(Don't) Hold the Phone: Accelerometer Sensor-Based Side Channels on Smartphones

Anonymous Submission

Abstract

Modern smartphones are equipped with a plethora of sensors that enable a wide range of interactions with the device; however, these sensors can measure much more than the user's intentions within a single application. In this paper, we examine the implications of one sensor, the accelerometer, and its use as a side channel. We show that the accelerometer sensor is a sufficiently high-bandwidth side channel that can be used to learn (or greatly reduce the search space for) secure or confidential user input, such as the Android password pattern and PINs. We apply off-the-shelf machine learning techniques to accelerometer data collected while entering a set of 50 password patterns and 50 PINs and show that a pattern can be identified with an accuracy as high as 60%, and a PIN can be identified with an accuracy as high as 49%. There is also surprising consistency across users and devices, and an attacker can train a model based on one user's input to attack another user. Our results suggest that precautions should be taken to restrict access to sensor data during security-sensitive operations.

1 Introduction

Modern smartphones ship with a increasing range of sensors to measure the phone's environment. These sensors are used for a wide variety of tasks; for example, the gyroscopic and accelerometer sensor can measure the movement of the phone in space and are often used in gaming applications. Applications are generally granted access to these sensors without much concern; however, certain sensors, particularly the accelerometer, may be able to measure much more than just the user's intention within a single application.

Consider an application that runs in the background and has access to the accelerometer sensor, the key question is: *What can the background application learn about user input to the foreground application via the accelerometer readings?* In this paper, we answer this question and show that the accelerometer is sufficiently sensitive to learn private and confidential user input, and that the accelerometer is a surprisingly effective side channel that can leak information with remarkable fidelity.

The accelerometer sensor's capability as a side channel is a direct result of the new computer interaction layer promoted by smartphones. As compared to traditional computing platforms, smartphones are tactile, hand-held devices, and users provide input by physically touching and gesturing on the touchscreen. These actions implicitly shift and adjust the device in measurable ways that can be recorded using the on-board accelerometer sensor. Our results show that these slight adjustments and movements can correlate with the precise input provided, or reduce the search space necessary to determine the input.

Particularly, we focus on two secure input types that are representative of touchscreen input: four-digit PINs (point touching) and the Android password pattern (gesturing). In our experiments, using off-the-shell machine learning techniques, we show that accelerometer measurements can be used to identify the PIN or pattern that was entered. Using a set of 50 PINs and 50 patterns, we collected 3,600 samples and can identify the precise PIN or pattern entered with an accuracy of 60% for patterns and 49% for PINs. We also find that there is relative consistency in accelerometer readings across users and smartphone devices, and accelerometer measurements from one user can be used to identify input from another user. Further, we show that accelerometer measurements can be used to classify individual swipe motions or touch events – *e.g.*, swiping left or right, or touching the digit 9 – and that these smaller classifications can be combined

to form larger classifications about potentially unseen input not used in training.

These results suggest that the accelerometer sensor is highly sensitive, much more so than previously thought, and that a wide variety of user input (touching and gesturing) is susceptible to this side channel. This is supported by previous results on using on-board sensors to learn single touch events (*touchlogger* [4]) or short sequences of touch input input (*ACCessory* [15]). The combination of this work and the previous side channels, now constitutes the vast majority of user input, *i.e.*, swiping and touching, and thus the accelerometer sensor could potentially reveal nearly all user input. The security model for applications with access to sensitive sensors, like the accelerometer, should be carefully reconsidered in light of these and previous results, and we propose a new approach to smartphone security that can greatly reduce the threat of a side channel based on sensor readings.

To summarize, the contributions of this paper are as follows:

- We show that the accelerometer sensor can be employed as a high-bandwidth side channel that can leak a large amount of information, more than previously suggested, and that the accelerometer sensor can be used to greatly reduce the search space for secure input, such as PINs and password patterns.
- We show that accelerometer measurements are relatively consistent across users and smartphone devices, and that measurements from one phone or user can be used to identify input from another user or device.
- We propose architectural changes to the smartphone security models that can mitigate sensor based side channels.

2 Attack Scenario

In this paper, we consider an attacker who wishes to learn the secure input of smartphone users via an accelerometer side channel. An attacker may gain access to accelerometer data in a wide variety of ways – *e.g.*, the attacker finds a phone where an application has written accelerometer data to the sd-card. We consider a more active attacker who distributes a malicious smartphone application that can run in the background, has access to the accelerometer, and can communicate over the network. As an example of the kinds of input an attacker may be able to learn, we focus on the information that is leaked by two common input types, entering a PIN or Android password pattern that is used to lock the smartphone.

To this end, the malicious application is aware when the phone initially wakes and, thus, the smartphone will prompt a user for a PIN or password pattern while the malicious application is running in the background. The application then activates the accelerometer sensor, recording measurements for a short time period. We found that it takes 1.1 seconds to enter a pattern and 1.7 seconds to enter a PIN, on average, so the accelerometer does not need to be active for very long. The accelerometer measurements are eventually sent over the network to be analyzed offline.

The attacker's goal at this point is to develop a method for comparing the captured accelerometer data to a corpus of labeled accelerometer data[1]. That is, the attacker has at his/her disposal accelerometer data that he/she knows was collected when a particular PIN or pattern is entered. The problem of identifying the PIN or pattern that was entered is now reduced to a classic machine learning problem: Given previously label input, what is the label of the unknown input? In this scenario, the label is the PIN or pattern of the victim.

We consider two scenarios in our experiments for the attackers capabilities to make this comparison to the corpus at his/her disposal. In the first scenario, we assume that the attacker has a large corpus, and samples of the PIN or pattern he/she is trying to learn can be found in the corpus. In the second scenario, we assume that the attacker does not have samples in the corpus, or not enough to generate a strong model.

In our experiments, we model these two scenarios by first considering a sample set of 50 patterns and 50 PINs. Here the goal of the experiment is to measure how accurately a pattern and PIN can be identified based on previously seen input. In the second scenario, where the attacker does not have sufficient labeled data, the goal of the experiment is to measure the accuracy of a sequence predictor that tries to identify a pattern by making a sequence of smaller predictions (*e.g.*, a single swipe or digit press). We present more details of our machine learning setup in Section 5.

Of course, an important question is: What can an attacker do with the information learned? Clearly, if the attacker has learned a user's password pattern, it is only useful if the attacker gains physical access to the victim's phone at some later point because the Android password pattern is not a widely used security mechanism.

[1]The attacker could build such a corpus by distributing an application that requires users to enter patterns for other purposes, such as [7, 10, 16].

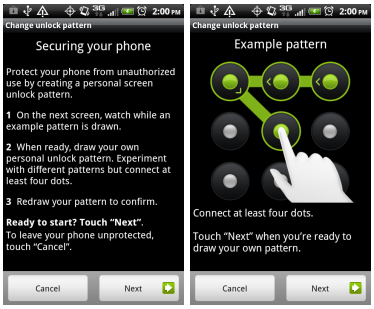

Figure 1: Password Pattern Instructions Provided on Android Smarpthones

Granted, this is a reasonable attack scenario. However, learning a user's smartphone unlock PIN may be applicable in other settings if the user reuses his/her PIN, such as an ATM pin or in online banking application.

More broadly, we focus on PINs and Android password patterns because they represent a larger set of user input on touchscreens that is composed of point touching and gesturing. Demonstrating an accelerometer side channel against these input types is an example of a broader family of sensitive touchscreen inputs that may be susceptible to this side channel.

3 Background

Before proceeding, we first provide background on the secure input types used in our experiments. Additionally, we provide background on the accelerometer sensor and the measurements it takes. In the following sections, we show how to use the accelerometer measurements to learn the PIN or pattern entered.

PINs Both Apple iOS and Android based smartphones support PINs as a screen lock mechanism. iOS'es primary screen lock interface uses a PIN, but Android provides two other options: a graphical password pattern (see below) or a pass-phrase consisting of both numbers and letters. A PIN consists of a sequence of four digits, 0-9, and digits may repeat. Thus, there is a total of 10,000 possible PINs, and iOS will lock down the phone after 10 failed attempts, while Android allows for 20 failed attempts. In addition to securing the device, PINs are also used in banking applications, particularly Google Wallet [8] requires a user to enter a PIN to confirm transactions.

Password Pattern The Android password pattern was the primary phone lock interface on Android smartphones prior to the release of Android 2.2. In current versions of Android, users have the option to lock their phone using a four-digit PIN or a pass-phrase in addition to the password pattern. In recent user studies, despite known security issues with the password pattern [2], the vast majority of users who lock their phone, do so using the password pattern [12].

The Android password pattern is a graphical password scheme that requires users to enter a sequence of swipes that connect contact points in a three-by-three grid. The user must maintain contact with the screen while entering a pattern, and a user's pattern must minimally contacts four points (see Figure 1). Android allows for twenty failed pattern entry attempts before locking the device permanently.

There exists a number of restrictions on allowable patterns; particularly, if their exists an intermediate point not previously contacted between two points. The user implicitly includes these intermediate points regardless if the point is physically touched while entering a pattern. This restriction leads to a password set of just 389,112 possible patterns, much less than one might expect. Of course, with any password scheme, the actually set of usable passwords is likely much less because people choose passwords of convenience given that it must be entered over and over again. In our experience, as we will discuss in Section 4, a large portion of password patterns are simply to complicated or difficult to enter and are likely not part of the usable set.

Accelerometer Sensor The accelerometer sensor measures linear movements in three dimensions, side-to-side, forward-and-back, and up-and-down (labeled x, y, and z respectively in Figure 2). Upon each reading, a data element is provided that contains the acceleration reading in all three directions, and the units are in m/s^2.

Accelerometers have been previously studied in the computer science community, and researchers have shown that accelerometer readings can provide a rich source of information about the actions of individuals [3, 11, 14, 17]. The accelerometer sensor is used in many applications, notably in *Bump* [20], an application to quickly exchange contact information between smartphones by "bumping" them together. More light weight applications also make use of the accelerometer, for example applications that simulate the noises of a "light saber" will use the accelerometer to determine when to play a sound effect [9].

4 Data Collection

The hypothesis that we are testing in this paper is: A background application with access to the accelerometer can learn about input to a foreground application. To

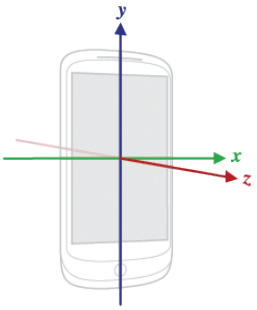

Figure 2: Accelerometer Axis of Measurement (Source [6])

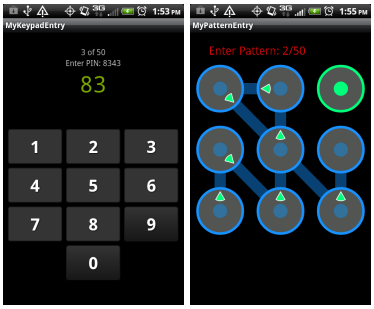

Figure 3: PIN and Pattern Entry Applications

test this hypothesis, we focus on two types of input that are representative of broad touchscreen input, Android password patterns and PINs, and to model the attacker's perspective, we developed two applications that collect accelerometer data while users enter PINs and password patterns. A visual of the applications can be found in Figure 3.

Both applications present the user with secure input to enter and record accelerometer readings while the user performs the assigned task. The accelerometer on the smartphone has different measurement frequencies, and we chose the highest frequency readings offered in the Android API. In practice, the actual rate of readings varied between 76 hz and 24 hz. This is because the accelerometer data is a service request, and the underlying Android OS decides when and how frequently to report acceleration changes.

In total we collected 3,600 samples of users entering in patterns and PINs selected from a set of 50 patterns and 50 PINs. A summary of the data collected can be found in Table 1. Although a sample size of 50 patterns or PINs might seem relatively small compared with the total pattern and PIN password space, the variance we measured across our cross validation runs is only 1.3% for patterns and 1.2% for PINs. This suggests that our sample size is sufficient to consistently learn effective classifiers.

We asked all test users to enter patterns and PINs in a consistent posture. Specifically, we asked users to sit at a table, and hold the phone in their right hand with their right elbow on the table. The PIN or pattern is entered with the thumb on the right hand without assistance from the left hand. All experiments were conducted with the phone in the vertical orientation, which differs from previous smartphone side-channel experiments in [4, 15]. We used three different Android phones in our experiment, HTC Nexus 1 (N1), HTC Droid Incredible 2 (Incr2), and HTC G2 (G2).

It is important to note that the patterns and PINs used in the experiment are not the test user's real patterns or PINs, and that real-world users will likely be very well practiced at entering in their own PIN or pattern. This familiarity could affect the way (*e.g.*, the way the phone moves in space) a user enters a pattern or PIN. We do not model this in our experiments (indeed, performing such an experiment on users actual secure input could be seen as unethical). However, our test users, by the end of data collection, have entered each PIN and pattern 12 times, providing considerable practice.

PIN Data We selected 50 PINs at random to form the attacker's training corpus, and all three test users enter the same set of 50 PINs a total of 12 times. If a user enters one of the PINs incorrectly, the application continues to prompt the PIN until it is entered correctly. In our analysis, we only consider accelerometer readings from correctly entered PINs and patterns. In addition to recording accelerometer readings, we also log the timing of the touch events to ensure that the accelerometer data matched the timing of PIN entry. We considered all accelerometer readings that occurred within 50 ms of entering the first digit and 50 ms after entering the last digit. A complete list of the PINs used in the experiments can be found in the Appendix.

Pattern Data We conducted two types of data collection for pattern entry. First, we collected information about individual swipes. The user is presented with a series of swipe gestures that connect all possible pairs of contact points. With 9 contact points, this data set consists of 72 unique swipes, and we conducted 10 runs with a single user.

The second data collection requires users to enter 50 different patterns a total of 12 distinct times. Initially, we chose patterns at random, but we quickly discovered that the vast majority of the patterns selected were incredibly hard to enter. They were convoluted and overly complicated, and our users reported that it took many iterations to enter them correctly. As a result, we wished

	Users	Patterns/PINs	Runs	Phones
Swipes	1	72	10	Incr2
Patterns	3	50	12	N1, Incr2
PINs	3	50	12	G2, Incr2

Table 1: Collected Data: Users per experiment, size of test set, runs per user, and which phones were used

Pattern Length	4	5	6	7	8	9
Distribution	2	5	14	18	8	3

Table 2: Length Distribution of 50 Pattern Test Set

to use a set of reasonable and representative password patterns that our test users can reliably enter on their first attempt. We developed two simple criteria to select patterns at random that meet this requirement.

The first criteria limits the number of cross overs, that is, it limits the number of swipe segments that cross (or double back) over previous swipe segments (*e.g.*, the pattern in Figure 3 contains a single cross over). The motivation for this criteria is that users would likely move in consistent directions. For example, we anticipate that users would generally select dots starting in one region and move to another region without doubling back too many times. The second criteria restricts contact points that are untouched: It requires that untouched contact points be next to other untouched contact points. Similar to the cross over criteria, this restriction again assumes that users will likely connect points in nearby regions, *e.g.*, just in the top half of the grid.

We do not argue that real world users apply these criteria while selecting their patterns, but in our experience, these criteria do provide patterns that our test users found much more reasonable to enter. Studying user selection criteria for password patterns is beyond the scope of this paper, and we are unaware of any such study. It should be noted that if this information were to become available, it could be easily incorporated into the machine learning classifiers used in this study and would further reduce the search space, yielding even better results than presented herein.

The distribution of pattern length for the test set of 50 patterns is presented in Table 2. A complete list of the 50 patterns can be found in the Appendix.

5 Data Analysis and ML Techniques

In this section, we present our analysis of the collected accelerometer data as well as present our machine learning techniques for classifying data. The accelerometer measurements for both PINs and patterns consist of a sequence of readings in each linear direction, plus a time stamp: $M = \{(x_i, y_i, z_i, ts_i)\}$. In addition to the accelerometer measurements, we also record the timing of touch events. A touch event for a PIN is when the user presses a digit, and a touch event for a pattern is when a user swipes across a contact point. The touch events are used to properly align the accelerometer data.

Malicious applications will not have direct access to touch events from other applications – if it did, then there would be no need to employ side channels. A malicious application must also determine when secure input begins and how to segment the accelerometer readings. Algorithmically learning touch events from raw accelerometer data is beyond the scope of this study; however, other machine learning techniques (or information form other side channels) could be employed to solve this problem.

Before proceedings with classification analysis, it is important to confirm that accelerometer measurements differ across different patterns and PINs. In Figures 4 and 5 we present four sample accelerometer data for PINs and patterns collected from a single user. There are noticeable differences in the readings between PINs and patterns; particularly, patterns produce smoother curves while PINs have more noise. This is to be expected because striking the smartphone screen causes more movement than gesturing.

Observe in Figures 4 and 5 is that there are strong visual differences between the two PINs and between the two patterns. For example, the y and z curves in Figure 5 display very different shapes near the 1000 ms mark. The graph on the right in Figure 5 shows a consistent flat period as the user transitions from striking a 2 to striking the 6 in the PIN 6626. During the same time period, the graph on the left has no such flat period. It is precisely these properties that an attacker can use to differentiate between input entered on the smartphone. In the rest of this section, we describe the procedure for extracting features from the accelerometer data, and following, we discuss the machine learning classifiers and experimentation techniques.

5.1 Feature Extraction

We employ a very simple set of statistical features to analyze accelerometer measurements. The first step in feature extraction is to divide the measurements into fixed size time windows (*e.g.*, of 100 ms). Then, within each window (or segment), the average, median, standard deviation, variance, max and min values are calculated for each x, y, and z signal. Each segment has a total of 18 features, 6 statistical features in 3 linear directions. The combination of the features for each segment form a fea-

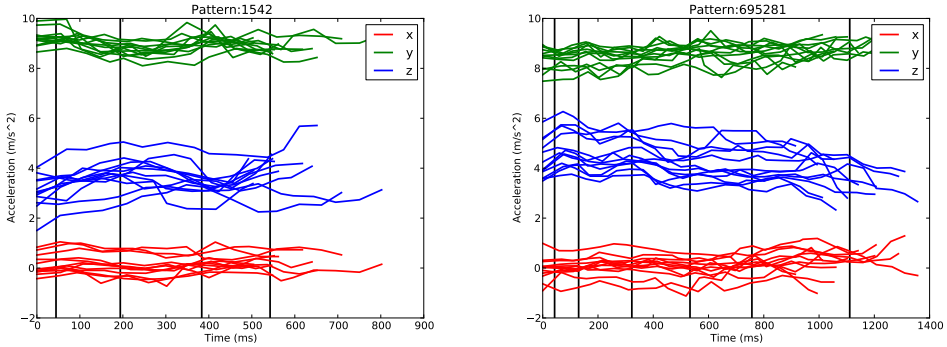

Figure 4: Sample Accelerometer Data for Patterns 1542 (*left*) and 695281 (*right*): Vertical lines indicate average time of contact with a contact point across 12 runs.

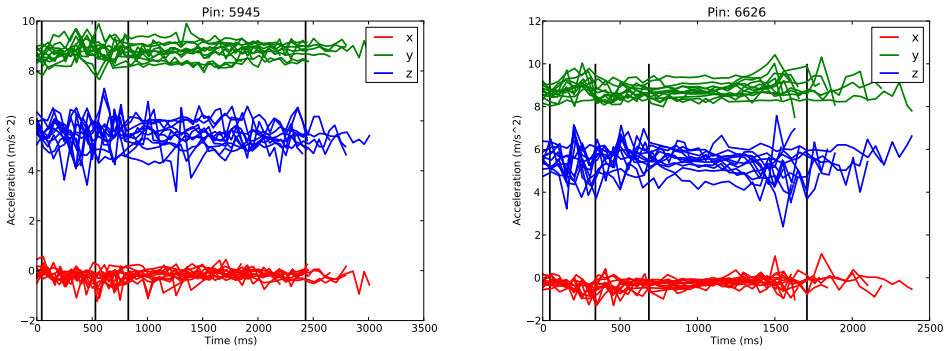

Figure 5: Sample Accelerometer Data for PINs 5945 (*left*) and 6626 (*right*): Vertical lines indicate average time of contact with number pad across 12 runs.

ture vector that is used as input to train a machine learning classifier.

The total number of features in the feature vector is dependent on the amount of time taken to enter the PIN or pattern. For example, if a pattern takes 1000 ms to enter and the segment size is 100 ms, then there exists 10 distinct segments resulting in 180 entries in the feature vector. Similarly, a pattern that takes 1200 ms will have 12 segments and 216 features.

To account for the difference in length across feature vectors, empty vector values are filled with zeros. This has the unfortunate side effect of potentially training the classifier on the length of the feature vector (or the time it takes to enter the PIN or pattern). Fortunately, there is relatively low variance in the number of segments for different PINs and patterns for the optimal segment length, and further, we found that predictions based only on the length of the input produces poor results, a small fraction greater than random guess. See Section 6 for an analysis on choosing an optimal segment length.

5.2 Machine Learning Classifier

Two classification procedures are used in experimentation to match the attack scenario described in Section 2. Recall that we wish to model two scenarios: (1) The attacker has a large corpus of labeled accelerometer data at his/her disposal and attempts to match unknown input to some label in the corpus; and (2), the unknown input is not in the corpus (or not well represented).

To model the first scenario, where the attacker is matching unknown input to labels in a corpus, we train a SVM (Support Vector Machine) based on the feature vector labeled with the PIN or password pattern. Given accelerometer data from entering a PIN or pattern not used in training, the resulting SVM model will output a predicted label (*i.e.*, a PIN or pattern). If the label matches the input, we consider this a successful prediction.

There are some limitations to this experiment because the SVM model only knows about the PINs and patterns

in the training set; that is, the 50 pattern or 50 PINs used in the experiment as opposed to all 389,112 possible patterns and 10,000 possible PINs. However, picking from random chance of the possible 50 patterns would result in a 2% prediction accuracy. The model greatly exceeds random guessing by a factor of 20 or more, and in our best performing setting we predict 60% of the patterns correctly and 49% of the PINs.

To model the second scenario, where the attacker's corpus may not have sufficient samples of the unknown input, we build a classifier that can predict potentially unseen patterns and PINs. To achieve this, we incorporate the probabilistic output of SVM classifiers into a Hidden Markov Model (HMM). The HMM finds the most likely sequence of input patterns or PINs (*maximum a posteriori*) by jointly considering the probabilities of individual swipe or digit entry classifications along with the likely transitions between swipes or digit entries. For example, for a four-digit PIN, the HMM jointly infers the most likely set of four digits given the individual beliefs in what digit was pressed at what time, and what digits are likely to follow other digits – certain combinations of digit transitions are impossible, and others are more likely than others. The same inference process can be used for patterns based on which swipes (connecting two contact points) are likely to follow previous swipes.

In our experiments, we utilize both a uni-gram and bi-gram HMM and estimated our transition matrix via maximum likelihood from a small dataset covering the 50 patterns and 50 PINs used in our experiments. This is a proof of concept, and a larger model could incorporate more refined transition matrixes that accounts for human pattern/PIN selection factors. In the uni-gram setting, smaller classifications are based on predicting a single swipe or digit, and in the bi-gram setting, prediction is based on two swipes or two digits.

6 Experimental Results

In this section, we present the results of our experiments. Initially, we explore the ability of the machine classifier to identify a single swipe motion, both the direction of the swipe, *e.g.*, swiping left or right, as well as identifying the start and end point of a swipe, *e.g.*, swiping from the first contact point to the last contact point. Next, we present our results for classifying an entire PIN or pattern from the test set of 50. We also investigate using accelerometer data collected from one user to predict the input of another. Finally, we discuss the results of building a general classifier that can handle potentially unseen PINs or patterns using a Hidden Markov Model.

All the results presented in this section were calculated by performing a five-fold cross validation. We performed 5 such cross validation runs and report the average across those runs. We found that there is very little variance in the results for each cross validation run: 1.3% for patterns and 1.2% for PINs.

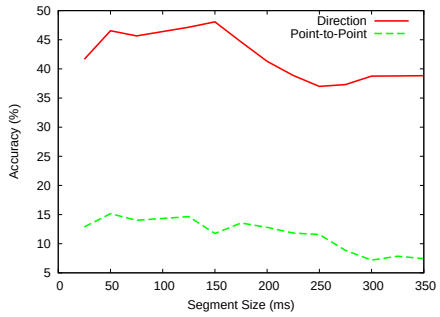

Figure 6: Results for Classifying a Single Swipe Gesture for Different Segment Sizes

6.1 SVM Classification

Single Swipes First, we conducted an experiment that tests the accelerometer's sensitivity to single swipe motions. We consider classification of two types: identifying the direction of a swipe and identifying the starting and ending point of the swipe.

As a test set, a single user connected all combinations of contact points in a pattern, consisting of 72 unique swipes. To classify a direction, we considered eight possible directions of a swipe: N, S, E, W, NE, SE, NW, SW, where N (north) is moving upward on the screen, S (south) is moving downward, E (east) is moving right on the screen, W (west) is moving left on the screen, and *etc.*. Of the 72 unique swipes, there is an even distribution in each direction, and thus classification by random guessing would predict a swipe's directions with 12.5% accuracy. To classify point-to-point, *i.e.*, predicting the start and end point of a swipe, each swipe in the test set is unique, and thus classification by random guessing would predict with 1.3% accuracy.

The results of classification for single swipes are presented in Figure 6. The accuracy values are obtained by performing a five-fold cross validation across the 10 runs for a test user. We also consider different segment sizes (the x-axis) for feature extraction. The model is able to identify the swipe direction with high accuracy, 47% when the segment size is 150 ms, which is 4 times greater than random guessing. Similar results were seen for point-to-point classification: 15% prediction accu-

racy with a segment size of 50 ms, which is a factor of 11 times greater than random guessing.

The results of this experiment are informative about an attacker's ability to learn general input, as well as the attacker's ability to perform sequence prediction for previously unseen input.

Patterns Next, we experiment with detecting an entire pattern from the test set of 50. This experiment models the first attack scenario where the attacker has a large corpus of labeled data to compare to unknown input. The data used in this experiment consists of 50 test patterns entered in 12 independent runs by each of the three test users. Two of the users, User 1 and User 2, entered the patterns on a HTC Droid Incredible 2, while the remaining user, User 3, entered the patterns on a HTC Nexus 1.

We are interested in two primary results. First, we investigate how well the model can perform if it is trained and tested with data collected from a single user. This would be the case if the attacker's corpus contains samples from the victim user. However, this may not be the case, so we also investigate the performance of the machine classifier if it is trained on one user and tested on another, as well as across different phone types.

The results for identifying a pattern from the set of 50 test patterns is presented in Figure 7 (*left*). The graph presents the results of a five-fold cross validation using different segment sizes for feature extraction. The first observation is that the patterns entered by all three users are well predicted. In the best performing experiment, the model can classify the pattern entered by User 1 with an accuracy of 60% using a segment size of 200 ms. The best experiment for User 2 identified patterns with 50% accuracy using a segment size of 225 ms, and for User 3, a prediction accuracy of 48% was achieved with a segment size of 200 ms. These results indicate that there is a strong correlation between pattern input of the same type, and that if an attacker has sufficient samples, the password pattern will likely be identified with high probability.

However, the attacker may not have sufficient samples from the victim user, and may need to train the machine learning classifier using samples collected from other users. In Table 3, the results for training and testing on different users are presented. We choose a segment size of 225 ms for this experiment because this segment size was close to the maximal prediction for all three users. In each row, a model is generated where the training data comes from a single user, and the model is then used to predict the input from a different user, indicated in the column header.

Although the prediction accuracy is degraded; however, classification still performs well. Note that random guessing provides a prediction accuracy of 2%, and in all combinations of users, the prediction accuracy far exceeds random guessing. Further, training on a different phones also provides strong results – User 3 used a Nexus 1 phone, while the other two users used an Incredible 2 – and this suggests that training can occur broadly with many different users and devices.

PINs We performed similar experiments for measuring the ability of an attacker to learn PIN input. Like patterns, three users each entered in a set of 50 PINs a total of 12 times. Figure 7 (*right*) presents the five-fold cross validation results for different segment sizes.

The results for classifying different PINs is also very strong. The model could classify 49% of the PINs for User 1 with a segment size of 250 ms. Similarly, for User 2, the model can predict PINs with a 47% accuracy with a segment size of 150 ms, and the model can predict User 3 with an accuracy of 43% with a segment size of 175 ms. Again, these results show that an attacker with a sufficient corpus of samples from a victim user can identify the user's PIN.

As before, we are also interested in the ability of an attacker to classify input if the model is trained on input from different users as would be the case if the attacker does not have samples from the victim user. The results of that experiment are presented in Table 4, and the results are similar to those for patterns. Although the prediction accuracy is reduced, it is still a factor greater than random guessing, and these results, again, indicate that training can occur broadly across users and phones.

6.2 HMM Classification

Encouraged by the results for classifying input from the test set of 50 patterns or PINs, we wish to build a model that can potentially classify arbitrary input of varying length. This experiment models the second attack scenario where an attacker does not have a sufficient corpus to compare to for unknown input. Instead, the attacker attempts to make a sequence of smaller predictions to identify the pattern or PIN.

A Hidden Markov Model (HMM) is a standard technique for solving this problem. An HMM combines the probabilistic output of the SVM models with a set of likely transitions.

We constructed an HMM by training a transition matrix based on our sample set of 50 patterns and PINs. This transition matrix is just a proof of concept, and the attacker could build a more representative transition ma-

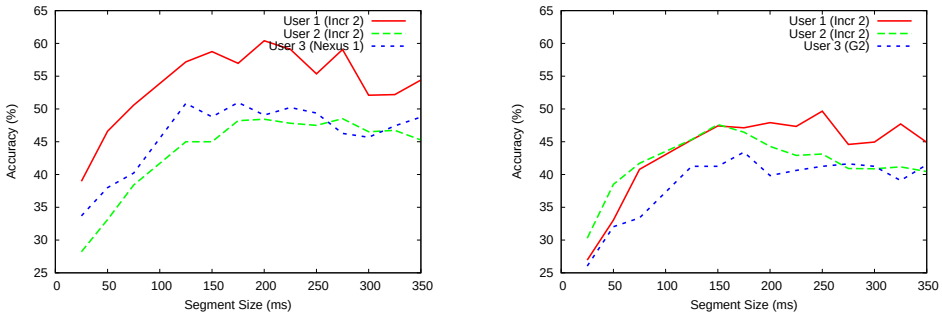

Figure 7: Classification Results for (*left*) Patterns and (*right*) PINs with Different Segment Sizes

		Testing Set		
		User 1	User 2	User 3
Training Set	User 1		16%	8%
	User 2	22%		17%
	User 3	13%	23%	

Table 3: Results for *Pattern Prediction* when Training on Different Users (225 ms Segment Size)

		Testing Set		
		User 1	User 2	User 3
Training Set	User 1		14%	21%
	User 2	14%		16%
	User 3	18%	15%	

Table 4: Results for *PIN Prediction* when Training on Different Users (175 ms Segment Size)

trix based on a larger set of patterns and PINs. Next, we trained an SVM to predict individual swipes or digit presses (like in the previous experiment), and then combined the transition matrix with the SVM to select the most likely sequence of swipes or digit presses. Further, we can use the HMM to sample the most likely set of patterns of PINs based on the prediction probabilities. This is particularly useful because Android allows for up to 20 guesses before locking the phone. We consider a pattern or PIN accurately predicted if it is in the top 20 most likely patterns or PINs obtained from the HMM analysis.

Initially, we built an HMM using uni-gram features and found the results were poor, so we extended the HMM for bi-grams. A bi-gram HMM predicts pairs of swipes or digits being entered to form larger predictions, as opposed to uni-gram HMM which predicts individual swipes or digit presses. The bi-gram HMM greatly outperform the uni-gram version. In five-fold cross validation, the bi-gram HMM is able to identify the pattern entered within the top 20 guesses 14% of the time, on average across all three users. The pattern entered was the top guess 8% of the time. Similarly for PINs, the bi-gram HMM can select the correct PIN within 20 guesses 23% of the time on average across all three users; however, the top choice was the correct PIN 2% of the time.

Finally, it is important to note that an attacker does not have just a single try to predict the password pattern or PIN. Beyond the 20 guesses, the attacker will likely have collected many accelerometer readings for a particular input because the victim enters secure input many times over. Consider how many times a day a user must unlock his/her smartphone, and the accelerometer readings from each of these events can be used as input to the model. As a result, the attacker will have many sets of 20 possible labels for the input, and the attacker can perform analysis across those predictions. For example, if a PIN is found in 90% of the predicted sets of 20, it is probably the user's PIN with high likelihood. In this lies the real power of the accelerometer side-channel: The attacker does not need to guess immediately and can train over time as more and more accelerometer data is collected.

7 Sensors and Mobile Device Security

As we have shown, an accelerometer sensor on a mobile device can constitute a surprisingly high-bandwidth side channel for touchscreen user input. Under some circumstances, this channel is of sufficiently fidelity to reduce the search space for PINs and graphical passwords to a tractable set of possibilities. Clearly, any effective security mechanisms for touchscreen devices with such sensors must deny untrusted applications access, at a minimum, to the accelerometer when sensitive touchscreen input is being provided to other applications.

At the same time, it may be equally undesirable to

restrict access to the accelerometer (and other sensors) when sensitive input operations are *not* being performed. Environmental sensors are used, to good effect, to provide a rich user experience in a diverse range of applications, including popular games (for kinetic input) and even for personal health (*e.g.*, pedometers). Many of these (legitimate) applications are designed to run in the background at all times. Preventing such applications from gaining access to the accelerometer at any time, or requiring the user to manually shut them down before performing any sensitive operation, would greatly reduce the appeal of the current trend toward general-purpose mobile platforms.

One approach might be to carefully vet applications that use sensors for malicious behavior before allowing them to be installed or before making them available in application markets. Unfortunately, this approach is logistically impractical at scale and, in any case, would require a level of analysis that ultimately reduces to the Halting Problem. Despite the clear drawbacks, this is the approach taken by Apple when vetting applications for the App Store. An alternative approach, as exemplified by Google in the Android App Market, is to label applications that access sensors (or other services) using a permission model; however, this is also insufficient because users may either ignore such labels or do not understand their implications.

We propose a different approach. Applications installed by the user that require access to sensors, however frivolous they may seem, should be able to use them. But, the sensors should be disabled (or untrusted applications denied access to them) whenever a trusted input function – such as password entry – is being performed.

Unfortunately, the security models implemented by current handheld platforms do not allow this kind of temporal control over sensors. Instead, applications declare what they need access to once (typically when they are first installed by the user or first run), and, from that point onward, have essentially unrestricted, permanent access to everything they asked for at any time they wish.

Although current mobile platforms do not support temporary revocation of sensor access, it could be implemented in a straight forward way, *e.g.*, via a system call available to trusted input functions to obtain and revoke exclusive access to sensors. One approach would be for this system call to cause any untrusted application that requests access to a sensitive sensor to block (or fail) until the sensitive operation has concluded. Alternatively, untrusted applications could simply be suspended for the duration of the sensitive input.

8 Related Work

The study of sensor-based side channels on smartphones is in its infancy: Two previous papers have investigated the ability to learn touch input from sensors [4, 15]. Our work builds and expands upon that research, showing that the accelerometer is even more sensitive than previously thought, and input can be leaked with greater accuracy. Additionally, we show that gesturing input types, not just point touching, can also be leaked via an accelerometer side channel. Our results also continue a longer tradition of side channel research. In particular, research on side channels for keyboard input is closely related [13, 19, 1, 22]. In the rest of this section, we describe the previous work.

Sensor Based Side Channels Cai *et al.* first proposed using on board sensors as a side channel to learn users input [4]. Their system, *touchlogger*, describes a side channel that employs the gyroscopic orientation sensor to determine broadly where a user touches on a large keypad. Their results were very encouraging, and under controlled setting, were able to infer which of the 10 regions a user touched with 70% accuracy. Our work differs in that we are using the accelerometer sensor to infer swipes as well as touches, and we focus on known secure input, such as PIN or pattern entry.

More similar to our work is *ACCessory* [15] by Owusu *et al.*. In *ACCessory*, the authors demonstrate that the accelerometer can be used as a side channel to infer short sequences of touches on a soft keyboard, and that standard machine learning techniques can be employed to infer input like passwords. Similarly, we show that the accelerometer can be used to infer secure input, and we also demonstrate that input can be classified with a sequence predictor. Our work differs from Owusu *et al.* in that we also demonstrate that swiping can be inferred from accelerometer data in addition to touch input. We additionally show that certain touch input, like PIN entry, can be classified at a much higher rate and with fewer guesses than suggested by Owusu *et al.*. *ACCesory* was able to classify input strings of length 6 with 60% accuracy, but needed 2^{12} guesses to achieve that result. In a similar experiment with PIN entry (*e.g.*, identifying touch events), we showed that the PIN entered can be classified with 23% accuracy in just 20 guesses.

Further, these two prior works when combined with the results herein, demonstrate that smartphone sensors are highly sensitive, the accelerometer sensor in particular. The kinds of input that the accelerometer sensor can record is broad, encompassing both touching and gesturing. Effectively, nearly all types of user input could

be inferred from the accelerometer sensor. The implications of this are far reaching with respect to the security of input on smartphones when an application has access to the accelerometer.

Smartphone Side Channels Side channels against secure smartphone input has been previously demonstrate for the password pattern input. Aviv *et al.* described a *smudge attack* that is based on observing the oily residues that remain on the touchscreen surface after a pattern is entered [2]. The side channel described here has a similar goal, but is based on internal sensors rather than external observations. It should be noted that a determined attackers should be able to combine information from both side channels to enhance their ability at correctly guessing the password pattern.

Other sensors and recording devices have been proposed as side channels. Shlegel *et al.* proposed Soundcomber [18] and demonstrated that a malicious app that has access to the microphone can learn the difference between general chatter and tone dialing, effectively learning the numbers a user calls. Xu *et al.* similarly considered information that can be leaked if a malicious app has access to the smartphones camera [21], and Cai *et al.* investigate sensors sniffing in earlier work, including the microphone, camera, and GPS receiver [5].

Keyboard Side Channels There is a rich history of research on side channels on user input based on keyboards. Most related is *(sp)iphone* by Marquardt *et al.*, where the authors demonstrated that the accelerometer on a smartphone placed next to a keyboard can record the timing of keypresses [13]. Previously, researchers have shown the timing of key presses can be sufficient to determining sensitive input, such as passwords [19]. Similar results to were shown for side channels based on acoustic emanations from keyboards [1, 22].

9 Conclusion

In this paper we showed that the accelerometer sensor is a surprisingly high-bandwidth side channel for touchscreen input that can be exploited by malicious applications to capture sensitive information (such as PIN entry or Android graphical passwords). Using off-the-shelf machine learning techniques, we were able to differentiate patterns from a set of 50 with 60% accuracy, and PINs with an accuracy of 49%. We also showed that there is some consistency in accelerometer data across different users and phones, and that a model trained on one user may be used to classify input from another with reasonable performance. Finally, we also showed that it is possible to classify individual parts of secure input – *e.g.*, a single swipe left or right, or touching a digit on the keypad – and that these smaller classifications can be combined to make larger, general classifications about user input.

The security models for current mobile platforms are inadequate to protect against sensor-based attacks in practice. In particular, applications that have access to the accelerometer sensor should not be able to read from the sensor while the user is providing sensitive input. But current mobile platform permission schemes are insufficiently to specify this; they provide applications with "all or nothing" access to every sensor they might ever need to use. Instead, the permission scheme and enforcement mechanism should restrict or allow access to sensors based on *context*. Untrusted applications that require access to a sensor should be granted access only when sensitive input operations are not occurring.

Smartphone sensors are undeniably useful for legitimate applications, and their potential for enhancing the user experience is still in the early stages of being unlocked. While granting an application access to "benign" sensors, such as the accelerometer, might seem inherently safe, the existence of powerful side-channels suggests, once again, that the situation is more complicated than it may seem.

References

[1] Dmitri Asonov and Rakesh Agrawal. Keyboard accoustic emanations. In *Proceedings of IEEE Syymposium on Security and Privacy*, 2004.

[2] Adam J. Aviv, Katherine Gibson, Evan Mossop, Matt Blaze, and Jonathan M. Smith. Smudge attacks on smartphone touch screens. In *Proceedings of the 4th USENIX Workshop On Offensive Technologies*, WOOT'10, 2010.

[3] Ling Bao and Stephen Intille. Activity recognition from user-annotated acceleration data. In *Pervasive Computing*, volume 3001 of *Lecture Notes in Computer Science*, pages 1–17. 2004.

[4] Liang Cai and Hao Chen. Touchlogger: inferring keystrokes on touch screen from smartphone motion. In *Proceedings of the 6th USENIX conference on Hot topics in security*, HotSec'11, 2011.

[5] Liang Cai, Sridhar Machiraju, and Hao Chen. Defending against sensor-sniffing attacks on mobile phones. In *Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds*, MobiHeld '09, 2009.

[6] Google Andorid Development. `http://developer.android.com/reference/android/hardware/SensorEvent.html`.

[7] Splasho Development. Pattern lock pro. https://market.android.com/details?id=com.splasho.patternlockpro.

[8] Google Inc. Google wallet. http://www.google.com/wallet/.

[9] THQ Inc. Star wars: Lightsaber duel. http://itunes.apple.com/us/app/star-wars-lightsaber-duel/id362158521?mt=8.

[10] Rupesh Jain. Pattern encrypt/decrupt upgrad. https://market.android.com/details?id=PatternEncryptDecryptUpgrade.free.

[11] Jiayang Liu, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. uwave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive Mob. Comput.*, 5:657–675, December 2009.

[12] Shu Liu and Aaron Striegel. Accurate extraction of face-to-face proximity using smartphones and bluetooth. In *Proceedings of 20th International Conference on Computer Communications and Networks*, ICCN, 2011.

[13] Philip Marquardt, Arunabh Verma, Henry Carter, and Patrick Traynor. (sp)iphone: decoding vibrations from nearby keyboards using mobile phone accelerometers. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, 2011.

[14] Uwe Maurer, Anthony Rowe, Asim Smailagic, and Daniel P. Siewiorek. ewatch: A wearable sensor and notification platform. In *Proceedings of the International Workshop on Wearable and Implantable Body Sensor Networks*, 2006.

[15] Emmanuel Owusu, Jun Han, Sauvik Das, Adrian Perrig, and Joy Zhang. Accessory: Keystroke inference using accelerometers on smartphones. In *Proceedings of The Thirteenth Workshop on Mobile Computing Systems and Applications*, HotMobile, 2012.

[16] Rio Park. Memorize pattern. https://market.android.com/details?id=riopark.pattern.

[17] Nishkam Ravi, Nikhil D, Preetham Mysore, and Michael L. Littman. Activity recognition from accelerometer data. In *In Proceedings of the Seventeenth Conference on Innovative Applications of Artificial Intelligence(IAAI*, pages 1541–1546. AAAI Press, 2005.

[18] Roman Schlegel, Kehuan Zhang, Xiaoyong Zhou, Mehool Intwala, Apu Kapadia, and XiaoFeng Wang. Soundcomber: A stealthy and context-aware sound trojan for smartphones. In *Proceedgins of the Network and Distributed System Security Symbosium*, NDSS, 2011.

[19] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on ssh. In *Proceedings of the 10th conference on USENIX Security Symposium*, SSYM'01, 2001.

[20] Bump Technologies. Bump app. bu.mp.

[21] Nan Xu, Fan Zhang, Yisha Luo, Weijia Jia, Dong Xuan, and Jin Teng. Stealthy video capturer: a new video-based spyware in 3g smartphones. In *Proceedings of the second ACM conference on Wireless network security*, WiSec '09, 2009.

[22] Li Zhuang, Feng Zhou, and J. D. Tygar. Keyboard acoustic emanations revisited. *ACM Trans. Inf. Syst. Secur.*, 13, November 2009.

A PINs

Below are the PINs used in the experiment:

8671 4624 8343 6603 7026 5178 3251 2358 4458 8849 3031 6626 1629 0650 5975 1777 1382 1709 2766 7495 2087 5181 9422 6848 5616 6993 5945 9663 2996 7226 9590 6350 4915 4482 6407 4457 7337 4448 7050 1192 1407 4675 6068 0717 9051 5946 5763 5365 7238 2021

B Patterns

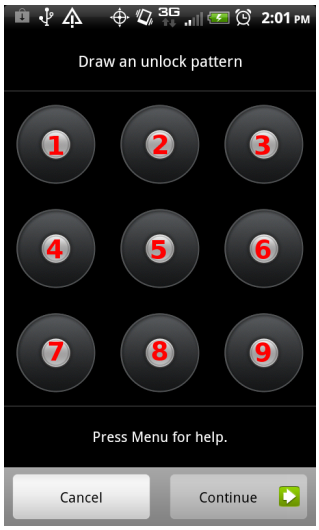

Figure 8: Contact Point Number References

Below are the patterns used in the experiment. Refer to Figure 8 for the ordering of the contact points.

5284693 6749231 2358417 58967 8695471 524638 524176839 7586923 3615294 594617 54982317 5879143 98652471 6392578 5836749 874563 12589436 6359471 58764239 35426 2547 8572639 1542 876529 36528914 695281 586241793 3695741 621458 749583 2584196 126594 769251 5872963 62584913 853476 125347 9658237 65491 3684179 74852196 578416 325914 564893217 7832169 14587 231548 32584697 51263 1523496