MOTION-BASED SIDE-CHANNEL ATTACK ON MOBILE KEYSTROKES

A PREPRINT

Jessy Lin
lnj@mit.edu

Jason Seibel
jseibel@mit.edu

May 19, 2019

1 Introduction

Modern smartphones are equipped with increasingly sensitive sensors that provide precise data about device motion. Unlike sound, video, and location data that pose obvious security and privacy risks, permissions for local device motion from the accelerometer and gyroscope are often overlooked. These sensors are often used for harmless applications like games (thus avoiding the user's suspicion) and on some devices, can be accessed from an app even when it is running in the background. Data from these sensors are thus prime targets for side-channel attacks. Past work has demonstrated that it is feasible to infer keystrokes from accelerometer and gyroscope data alone [1, 2, 3].

The goal of this project is to implement this side-channel attack to infer keystrokes on a smartphone device purely from accelerometer and gyroscope data. Furthermore, we investigate the permissions that may restrict the extent of these attacks on modern devices.

2 Threat Model

In our threat model, we assume that the target has downloaded an app controlled by the attacker on to his/her mobile device. We assume that the app can continuously read accelerometer and gyroscope data while the user is using other apps with potentially sensitive information (e.g. bank PINs, passwords), although we examine the practicality of this assumption in Section 3.3. We do not assume that the attacker has access to the device.

We envision an attack flow as follows:

1. The user downloads a typical gaming app (a disguise). The permissions required for this app depend on the smartphone operating system and the length and frequency of background data collection. These permissions limitations are described more in section 2.4.

2. We collect keystrokes and press data as the user uses our app and train a user-specific model (or fine-tune a general prediction model trained across users).

3. We prompt the user for authentication or payment on some third-party app (e.g. "Login with Facebook," "Pay with Venmo"). Normally this does not pose a security risk, since the credentials are handled by the third-party app and not accessible to the requesting application. However, because sensor data is still accessible in the background, we can infer the keystrokes typed into the third-party app. An alternative to this approach (but only available on certain operating systems as described in section 2.4), would be to just let the user play the game normally, and silently include a background process that continuously records and sends sensor data to our server.

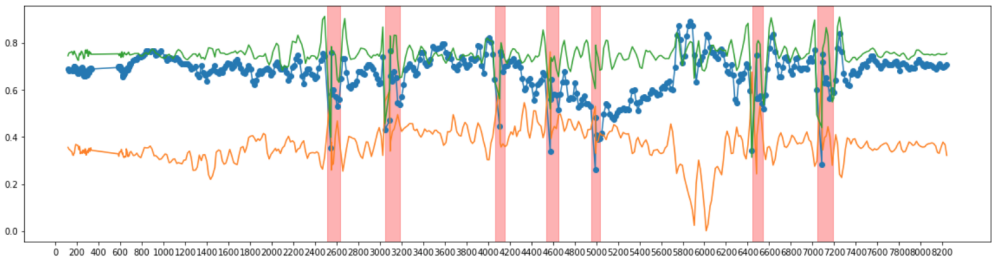


Figure 1: A segment of accelerometer (x,y,z) data. The samples outputted by the device are by dots on the blue curve and later interpolated into 10ms samples. We send both press and release events to train the model; these touch events are highlighted in red and largely lie within 200ms windows.

3 Design

3.1 Data Collection

We set up an iOS and Android mobile app (using React Native) that sends timestamped accelerometer, gyroscope, and ground-truth touch events with (x,y) location to a server through a WebSocket connection. We collected over an hour of touch data in a variety of conditions on Google Pixel 2 XL and iPhone X smartphones. The final models were trained with 15 minutes of one-handed touches at the rate one would type normally, e.g. as if unlocking the device. The data includes a diverse range of motion (swinging the device while walking, turning the phone over, etc.) when the device is not being touched, to account for sensor readings that might result from everyday device use.

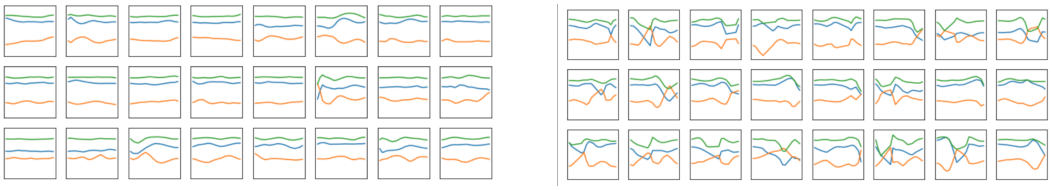
Since sensor readings are emitted at sporadic intervals by the device, we pre-process the data by interpolating to regular 10ms samples and grouping the samples into 200ms windows (window size determined by inspecting the data, see Figure 1 for a sample segment of accelerometer data). We normalize both sensor readings and touch locations to the range $[0, 1]$.

3.2 Prediction Model

We train one machine learning model to determine whether a touch event occurs in a window. For each window with a detected touch event, we train a separate model to predict the touch location. Both models are convolutional neural networks (see the code repo for architectural details). We spent little time tuning the architecture or hyperparameters of the model, so there are potentially performance improvements to be gained from more careful tuning.

To verify that this prediction task is feasible, we visually compare windows with and without touches (Figure 2).

At test time, we predict in real-time by sliding a 200ms window over the data with a stride of 100ms. For each window, we predict whether there is a touch, and if there is, we predict the touch location.



(a) Windows where no touches have occurred.

(b) Windows where a touch has occurred.

Figure 2: Accelerometer (x,y,z) data for touches and non-touch windows.

3.3 Permissions

To make this attack successful in practice, the malicious app needs to continuously collect data while the user is on other applications, e.g. while the user is typing the PIN into his or her banking application. The difficulty is doing so invisibly without requiring permissions that will arouse the user's suspicions. Unfortunately, handling permissions is heavily dependent on the smartphone operating system and version as both iOS and Android have introduced tighter

control over what apps can do in the background in their most recent operating system (OS) updates. By default, on the most recent versions of iOS and Android, sensor readings are paused when the user is not focused on the application in the foreground. Here, we investigate the extent of these restrictions on older versions of each OS, and present some approaches that could be feasible attacks.

Android In Android 9.0, accelerometers and gyroscopes are not allowed to be accessed by background services at all [5]. In Android 8.0, new restrictions limit the amount of time a background service can run after the app responsible for the service has left the foreground (i.e. once the user is not actively using the app on their phone's screen). This limit is "several minutes" and can only be marginally extended in certain cases [4]. Thus, if someone wanted to target Android version 8, they could make an app that redirects the user to a sensitive app like a banking app (e.g. for in-app purchases), and exploits the several minutes of background service time to infer useful data.

If both the limited background run time and sensor data access prove too restrictive to collect enough data, the remaining option is a foreground service. A foreground service is exempt from these limitations but requires a persistent notification item to be visible to the user at all times while the service is running. Apps that could justify needing a foreground service include battery monitors, navigation apps, etc. At the same time, the app must still collect ground-truth touch locations from the user interacting directly with the app for an extended time (to train a user-specific touch location prediction model). One potential difficulty is constructing a disguise that has a reasonable justification for both types of interfaces with the user.

Alternatively, Android 7 and earlier versions relax these restrictions. Fortunately, Android has an extremely fragmented operating system version division amongst its users. Roughly 55 percent of Android users were using version 7 or older as of April 2019 and around 35% were on version 8 [6]. This means that a malicious app could potentially disguise itself by running hidden background services on older devices, and simply acting benignly on newer devices.

iOS In iOS, roughly 80 percent of users worldwide are using a version of iOS 12, so we would need to primarily focus on users of iOS 12 to make a wide reaching attack [7]. The latest iOS permissions makes it much harder for an attack like this to work. In order to continuously monitor accelerometer and gyroscope sensor data in the background, it seems necessary to request continuous location updates as a permission [8]. This is because iOS only allows certain types of apps to be allowed to run a "Long-Running" background task [8]. The App Store's review process focuses on finding discrepancies between the claimed application use and the functionality of the background service. Most likely, an attacker would need to make an app that could justify collecting continuous gyroscope and accelerometer data in the background; which narrows the scope of potential apps significantly. It might be unlikely to convince a significant group of users to download a new app that tracks their location continuously in the background since we would either need to be better than current navigation apps like Google and Apple Maps, or provide a different service that would still warrant users being comfortable with such extreme user data collection. Overall, this makes iOS a difficult target for this attack.

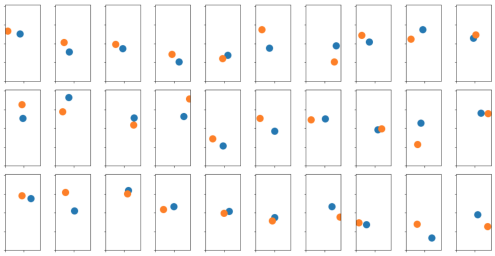
4 Results

With 4400 200ms windows (approx. 15 min of data), we detected touch events from sensor data on a held-out test set with an accuracy of **83%**. For the windows with predicted touch events, we achieved a mean-squared error of **0.06** between the predicted touch location and the true touch location. For an iPhone X (375 px by 812 px), this translates into a discrepancy of 92-199px. However, as shown in Figure 3, most of the predictions have a small mean-squared error, with a long tail of outliers with predicted locations that are far from the true locations. We also show a sample of predicted and true locations scaled to an iPhone X screen size.

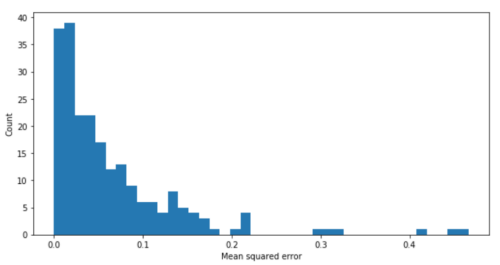
A video of the real-time predictions can be found here.

5 Conclusion

Overall, smartphone operating systems have responded well to the discovery of accelerometer and gyroscope based side channel attacks. Implementing this attack would require aiming for users with older operating systems, which luckily, Android still has lots of. We were able to achieve an accuracy that was high enough that, if downloaded by a large group of users, could reveal sensitive data like user's PINs and might even have the potential to start revealing some simple password inputs under more ideal typing conditions. This means this threat is still an active concern and for particularly well disguised apps, can still be a major concern even in newer Android versions. But looking into the future, it seems that the prediction aspect of this attack will remain successful, but the the constant battle to make a well



(a) Predicted (orange) and true (blue) touch locations.



(b) Distribution of mean-squared error between predicted and true locations.

Figure 3: Results on a held-out test set.

disguised app that works on as many operating systems as possible will require constant adjustment as operating system designers try to combat this kind of attack.

Our code can be found on Github. The master branch currently contains the most up-to-date python server and machine learning code, and the ejectedtest2 branch contains a mobile app that can read and report this sensor data on certain target operating systems.

References

[1] Maryam Mehrnezhad, Ehsan Toreini, Siamak F. Shahandashti, Feng Hao Stealing PINs via mobile sensors: actual risk versus user perception

[2] Liang Cai, Hao Chen TouchLogger: Inferring Keystrokes On Touch Screen From Smartphone Motion

[3] David Berend, Bernhard Jungk, Shivam Bhasin There Goes Your PIN: Exploiting Smartphone Sensor Fusion Under Single and Cross User Setting

[4] Android 8 Developer Releases: Behavior Changes: all apps https://developer.android.com/about/versions/oreo/background

[5] Android 9 Developer Releases: Background Execution Limits https://developer.android.com/about/versions/pie/android-9.0-changes-all

[6] Statcounter Mobile & Tablet Android Version Market Share Worldwide Apr 2018- Apr 2019 http://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide

[7] Statcounter Mobile & Tablet iOS Version Market Share Worldwide Apr 2018- Apr 2019 http://gs.statcounter.com/ios-version-market-share/mobile-tablet/worldwide

[8] Apple App Programming Guide for iOS: Background Execution https://developer.apple.com/library/archive/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/BackgroundExecution/ BackgroundExecution.html