marshmallow (aka, Android 6.0), Android used an installation time permission model. In this model, the app shows the list of all its required permissions with a brief description of what they do at the time of installation. In this model, the installation fails if the user does not grant the app all the permissions in the list [13]. According to a study by Felt et al. [12], about one-third of the Android apps were overprivileged and had access to dangerous and unneeded permissions. Starting with Android Marshmallow in 2015, Android introduced the runtime permission model. In this model, normal permissions are requested at the installation time, while dangerous permissions, such as camera access, are requested at runtime [13]. This permission model allows user to install an app without giving it any dangerous permissions. It also gives users the chance to deny permission when the motivation behind a requested access is not clear [13].

There are some sensors whose access does not require any permission. In Android devices, these sensors are:

- Accelerometer
- Gyroscope
- Barometer
- Ambient light sensor
- Proximity sensor
- Magnetic field sensor

However, according to Android documentation since Android 9 (API 28), apps running in the background have some restrictions on receiving sensor events [14]. There are two main problems with this approach. First being, that according to Section 1.2.8, it leaves more than 60% percent of the existing Android devices vulnerable. Second, There are workarounds around this restriction. For example, running the app as a foreground service or with START_STICKY flag on the activity and using a notification in the notification bar to make Android identify it as foreground.

Android also supports multitasking and background services. The combination of unregulated access in the list above and the multitasking nature in Android can lead to information leakage of sensitive user information via RIG attacks.

1.2.3 Ingress Protection

Smartphones are an inseparable part of our daily lives and activities, and as a result, they go through quite an abuse, including getting exposed to dust and liquids. As a response to this problem, manufacturers started incorporating ingress protection to their smartphones to prolong their lifespan. Most of them go through thorough testing and get an Ingress Protection (IP) rating, which is based on National Electrical Manufacturers Association's (NEMA) Standard approved by American National Standards Institute (ANSI).

IEC 60529, Degrees of Protection Provided by Enclosures, is the source of IP rating for today's smartphones [15]. The rating uses the symbol IP followed by two digits. The first digit refers to the ingress protection against solids. This scale goes from 0 to 6, where 0 is no protection, and 6 is dust-tight. Second digit refers to the protection against water. The digit ranges from 0 - 9; the higher the number, the more the protection [16]. Most of the mid to high-end Android smartphones are either IP68 or IP67 certified. These ratings imply that they're dust-tight and also protected against immersion in water, and it's effects on the device [16].

1.2.4 Barometer and barometric vents

A barometer, also known as the air-pressure sensor, is now found in most mid-range to high-end smartphones (e.g. iPhone 6/7/8/X/11, Samsung note 6/7/8/9/10/20, Google Pixel 2/3). It can act as an altimeter and provide a rough altitude to improve GPS accuracy in vertical direction [17], and indoor navigation by revealing floor changes [18], and provide additional data in applications such as fitness apps to improve their accuracy [19].

In devices with ingress protection, barometric vents are added to allow an internal barometer to measure outside atmospheric pressure. This also protects the device's seals and its internal components from the stress that can be placed by the difference in air pressure inside and outside the device. Today, most smartphones use their speaker and microphone holes as their barometric vents. They use polytetrafluoroethylene (ePTFE) meshes on these vents to keep the device water resistant [16]. They are structured to prevent penetration of dust particles and to deter water while allowing a flow of air to equalize internal and external barometric pressures. The air pressure equalization is, however, slow and occurs over a short period of time. This makes it possible to capture sudden changes to the internal pressure using low-rate barometric samples provided by the barometer.

Figure 1.1 shows the placement of the barometer sensor inside our testing smartphone, Samsung Galaxy S10 Plus. The barometer used in this smartphone is an ultra-compact LPS22HH pressure sensor by STMicroelectronics. According to the LPS22HH datasheet, the sensor is capable of sampling air pressure at rate 200 Hz [20]. However, the Android API limits the output pressure sampling rate to 25 Hz.

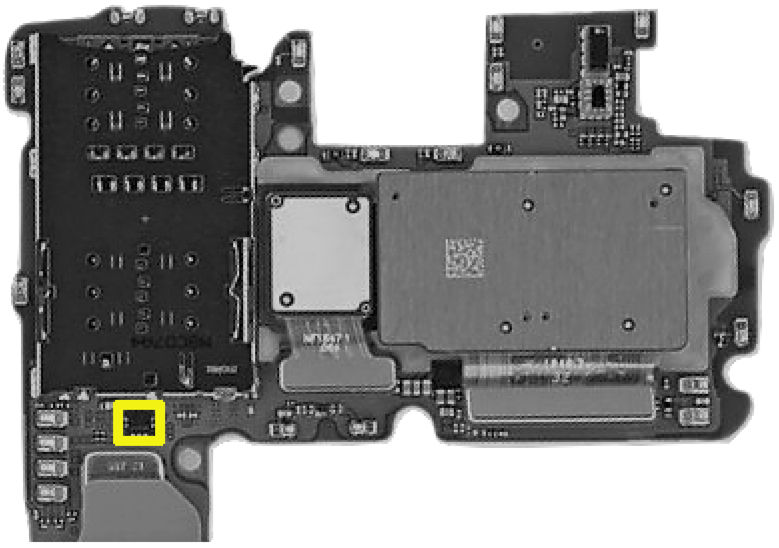

Figure 1.1: Samsung Galaxy S10 Plus barometer sensor.

1.2.5 Earpiece Speaker

Figure 1.2 shows the Samsung Galaxy S10 Plus earpiece speaker. Vent holes on both sides of the speaker allow air to get in and out of the speaker when it plays a sound.

Sound is a pressure wave: it consists of a repeating pattern of low-pressure and high-pressure regions. Moreover, both the barometer and earpiece speaker are inside the smartphone. Therefore, it is natural to ask if the barometer would leak information about the activity of the speaker even when it samples air pressure at a low rate. In this work, we show that this is indeed the case.

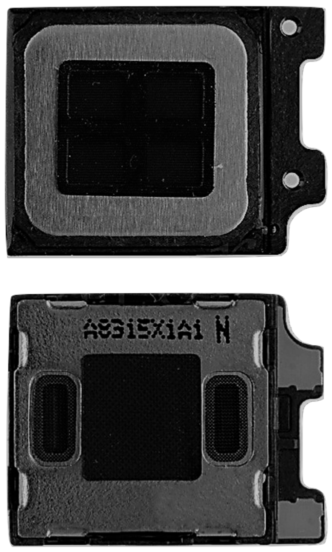

Figure 1.2: The Samsung Galaxy S10 Plus earpiece speaker with vent holes on both sides to allow air to get in and out.

1.2.6 Trusted Execution Environment

Trusted Execution Environment (TEE) is an isolated, secure and tamper-resistant environment [21]. It guarantees the integrity of code being executed and data being processed inside of it. TEE provides a separation between normal and secure world. The normal world runs the Android OS, and the secure world handles the security operations [22]. TEE provides a secure way for sensitive apps like banking, mobile payment, and healthcare to store and process their sensitive data, like user credentials or keychains [23].

1.2.7 Samsung KNOX

Samsung KNOX is a secure container; it provides a secure environment alongside the normal user environment. KNOX allows the running and configuration of enterprise apps in an isolated environment [24]. KNOX isolates the KNOX apps (apps that are running on top of the KNOX platform) from other user apps. KNOX's runtime protection relies on ARM TrustZone (an implementation of the TEE standard) accompanied by SELinux in Android OS [24].

1.2.8 Android Updates and Version Distributions

In the Android ecosystem, there is a significant delay between the time a new version of Android is released and when phone manufacturers adopt the new version to their phones. This delay usually happens for two main reasons. First, because most vendors tend to customize the Android to their will and add extra features and functionalities (like their own user interface or home screen) to it. This makes it time-consuming to adopt a new release or apply a bug fix to existing smartphones. Second, the device might be out of its support period; most high-end Android devices get up to 18 months of update and support. As the day of writing this Thesis, Android 11 (released on February, 2020) is the most recent version, and only 39.5 percent of all Android devices run Android 9 (released on August, 2018) and higher (SDK 28+) [25].

1.2.9 Support Vector Machine

A support-vector machine (SVM) is a popular classification method. It gets a set of n-dimensional points, and constructs a hyperplane or a set of hyperplanes to separate them. In this work, we used SVM for two main reasons. First, the SVM's classification results in our study were at least as good as the results of other classification methods we used, including trees and ensemble. Second, there are highly efficient SVM libraries for Android OS (e.g., AndroidLibSVM in NDK [26]). This allows an app to perform classification at the user's end (i.e., on the smartphone) without consuming too much power.

1.2.10 Cross-Validation

Cross-validation is a simple statistical method commonly used in applied machine learning to test the model's behaviour towards new or unseen data, especially when the dataset size is small [27]. In addition, cross-validation is a robust preventative measure against overfitting [27]. The general procedure of a k-fold cross-validation is as follows:

- Randomly shuffle the dataset
- Split the dataset into k equally sized groups
- For each group:
  - Mark the group as the test set
  - Mark the remaining groups as training sets
  - Train the model on the training sets and evaluate it on the test set
  - Store the evaluation score
- Summerize the model's performance based on the derived k evaluation scores

In a k-fold cross-validation, the performance of the model is usually summerized as the mean of the derived k evaluation scores [28]. In this work, we set $k = 5$ as test error rate estimations resulted from this setting do not suffer from a high bias or a high variance [29].

1.3 Contributions

In this thesis, we show that the barometer sensor in water-sealed smartphones can leak user's sensitive information. To this end, we present two proof of concepts. First, we show that air pressure low-rate samples from the smartphone's barometer can be used to detect gentle finger taps on the touch screen, and to some extent, the position of the tap. This improves and extends the previous work by Quinn et al. [30], which showed that these air pressure samples could be used to detect long-press and force touch [30]. Our second main contribution is to show, for the first time, that air pressure low-rate samples of the smartphone's barometer can reveal information about the smartphone's earpiece speaker activity. To the best of our knowledge, there is no prior work that studies the feasibility of detecting speaker activities, or detecting sounds in general, using smartphone's barometers. These findings are alarming because any app currently can access the smartphone's barometer without any permission or user notification.

1.4 Related Work

We cover two types of related work: 1) applications that rely on smartphone's barometric air pressure samples; 2) side-channel attacks that attempt to infer touch screen input.

1.4.1 Barometer Senors' Applications

Barometric air pressure measurements by smartphones' barometers can be used to estimate altitude, and detect altitude changes of as little as one meter [19]. Because of this, barometers on smartphones have been used in many applications such as aiding GPS [17], detecting floor level for indoor positioning [31], and improving calorie estimation in fitness apps [19].

Smartphone barometers have not been explicitly used in the literature for side-channel attacks. However, existing work suggest that barometer's measurements leak information about users' activities and surroundings. Sankaran et al. [19] showed that barometer data can be used to detect user's activities IDLE, WALKING, and in VEHICLE. Ho et al. [32] exploited barometer data to infer driving routes. Moreover, sudden changes in barometric data were utilized by Wu et al. [33] to detect the buildings' door opening/closing events. In an attempt to provide force sensing functionality in smartphones that lack integrated force-sensing hardware, Quinn showed that barometers on smartphones with ingress protection can estimate robotic forces on their touch screen. One of our main contributions, as mentioned earlier, is to show that barometers on such devices are able to detect even gentle finger taps, and to some extent, their positions.

1.4.2 Touch Screen Input Inference

The touch screen, which is the main source of user input, carries some of the most valuable user information, such as banking credentials, passwords, and text messages. Consequently, accessing this input and protecting it have been at the center of attention of adversaries, and security solutions.

One of the earliest side-channel attacks on touch screen input in Android devices was achieved by using the motion sensors. Cai et al. [34] is one of the earliest works that exploited the internal motion sensors, accelerometer and Gyroscope to track the small vibrations and pivots in the device when user interacts with the device to extract the user keystrokes on a numpad. Later, Spreitzer et al. [35] showed that by exploiting the light sensor in Android smartphones, it is also possible to infer and recover the user's input.

Chapter 2

Finger Tap Positioning and Speaker Activity Detection in Android Smartphones Using Internal Barometer

2.1 Introduction

The substantial increase of smartphone usage in our daily life, alongside them equipped with dozens of accurate sensors, opens a new window for side-channel attacks and information leaks. Side-Channel attacks are posing a severe threat to the user's security and privacy these days.

Many smartphone's sensors require user's permission to access, while others do not need any permission as their information is considered safe, and not a source of information leakage. In this chapter, we investigate one of these widely used "safe" sensors, namely the barometric pressure sensor. To this end, we design apps that collect low-rate barometric samples from the sensor, and then use these samples to extract sensitive user's information. In our first experience, we use the barometric pressure samples to detect and position user's finger taps on the touch screen in an attempt to discover user's pin entered through a custom numpad. In our second experience, we use the samples to detect the smartphone's earpiece speaker's activity, e.g. whether the speaker is playing a ringtone or not. We conclude that both these detections are possible.

2.2 Observations

As mentioned earlier, in devices with ingress protection, the internal pressure equalizes with the external pressure over a short period of time. This makes it possible to observe changes to the internal pressure using low-rate samples from the built-in barometer.

Figure 2.1 shows the effect of a 12 Hz sinusoid tone played by our IP68 rated testing device's speaker[1] on the internal barometric pressure sampled at 25 Hz[2] by the barometer. Notice that the barometer's sample signal roughly follows the sinusoid signal recorded by the device's microphone. This primary observation we made suggested that it may be possible to detect earpiece speaker activity using the device's internal barometer.

Figure 2.2 shows the effect of the same signal played by our non-IP rated testing device's speaker[3] on the internal barometric pressure sampled by the barometer. Notice that the barometer's sample signal does not follow any pattern of the actual sinusoid signal, and does not show any resemblance to the actual data. This makes sense as the internal pressure of our non IP rated device equalizes to the outside barometric pressure so fast that the low-rate barometric pressure samples are not able to capture the change in the internal pressure.

Figure 2.3 shows the impact of a touch screen finger tap on the internal pressure of our testing device. Note that a tap slightly flexes the screen inwards,

[1]Samsung Galaxy S10 Plus
[2]Maximum sample rate is limited by the Android API to 25 Hz
[3]Samsung Galaxy S6 Edge

causing the device's internal volume to slightly decrease. Since the device is nearly airtight, the gas inside the device cannot escape immediately through the vents. Consequently, the gas inside the device is compressed, causing the internal barometric pressure to increase. When the user removes the finger of the screen, the screen returns to its normal position and causes a short vacuum, this time causing the internal pressure to decrease.

Figures 2.4 and 2.5 show the average[4] impact of a finger tap on two different positions on the touch screen (number 2 and number 6 in the custom numpad). The variance for the peak and valley of the shown signals in the case of number 2 are 0.22 and 0.06, and for number 6 are 0.08 and 0.04, respectively. The difference between these two depicted impact signals suggested that it may be possible to detect the positions of finger taps on the touch screen. The main reason for the difference between these two impact signals is that taps on different positions of the touch screen can cause different amount of screen flex.

[4]Average of all the available samples, 124 samples for each number

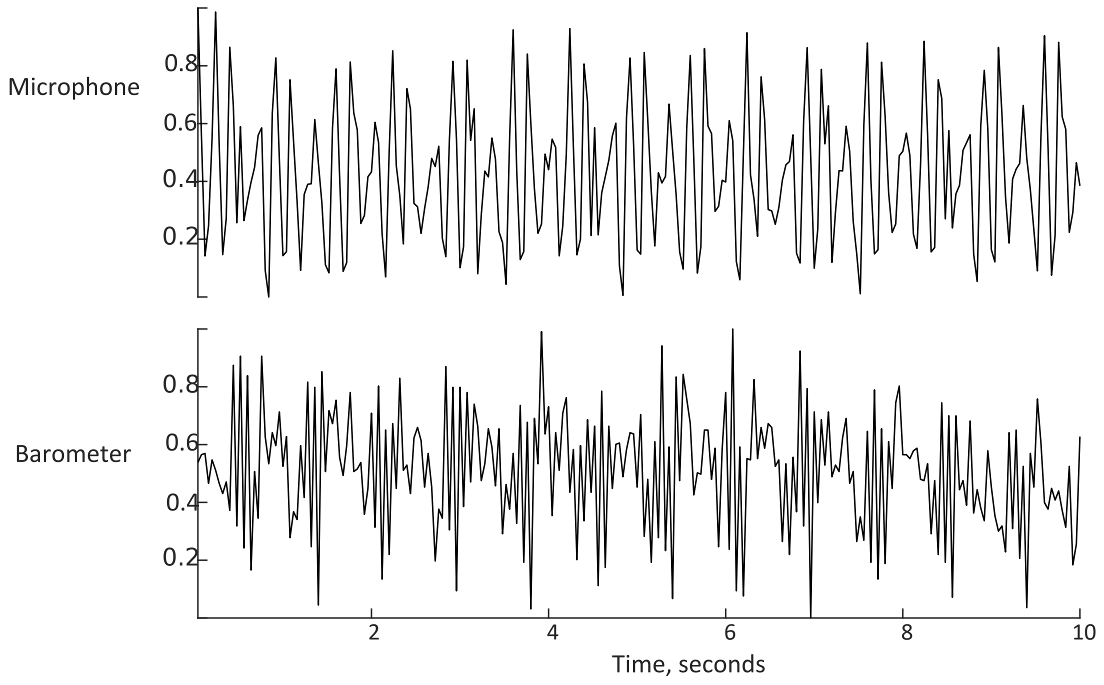

Figure 2.1: Signals recorded by our IP68 testing device's barometer, and the device's microphone when a 12 Hz sinusoid tone was played on the device's earpiece speaker.

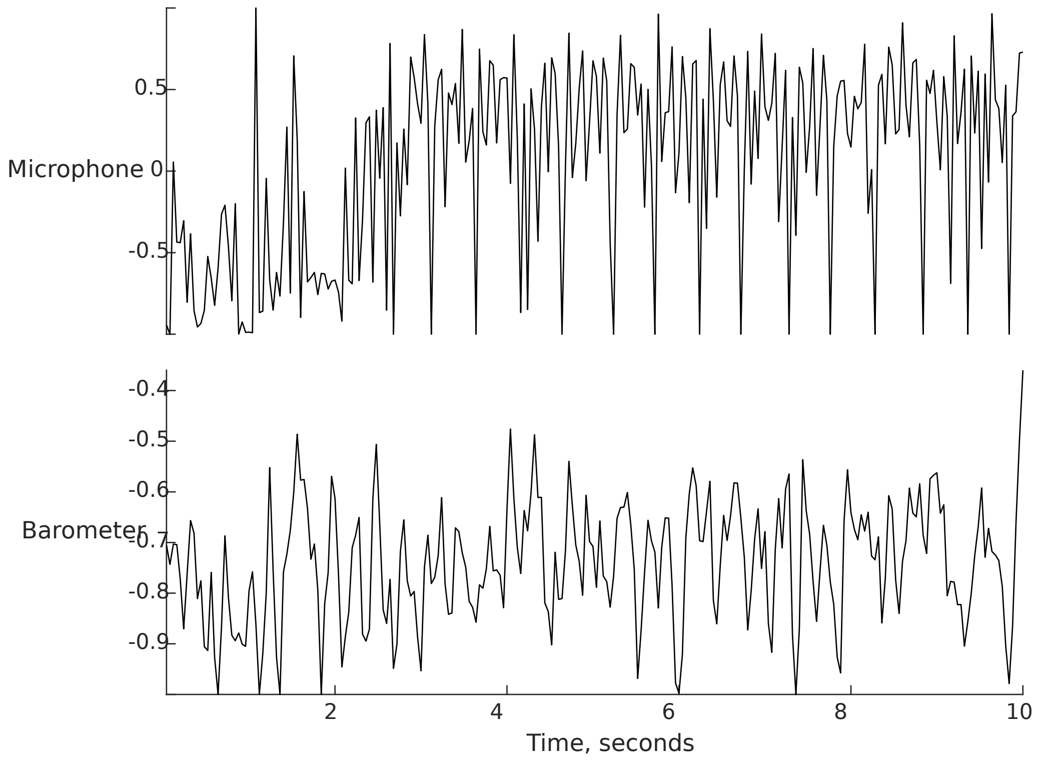

Figure 2.2: Signals recorded by our non IP rated device's barometer, and its microphone when a 12 Hz sinusoid tone was played on the device's earpiece speaker.

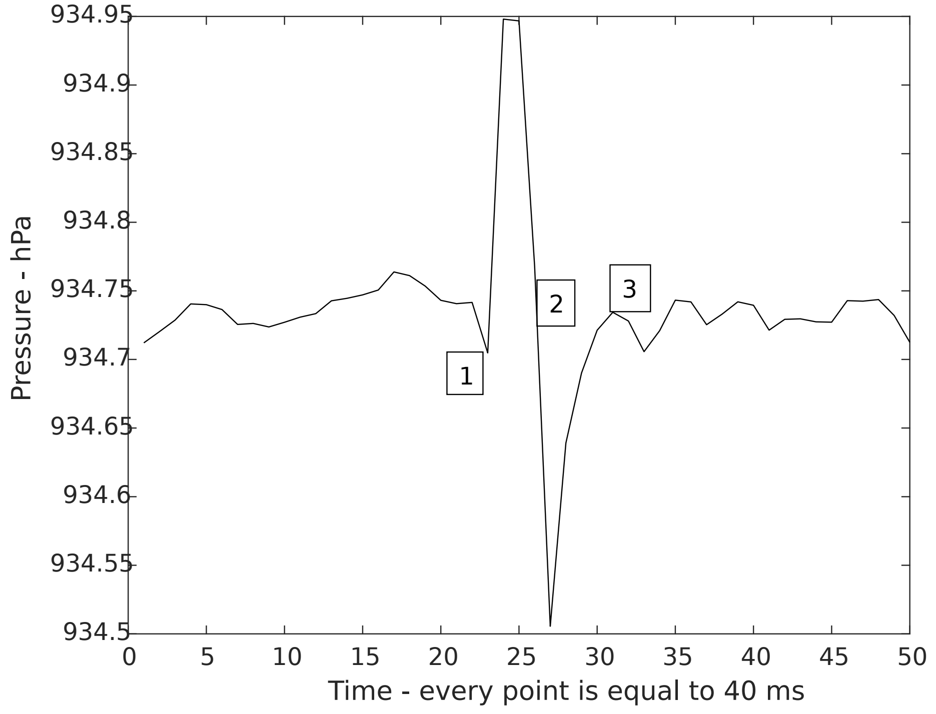

Figure 2.3: The impact of a finger tap on the internal pressure captured by the device's barometer. (1) internal pressure increases as the result of finger tap, (2) a decrease in the pressure after the finger tap (3) pressure equalization.

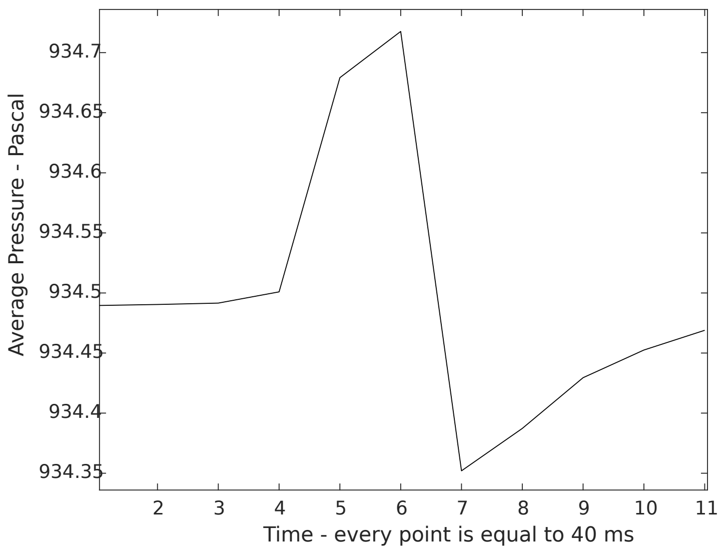

Figure 2.4: The average impact of a finger tap as the result of tapping on Number 6 of our custom numpad. As shown in Figure 2.7, this number is located on the right side of the screen.

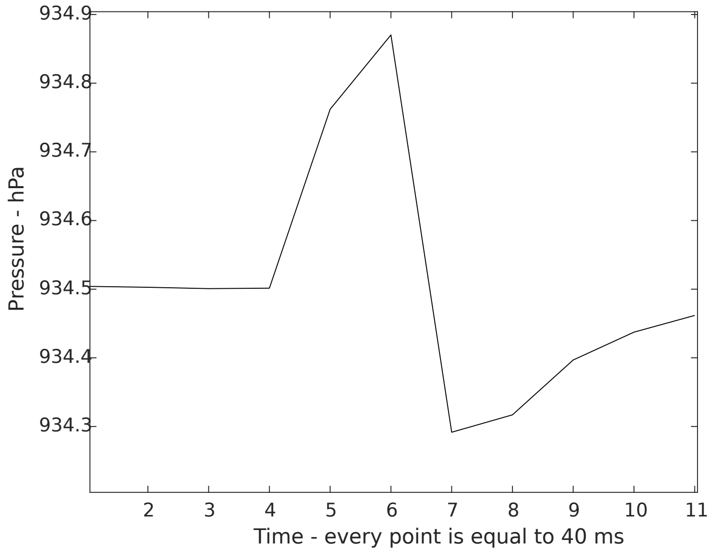

Figure 2.5: The average signal of a finger tap on Number 2 of our custom numpad. As shown in Figure 2.7, this number is located around the center of the screen.

2.3 Experiments

2.3.1 Data collection

We developed two custom Android apps for data collection: SpeakerSpy and TouchSpy. Using SpeakerSpy (Figure 2.6), we read and recorded barometric pressure samples during earpiece speaker activity (i.e., when a sound is played) and during inactivity (i.e., when the speaker is silent). A data record was labeled *active* if the barometer samples were taken when the earpiece speaker was playing a sound; otherwise, it was labeled *inactive*. We collected in total 450 records, half of which were labeled as active and the other half as inactive.

We used the second app, TouchSpy, to read and record barometric pressure samples during a screen finger tap. Each record spans an interval of two seconds, starting one second prior to the tap. The barometer sampling rate is 25 Hz; thus each record consists of 50 samples.

To collect real data using TouchSpy, we employed three participants. Each participant was asked to tap (as they normally do) the screen of a stationary phone placed on a desk. In one setting, the participants were asked to tap a randomly selected position on the screen. In the second setting, the participants were asked to tap a random number on a custom 3 by 3 pin entry numpad shown in Figure 2.7. For each setting, we collected 1116 records per participant. The records collected in the first setting were used to see how accurate we can detect the occurrence of a tap, while the ones collected in the second setting were used to see if we can guess the position of a tap.

Normalizing. We treated each record as a time series, and normalized it to reduce/eliminate the impact of the measurement unit, signal power and gradual changes in atmospheric pressure in the course of data gathering. To this end, we used a simple normalization called standardizing, in which each data point x in the time series is replaced by its z-score $z = \frac{x-\bar{x}}{S}$, where $\bar{x}$ and S are the mean and standard deviation of the time series, respectively.

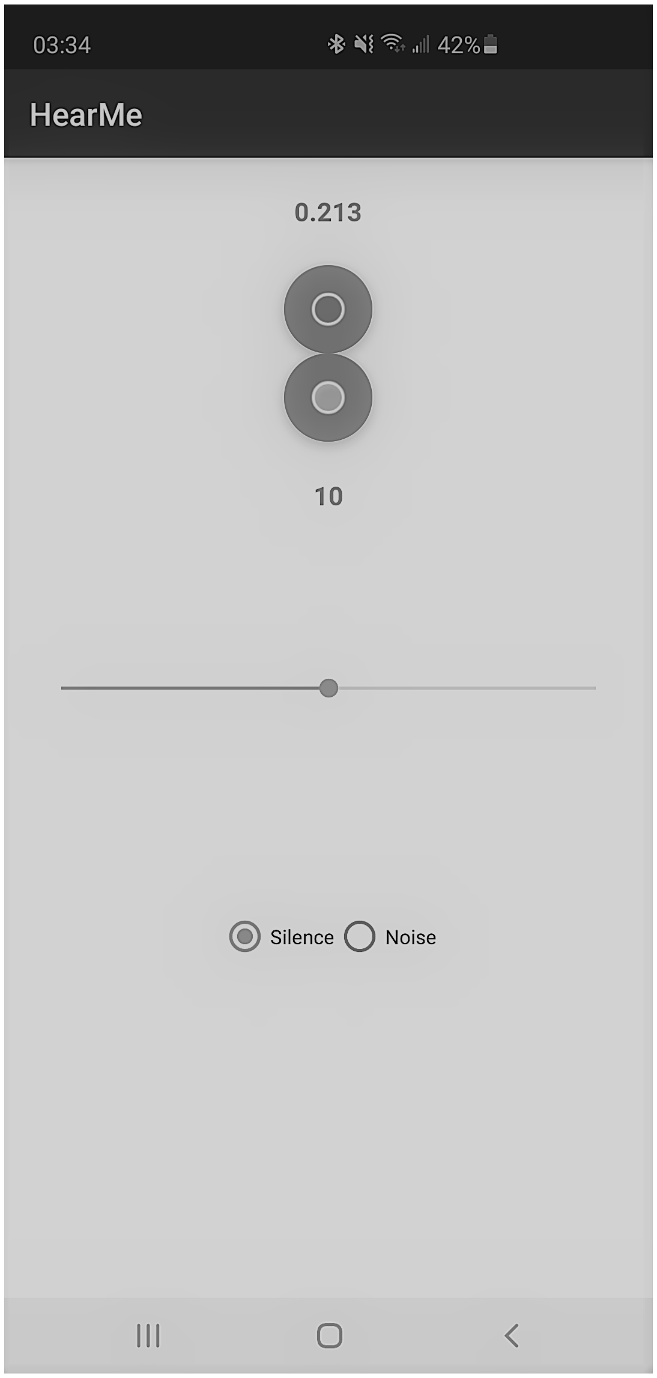

Figure 2.6: Automated UI for logging the barometric pressure alongside the microphone data in the SpeakerSpy app.

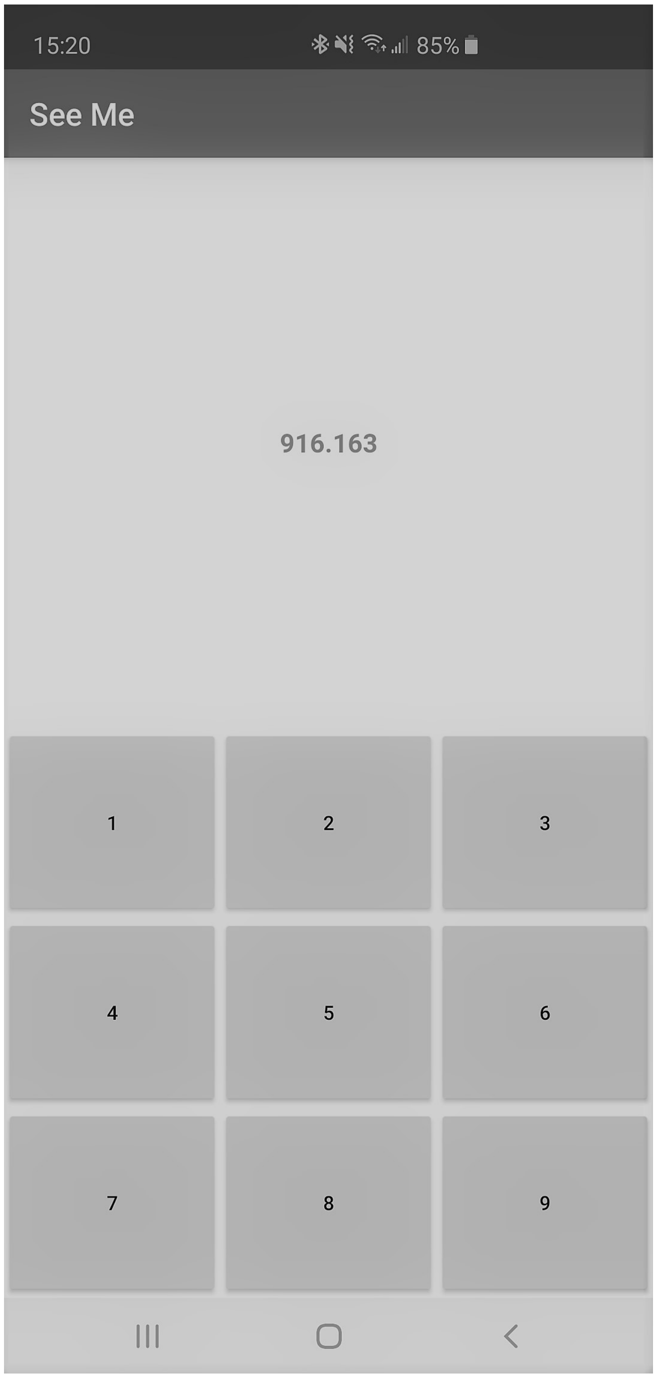

Figure 2.7: Our custom pin entry numpad in the TouchSpy app.

2.3.2 Detection accuracy results

We used SVM as our classification method, and 5-fold cross-validations to evaluate the performance of SVM. Each reported accuracy performance in this section is the average of the results of ten different 5-fold cross-validations.

Earpiece speaker activity detection. Barometer-based speaker activity detection (B-SAD) is a binary classification problem: it uses the smartphone's barometer to predict whether the smartphone's speaker is playing a given sound or is silent. A malicious app with B-SAD capability can steal user's sensitive information. For example, it can set the sound to, say, the device's default ringtone in an attempt to detect the onset of received phone calls.

To evaluate the performance of SVM in solving B-SAD, we used two type of sounds. The first one was a simple sinusoid tone of 12 Hz (nearly half the barometer's sampling rate of 25 Hz). For the second sound we used a popular Android ring tone (Over The Horizon, Samsung Phones). In our experiment we achieved an overall detection accuracy of 96% for the 12 Hz sinusoid tone and 95% for the default Samsung ringtone[5].

Detecting taps. Quinn [30] recently showed that, in devices with ingress protection, forces on touch screen can be detected using the device's internal barometer. To this end, a robotic arm was programmed to perform five second touch contacts with three different force levels (1, 3, and 5N). In our work, however, we explore the possibility of detecting taps (rather than robotic arm pressures) and their positions.

Unlike the 5-second robotic arm touch contacts in [30], taps make a quick force on the touch screen (fraction of a second). As stated in [30] if a user's input force changes too quickly, it may not be captured by the barometer. In addition, based on the study in [36], finger taps typically produce less force than those applied in Quinn's experiment. Therefore it is not clear if finger

[5]Over The Horizon, extracted from Samsung Galaxy S10 Plus

taps are detectable at all. Nevertheless, our SVM classifier achieves an accuracy of 100% in detecting occurrence of taps. Next, we show that the classifier can reveal information about the position of taps too.

Detecting the position of a tap. Our custom numpad (Figure 2.7) uses exclusively the lower half of the screen, as this is the default location of touch keyboards and numpads on most smartphones. The numpad divides the lower half into nine different regions/keys. Each region/key has different degrees of freedom. The difference in degrees of freedom leads to different amounts of screen flex and recovery as the result of a tap. Ultimately, this makes it possible to profile each region/key, hence detect the number tapped.

Our accuracy results are summarized in Table 2.1. The number in row i and column j of the table is the probability of predicting i when the actual number tapped was j. For instance, when user presses 5 our SVM classifier detects it as 2, 3, 5, 6 and 8 with probabilities 24%, 8%, 41%, 17% and 10%, respectively. As shown in Table 2.1, probabilities of correct identification (i.e., numbers on the main diagonal of the table) range from 18% to 69% with the average of 41%. One may compare this to the probability of correct identification in random guess, which is about 11%.

The above experiment was conducted in a room environment with inconsistent/unreliable air pressure. We expect our prediction results to somewhat improve when the environment's air pressure is steady. To evaluate this claim, we repeated the above experiment in a commercial flight, which, as we observed, had a more consistent air pressure in the cabin, compared to a room with air conditioning. The results of this experiment are shown in Table 2.2. The average correct identification in Table 2.2 is 44%, a slight increase to that in Table 2.1.

#	#1	#2	#3	#4	#5	#6	#7	#8	#9
#1	**18**			**4**			**27**	**31**	**16**
#2		**42**	**45**		**24**	**6**			
#3		**39**	**40**		**8**	**6**			
#4	**10**			**69**		**13**	**4**	**2**	
#5		**11**	**15**	**7**	**41**	**25**			
#6	**3**	**8**		**16**	**17**	**50**			
#7	**28**						**33**	**27**	**9**
#8	**31**			**4**	**10**		**26**	**19**	**18**
#9	**10**						**10**	**21**	**57**

Table 2.1: The number in row i and column j of the table is the probability of predicting i when the actual number tapped was j. The test was conducted in a room condition.

#	#1	#2	#3	#4	#5	#6	#7	#8	#9
#1	**23**			**8**			**26**	**25**	**18**
#2		**51**	**37**		**10**	**7**			
#3		**30**	**44**		**21**	**7**			
#4	**6**			**60**		**16**	**11**	**4**	
#5		**14**	**13**	**12**	**40**	**22**			
#6		**5**	**6**	**12**	**29**	**48**	**3**		
#7	**33**			**2**			**37**	**25**	**10**
#8	**29**			**6**			**18**	**27**	**20**
#9	**9**						**5**	**19**	**52**

Table 2.2: The number in row i and column j of the table is the probability of predicting i when the actual number tapped was j. The test was conducted in a commercial airplane.

2.4 Attack Scenario

Traditionally, to conduct a side-channel attack such as the one by Van Eck [6], an adversary requires to have physical access to the device or be in the close vicinity of the device. The trend, however, has shifted for smartphones. Although the work that we have presented in this thesis is taking advantage of the device's physical properties, an advarsary can execute our side-channel attacks entirely remotely. To do so, the adversary has to take the following steps to execute a successful touch screen tap inference or B-SAD on the victims' phone:

1. **Embed a malicious code into an Android application package (APK)**. Since the barometer sensor's API is not accessible through browsers [37], an attacker is left with two options to access the API. The first option is to build a completely new app (e.g., a weather forcast app) from scratch, and embed the malicious code (which gathers barometer's samples, and performs inference) inside it. The second option is to inject the malicious code into a popular and well-known app such as a popular game. There are pros and cons to each method. Making an app from scratch gives the attacker the ability to publish the malicious app silently via Google Play Store. Also, it gives the attacker the chance to request some permissions (like Internet access) that may initially look legit. Such accesses can come in handy. For example, the Internet access can be used to transfer the inferred user's information to the attacker. By injecting the malicious code into a well-known and popular app, on the other hand, the attacker can hit a larger population. This option, however, does not allow the attacker to put the modified app on a trusted source like Google Play Store. This makes it harder for the app to find its way to the users' phones.

2. **Perform data gathering and training**. Data gathering does not require any permissions from the user but needs the user to interact with

the malicious app that has been disguised as we described. Putting an invisible overlay layout on top of the malicious app makes it possible to gather the necessary data for training our model for the user's typing habits (the exact force of user's finger taps) and phone screen size. In our tests, about 50 samples per region (key) was the minimum for our classification to be viable. As for B-SAD, embedded sound queues in the malicious app are enough to train the model. Also, to perform an accurate B-SAD, a minimum of 25 samples of each state (active/silence) is necessary. Finally, attackers can use the gathered data to train an SVM classifier either locally at the victims' smartphones, or remotely at the at their own end.

3. **Perform inference and extraction**. At this point, the malicious app continuously logs the barometer's samples in the background. At the same time, it feeds the collected samples to its SVM classifier to detect taps and/or earpiece speaker activity. The results of these detections can be stored locally, or sent to the attacker over the Internet.

2.5 Limitations

Our results confirm the possibility of detecting earpiece speaker activity and touch screen taps using barometric air pressure samples. These results rely on relative pressure rather than absolute pressure values of the barometer; thus they are applicable to different locations/altitude levels, and weather conditions.

However, there are certain limitations in collecting side-channel information from screen touch contacts and earpiece speaker activities. First, the device must have some level of ingress protection. It is because, otherwise, changes in internal pressure (due to, for example, finger taps) would quickly equalize the external atmosphere.

The second major limitation is related to the low sampling rate of the barometer sensor. The STMicro LPS22HH sensor datasheet shows that it is capable of sampling at 200 Hz. However, it appears that the Android sensor API is limiting the barometer sample rate to 25 Hz[6] [19]. In our experiments, the low sampling rate of 25 Hz was not deterrent in detecting earpiece speaker activity and detecting finger touch contacts. There are, however, certain tasks that are impossible to perform at low sampling rates. For instance, by the Nyquist–Shannon sampling theorem, it is impossible to reconstruct the speaker's sound signal fully if the signal has frequencies higher than half the sampling rate of the barometer.

The low sampling rate of barometer can be preventive in carrying certain tasks related to screen touch detection as well. For example, in our experiment, we need to have at least three sample points in our signal to register a finger tap, and wait for at least two sampling cycles for air pressures to equalize. At the sampling rate of 25 Hz, these five sampling cycles translate to a total of 200 milliseconds, which would limit the continuous detection of finger taps to five taps per second.

Finally, the flow rate of the barometric vent can also become a limiting factor. As stated in [30], if a user's input force changes either too quickly or too slowly, it may not be captured by the barometer.

2.6 Security Implications

Reading barometric pressure low-rate samples is considered harmless. Because of this, modern mobile platforms like Android allow apps to access the device's barometer sensor without user's permission or notification. Therefore, a malicious app can readily bypass the user's permission and attention. In addition, as shown in [38], sensor-based side-channel attacks can bypass strong separation

[6]The sampling rate of 25 Hz is achievable through the SensorManager.SENSOR DELAY FASTEST in Android SDK

mechanisms like Samsung KNOX, which tries to provide a secure environment to protect corporate data on smartphones.

Finally, due to low power usage and low frequency of the barometer sensors, accessing the barometric pressure samples may not be detected as, for example, a power virus. We ran a background service that continuously reads barometric pressure samples for over 40 hours. This service was never detected as a suspicious activity by McAfee, Avast, AVG, Bitdefender and Norton antivirus.

2.7 Defenses

A first line of defense is to use the device's permission system to ask users for permission to access the device's barometer. This gives users the chance to deny permission to apps whose motivation behind requesting the access is not clear. For instance, users can deny barometer access to a "flashlight app" as such access is clearly outside the scope of the app's functionality.

Another defense is to use third-party apps such as [39] that enforce security policies. These apps can pause sensor readings (or even kill) background processes/services when an app with sensitive input (e.g. a banking app) is running. In addition, malware-analysis apps such as [12] and [40] can be extended and used to check for malicious sensor accesses.

Finally, sensitive apps can implement their own custom security solution. For example, a PIN pad can rearrange its buttons prior to every sensitive input [41].

Chapter 3

Conclusion and future work

Microphone and touch screen inputs expose sensitive information. Because of this, microphone's access requires user's permission, and touch screen is only accessible to the app currently running in the foreground. On the other hand, any app (even those running in the background) can access the barometer's samples without any permission or notification. This is alarming as our results show that low-rate barometric pressure samples reveal information about earpiece speaker activity. In addition, the pressure samples give information about finger taps, and, to some extent, their positions.

Our work is a first feasibility study, and there are many ways to extend and improve our results. First, we only used SVM as our classifying method. More advanced classifiers and techniques can improve our reported accuracy results. In addition, barometric pressure samples may be combined with data samples from other sensors such as ambient-light or motion sensors to improve accuracy.

In detecting finger taps and their positions, we used a small number of participants to collect data. Further study is needed to show if similar results is observed with more number of participants with, say, various tapping habits. Also, we used a single type of smartphone in our experiments. However, we expect similar results to be observed in many other types of smartphones with ingress protection.

With regards to earpiece speaker activity, it is interesting to see how many different classes of activities we can distinguish using low-rate barometric pressure samples. Our initial study, not reported to this work, indicate that there is a possibility of distinguishing different sounds.

Finally, it would be interesting to see if external sounds can be distinguished from, say, silence using barometric pressure samples.

Bibliography

[1] Ash Turner. 1 billion more phones than people in the world! bankmycell, July 2020. URL `https://www.bankmycell.com/blog/how-many-phones-are-in-the-world`. Accessed on 09.07.2020.

[2] Haibo Ye, Li Sheng, Tao Gu, and Zhiqiu Huang. Seloc: Collect your location data using only a barometer sensor. *IEEE Access*, 7:88705–88717, 2019.

[3] R. Spreitzer, V. Moonsamy, T. Korak, and S. Mangard. Systematic classification of side-channel attacks: A case study for mobile devices. *IEEE Communications Surveys Tutorials*, 20(1):465–488, 2018.

[4] NSA TEMPEST. A signal problem. *Cryptologic Spectrum*, 2(3), 1972.

[5] Cassi Goodman. An introduction to tempest. *SANS Institute*, 18, 2001.

[6] Wim Van Eck. Electromagnetic radiation from video display units: An eavesdropping risk? *Computers & Security*, 4(4):269–286, 1985.

[7] Michael Backes, Markus Dürmuth, Sebastian Gerling, Manfred Pinkal, and Caroline Sporleder. Acoustic side-channel attacks on printers. page 20, 2010.

[8] Adam J Aviv, Katherine L Gibson, Evan Mossop, Matt Blaze, and Jonathan M Smith. Smudge attacks on smartphone touch screens. *Woot*, 10:1–7, 2010.

[9] Nan Zhang, Kan Yuan, Muhammad Naveed, Xiaoyong Zhou, and XiaoFeng Wang. Leave me alone: App-level protection against runtime information gathering on android. In *2015 IEEE Symposium on Security and Privacy*, pages 915–930. IEEE, 2015.

[10] Ryan Stevens, Jonathan Ganz, Vladimir Filkov, Premkumar Devanbu, and Hao Chen. Asking for (and about) permissions used by android apps. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 31–40. IEEE, 2013.

[11] Application security : Android open source project, 2020. URL `https://source.android.com/security/overview/app-security`. Accessed on 12.04.2020.

[12] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 627–638, 2011.

[13] App permissions best practices : Android developers. 2020. URL `https://developer.android.com/training/permissions/usage-notes`. Accessed on 04.04.2020.

[14] Sensors overview : Android developers. URL `https://developer.android.com/guide/topics/sensors/sensors_overview`. Accessed on 09.07.2020.

[15] International Electrotechnical Commission et al. *Degrees of protection provided by enclosures (IP Code)(IEC 60529: 1989+ A1: 1999+ A2: 2013)*. IEC, 2013.

[16] Quanqing Yu, Rui Xiong, Chuan Li, and Michael G Pecht. Water-resistant smartphone technologies. *IEEE Access*, 7:42757–42773, 2019.

[17] Jieying Zhang, Ezzaldeen Edwan, Junchuan Zhou, Wennan Chai, and Otmar Loffeld. Performance investigation of barometer aided gps/mems-imu integration. In *Proceedings of the 2012 IEEE/ION Position, Location and Navigation Symposium*, pages 598–604. IEEE, 2012.

[18] Kartik Muralidharan, Azeem Javed Khan, Archan Misra, Rajesh Krishna Balan, and Sharad Agarwal. Barometric phone sensors: More hype than hope! In *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications*, pages 1–6, 2014.

[19] Kartik Sankaran, Minhui Zhu, Xiang Fa Guo, Akkihebbal L Ananda, Mun Choon Chan, and Li-Shiuan Peh. Using mobile phone barometer for low-power transportation context detection. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, pages 191–205, 2014.

[20] Lps22hh. URL `https://www.st.com/en/mems-and-sensors/lps22hh.html`.

[21] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Trusted execution environment: what it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 57–64. IEEE, 2015.

[22] Tim Cooijmans, Joeri de Ruiter, and Erik Poll. Analysis of secure key storage solutions on android. In *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, pages 11–20, 2014.

[23] Brian McGillion, Tanel Dettenborn, Thomas Nyman, and N Asokan. Open-tee–an open virtual trusted execution environment. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 400–407. IEEE, 2015.

[24] Uri Kanonov and Avishai Wool. Secure containers in android: the samsung

knox case study. In *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*, pages 3–12, 2016.

[25] Mishaal Rahman. How to find the android version distribution statistics in android studio, Apr 2020. URL `https://www.xda-developers.com/android-version-distribution-statistics-android-studio/`. Accessed on 12.06.2020.

[26] Chih-Chung Chang and Chih-Jen Lin. URL `https://www.csie.ntu.edu.tw/~cjlin/libsvm/`. Accessed on 09.07.2020.

[27] Jake Lever, Martin Krzywinski, and Naomi Altman. Points of significance: model selection and overfitting, 2016.

[28] Jason Brownlee. A gentle introduction to k-fold cross-validation, Aug 2019. URL `https://machinelearningmastery.com/k-fold-cross-validation/`. Accessed on 20.07.2020.

[29] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[30] Philip Quinn. Estimating touch force with barometric pressure sensors. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–7, 2019.

[31] Salvatore Vanini and Silvia Giordano. Adaptive context-agnostic floor transition detection on smart mobile devices. In *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 2–7. IEEE, 2013.

[32] Bo-Jhang Ho, Paul Martin, Prashanth Swaminathan, and Mani Srivastava. From pressure to path: Barometer-based vehicle tracking. In *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, pages 65–74, 2015.

[33] Muchen Wu, Parth H Pathak, and Prasant Mohapatra. Monitoring building door events using barometer sensor in smartphones. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 319–323, 2015.

[34] Liang Cai and Hao Chen. Touchlogger: Inferring keystrokes on touch screen from smartphone motion. *HotSec*, 11(2011):9, 2011.

[35] Raphael Spreitzer. Pin skimming: Exploiting the ambient-light sensor in mobile devices. In *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, pages 51–62, 2014.

[36] Faisal Taher, Jason Alexander, John Hardy, and Eduardo Velloso. An empirical characterization of touch-gesture input-force on mobile devices. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*, pages 195–204, 2014.

[37] Mozilla. Sensor apis. URL `https://developer.mozilla.org/en-US/docs/Web/API/Sensor_APIs`. Accessed on 17.07.2020.

[38] Laurent Simon and Ross Anderson. Pin skimmer: Inferring pins through the camera and microphone. In *Proceedings of the Third ACM workshop on Security and privacy in smartphones & mobile devices*, pages 67–78, 2013.

[39] Michael Backes, Sebastian Gerling, Christian Hammer, Matteo Maffei, and Philipp von Styp-Rekowsky. Appguard–enforcing user requirements on android apps. In *International Conference on TOOLS and Algorithms for the Construction and Analysis of Systems*, pages 543–548. Springer, 2013.

[40] Clint Gibler, Jonathan Crussell, Jeremy Erickson, and Hao Chen. Androidleaks: automatically detecting potential privacy leaks in android applications on a large scale. In *International Conference on Trust and Trustworthy Computing*, pages 291–307. Springer, 2012.

[41] Emmanuel Owusu, Jun Han, Sauvik Das, Adrian Perrig, and Joy Zhang. Accessory: password inference using accelerometers on smartphones. In *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*, pages 1–6, 2012.