ABSTRACT
Mobile phones are equipped with an increasingly large number of precise and sophisticated sensors. This raises the risk of direct and indirect privacy breaches. In this paper, we investigate the feasibility of keystroke inference when user taps on a soft keyboard are captured by the stereoscopic microphones on an Android smartphone. We developed algorithms for sensor-signals processing and domain specific machine learning to infer key taps using a combination of stereo-microphones and gyroscopes. We implemented and evaluated the performance of our system on two popular mobile phones and a tablet: Samsung S2, Samsung Tab 8 and HTC One. Based on our experiments, and to the best of our knowledge, our system (1) is the first to exceed 90% accuracy requiring a single attempt, (2) operates on the standard Android QWERTY and number keyboards, and (3) is language agnostic. We show that stereo-microphones are a much more effective side channel as compared to the gyroscope, however, their data can be combined to boost the accuracy of prediction. While previous studies focused on larger key sizes and repetitive attempts, we show that by focusing on the specifics of the keyboard and creating machine learning models and algorithms based on keyboard areas combined with adequate filtering, we can achieve an accuracy of 90% - 94% for much smaller key sizes in a single attempt. We also demonstrate how such attacks can be instrumentalized by a malicious application to log the keystrokes of other sensitive applications. Finally, we describe some techniques to mitigate these attacks.

Categories and Subject Descriptors
K.6.5 [Security and Protection]: Unauthorized access; Invasive software; K.4.1 [Public Policy Issues]: Privacy

Keywords
Smartphone Security; Side-Channel Attacks; Sensor Malware; Tap Detection with Motion Sensors; Keystroke Inference using Gyroscope; Keystroke Inference using Stereo-Microphones; Machine Learning; Privacy

1. INTRODUCTION
Mobile devices have proliferated across the globe with an estimated 1.75 billion smartphone users world-wide by the end of 2014 [9]. These smartphones are used for various day-to-day and business activities, many containing sensitive information such as Personally Identifiable Information (PII), banking and credit card numbers, passwords, health records, and location information. This means that a malicious application installed on the smartphone can siphon off sensitive information and leave millions susceptible to data theft. Operating systems on these smartphones prevent unauthorized access to application / user information by implementing security mechanisms such as sandboxing and permissions. However, side-channels such as sensors can bypass such security restrictions. Most modern smartphones contain several sensors such as microphones, cameras, gyroscopes and accelerometers that provide a better user experience but these sensors, however useful, leak information that provides an adversary opportunity to covertly infer and steal sensitive data.

In this work, we demonstrate that by recording the tap sounds and vibrations from the stereo-microphones and gyroscope of a smartphone while a target application is running, it is possible to successfully infer user typed keys with high accuracy. We also show that the audio data can be combined with the gyroscope to further boost the accuracy. Android OS is popular among users and smartphone manufacturers with more than million activations daily [15]. Owing to this popularity, we chose to perform our experiments on two popular Android smartphones and a tablet: Samsung S2, Samsung Tab 8 and HTC One. While there have been related work, to the best of our knowledge, we are the first to break the 90% accuracy barrier on the standard Android QWERTY and number keyboard. For instance, some previous work that demonstrated key inference based on Android sensors, Aviv et al. [1] reached an accuracy of 43% and 75% on PIN and pattern passwords, respectively, requiring 5 attempts. Owusu et al. [23] were able to infer 6 passwords out of 99 six-character passwords in an average of 4.5 trials on the QWERTY keyboard in landscape mode, and Miluzzo et al. [21] predicted 4x5 icon taps on iPhone and Nexus S with an accuracy of 80%. To achieve an accuracy of 90%, they relied on more than 20 repetitions. To achieve a high accuracy on the standard keyboard, we developed an algorithm...
and framework based on statistical methods and machine learning that can predict keystrokes without repetition or multiple attempts. Our framework is language agnostic as we do not use any lexical properties of languages, however we do assume that the adversary knows the keyboard layout. We demonstrate the algorithm using data collected at an office and in a restaurant. A malicious application and a weak permission model for Android sensors coupled with data modeling techniques make our attack feasible and consequential. We make the following contributions:

- We show that by recording the tap sounds from the stereoscopic microphone and the tap vibrations from the gyroscope on a smartphone, it is possible to infer user's typed keys with a reasonably high accuracy of above 90%. We also show that this accuracy can be boosted by combining the audio data with the gyroscope data. We design an automated system that can process this raw keystroke data, perform noise filtering, build training models and use these models to make language agnostic keystroke predictions on unknown test data.

- We develop a specialized meta-algorithm that divides the keyboard into specific areas and trains models using audio and gyroscope data for those areas. The algorithm combines character-specific and area-specific models to make more accurate predictions. We show that by combining our algorithm with meta-algorithms such as Bagging and Boosting [8], we were able to achieve higher per algorithm accuracy than an elementary use of the machine learning algorithms on unknown test data.

- We demonstrate the feasibility of a Trojan that regularly queries the Android system for the foreground application and covertly records the microphone and gyroscope when a sensitive application is used.

The rest of the paper is organized as follows. In section 2, we describe a scenario of how a stealthy attack can be launched against a sensitive Android application by a victim inadvertently downloading a malicious application. In section 3, we describe the high-level architecture of our system and discuss why the gyroscope and microphone sensors leak information about a user's typing activity. In section 4, we describe our automated keystroke inference system, data collection process and the meta-algorithm. In section 5, we present the results of our experiments. In section 6, we describe techniques that can be used to mitigate these attacks. In section 7, we describe previous related work and in section 8, we conclude and discuss future research directions.

2. ATTACK VECTOR

To perform a successful attack, the adversary follows the set of steps described below. The adversary develops and distributes a malicious application usually as a Trojan (step 1). They trick the victim into installing the application (step 2) through techniques such as social engineering (e.g., a malicious application disguised as a game or a note taking application or through USB connection to bogus devices [2]). Previous works have found examples of such applications with backdoors in the Android marketplace [30, 32]. After compromising the victim’s smartphone, the application performs two roles. First, it presents a custom keyboard to the user to collect typing behavior for training models. Second, after training, it listens in the background for keypresses from sensitive Android applications.

To successfully perform the first role, the malicious application starts collecting user’s typing behavior by using a custom keyboard and recording the stereo microphones and gyroscope (step 3). The microphone requires explicit permissions during installation and the adversary needs to declare it in the manifest file. To avoid raising the victim’s suspicions, the adversary justifies such permissions by providing functionality such as: note taking, supporting voice commands and voice recognition. The application must serve an actual purpose (e.g., a Todo app) so that the user actually uses the application. After monitoring the victim’s behavior, the Trojan uploads the collected training data to a remote server to build models specific to the victim (step 4). The adversary can also use generic training data and prediction models as a trade-off for performance and stealth.

To accomplish the second role, the Trojan runs in the background and queries the smartphone OS for the current location and the application. To reduce battery drain, these queries run at predefined conservative intervals. On inferring that the victim is at a place of interest using the GPS or cellular and wireless networks (e.g., bank or residence garage door) or when the victim opens a sensitive application (e.g., a bank application), the Trojan starts to actively collect the microphone and gyroscope data (step 5). The current application can be found by using the Activity-Manager class in Android SDK if the application has the android.permission.GET_TASKS permission set in its manifest file. The data is filtered and keystrokes are extracted by the application and evaluated using the training models to infer the user’s typed keystrokes.

3. APPROACH

In this section, we describe the architecture of our key inference system that performs audio and gyroscope keystroke noise filtering and extraction, data consolidation for machine learning, model training, evaluation of test instances and evaluation of accuracy.

3.1 Architecture

The architecture of our keystroke inference system is composed of the components shown in Figure 1. The Application component is the trojan application that secretly records the gyroscope and stereo microphones when a user types in our application (training data) or another security sensitive application (test data). The raw gyroscope and audio data is uploaded to a remote server that runs the remaining components of our system to process this data, create training models and perform evaluations on the test data. This raw data contains user tap vibrations and sounds mixed with pauses and noise that occur due to external factors such as unstable hands (gyroscope noise) or background music (audio noise), see Figures 6 and 7. The Pre-processing component removes noise from the data using adequate filtering, extracts the user taps, and performs fitting to resample the data. The techniques used for filtering and extracting gyroscope and audio data are different and require two separate pre-processing components. Once the data has been processed, it is converted to a format that is understood by Weka [16]. During conversion, the gyroscope and audio data may also be combined for inference. This means that the two need to be synchronized with each other. The
Figure 1: Architecture of the key inference system.

Synchronization component synchronizes the gyroscope and audio data and the Consolidation component converts the extracted and filtered inference ready data to a machine learning toolkit format. In order to classify unknown test data, the consolidated training data is used to build models. The Training component analyzes the data for errors, randomizes them and uses several machine learning algorithms to build different inference models for the entire character set as well as specific areas on the keyboard. These models are then used by the Evaluation component to perform predictions on the unknown test data. The component uses a meta-algorithm that makes a final keystroke prediction. The meta-algorithm uses a multi-step approach based on the specific layout of Android QWERTY and number keyboards to optimize the inference accuracy of test keystrokes. Once predictions have been made for all unknown test data, the Accuracy Evaluation component compares them with the expected keystroke to evaluate our system’s performance.

3.2 Sensors

3.2.1 Gyroscope & Accelerometer

Gyroscope and Accelerometer sensors on smartphones can detect vibrations for every keystroke when a user types on a soft keyboard. Figure 2 shows the location of the gyroscope (in red), the location of keys ‘I’, ‘Q’ and ‘V’ on a standard QWERTY keyboard on the HTC One, and the co-ordinate system of Android sensors. The magnitude and orientation of these vibrations vary depending on the tap location with respect to the two axes which can be mapped to a standard fixed keyboard layout. In Figure 3, we see that key ‘Q’ shows significant vibration in the y axis and key ‘V’ shows significant vibration in the x axis. As the key ‘I’ is close to both the axes, it does not show considerable vibration in both the axes.

Gyroscope sensors are attractive targets for building smartphone keyloggers. They are easy to use in Android using the Android SDK APIs defined in the Sensor class. They do not require special permissions and they can run in the background without prompting or notifying the user. A potential issue with using the SDK API is that the sampling rate is not fixed and may reduce when more processor intensive services are running, thereby reducing the inference accuracy. A solution for obtaining high and constant sampling rate on an Android smartphone even when other high priority services are running on the system is to use the Android NDK APIs defined in sensors.h.

3.2.2 Microphones

Microphone arrays are becoming commonplace in smartphones. Some smartphones, such as the iPhone 5s, are equipped with three microphones. Other smartphones, such as the HTC One, are equipped with dual membrane microphones that focus on capturing different sound levels (one for sensitivity and the other for distance) on a single microphone. In both these arrangements, the audio captured by the arrays are combined and processed to provide high quality and distortion free audio recording to users with the capability to detect feeble sounds. In addition, the HTC One also supports stereo recording. We observed that a soft
tap on a smartphone, by a user, is audible. This tap can be covertly recorded and processed to infer a user’s typing activity with relatively higher accuracy than the gyroscope. There are two techniques to infer keystrokes, one that uses the amplitude of audio signals at the two microphones and the other that uses the time delay between signals reaching the two microphones. We use a combination of both the techniques to build our inference models. Figure 2 shows the location of the microphones (in blue) on the HTC One. Since the microphones are synchronized, for a tap at location T, the delay in tap detection between the two microphones M1 and M2 can be computed using the following formula.

\[
\text{Number of Samples} = \frac{(\text{Distance}(T,M1) - \text{Distance}(T,M2)) \times \text{Sampling Rate}}{\text{Speed of Sound}}
\]  

(1)

The distance between the two microphones on the HTC One is about 0.134 m. The current maximum supported sampling rate for Android is 48000 Hz and the speed of sound in air is about 340 m/s. Using these values with the formula, a difference of 18 samples is obtained for taps in close proximity to the microphones. This means that taps on different keys will produce varying sample differences based on their distance from the two microphones. The difference in samples will increase when smartphones start supporting higher sampling rates such as 192 KHz, currently supported by Blue-ray. Using a rate of 192 KHz with the formula, a difference of 75 samples can be obtained for taps in close proximity to the microphones. This will significantly improve the accuracy of inference and is indicative of the impact of the sampling rate on the accuracy.

To illustrate the time delay between the microphones, we recorded multiple samples for keys ‘Q’ and ‘V’ on a standard QWERTY keyboard on the HTC One at a sampling rate of 48KHz. Figure 4 shows a single tap for the two keys. We found that, for multiple taps, the two keys always have the same delay of 8 and 15, respectively, between M1 and M2. The figure also shows that the amplitudes at M1 were significantly lower than M2. This is because our application uses M2 as the primary microphone, and requests the Operating System to perform noise cancellation when supported. The lower amplitudes at M1 are a result of noise cancellation by the system.

Microphone functionality is easy to implement in Android using the Android SDK APIs defined in the AudioRecord class. Even though they initially require special permissions from the user, once installed, they can run in the background without prompting the user or showing a notification. This makes the microphones an attractive target for building Android keyloggers. Stereo recording is a relatively new concept in smartphones and we expect a lot of new smartphones to adopt this technology. We believe that this capability can be used maliciously and should be addressed.

4. SYSTEM DESIGN AND ALGORITHMS

In this section, we describe the hardware and software we developed, the data collected, the gyroscope and microphone noise filtering and extraction process, the training process and the meta-algorithm that we developed to make predictions on unknown test data.

4.1 Data Collection

4.1.1 Hardware & Software

In this work, we decided to evaluate our algorithms and system on two popular Android smartphones and a tablet: Samsung S2 (Android 4.1), Samsung Tab 8 (Android 4.1) and HTC One (Android 4.4); all three using the stock Android Operating System provided by the vendor. We did not make any modifications to the operating system and its underlying mechanisms. We developed an application to collect accelerometer, gyroscope and microphones data for training the models and for generating test samples for evaluation. It runs in two separate modes for training and test data. The data collected in the training mode is used for building the inference models and the data collected in the test mode is used for making predictions using these models. In case of the test mode, the application invokes a service that runs in the background. These two modes are completely independent of each other and do not overlap. Our application implemented a custom keyboard that has the same layout and capabilities as the standard Android QWERTY and Number keyboard. The only difference is that our keyboard has the capability to detect a key press and inject a key press event in the gyroscope data. Figure 5 shows the screenshot of the custom keyboard. The user presses the ‘Start Recording’ button and types data on the keyboard. Whenever they are done typing their data, they press the ‘Stop Recording’ button that is visible after recording is started. During this typing session, the accelerometer, gyroscope and microphones are activated in the background and their data are recorded and stored on the smartphone’s internal memory.

4.1.2 Gyroscope & Microphones

We collected accelerometer, gyroscope and audio training and test samples for both the number and QWERTY keyboard in Portrait mode on the Samsung S2, the Samsung Tab 8 and the HTC One. The data was collected by seven participants who were asked to take samples in their normal typing style. We observed their typing behavior and all of
of orientation in radians/second. We use this rate to construct the gyroscope recording and then use this as our raw data. The raw data is filtered, extracted, resampled and converted to a machine learning toolkit compatible format. Figure 6 shows the stages of pre-processing for a gyroscope sample, these stages are described below.

![Figure 6](image)

**Figure 6:** (Top Left) A noisy unprocessed raw gyroscope sample for letter ‘V’; (Top Right) Noise removed from the sample after frequency filtering; (Bottom Left) The filtered sample after tap extraction; (Bottom Right) The sample after resampling.

**Filtering:** The gyroscope noise during a typing session occurs mainly because of the instability of the hand and this noise is typically high frequency. When the change in orientation is much higher than the noise, i.e. high signal-to-noise ratio, the noise can be removed by simple frequency filtering techniques. We use a Fast Fourier Transform filter [11] to detect frequencies corresponding to the sample values (the tap). We keep these frequencies unchanged and remove the amplitudes at the other frequencies. We then apply an Inverse Fast Fourier Transform to obtain the filtered data. The technique works quite well, however, for noisier data advanced filtering schemes such as Kalman Filtering [28] can be applied. When the signal-to-noise ratio is low, these filters may remove significant tap specific information reducing the accuracy. One option is to use the unfiltered raw data in such cases but we observed that the accuracy with filtered data is much better than raw data. This is because the noise in the data changes the wave-form and makes them dissimilar even for the same location. Another option is to observe the gyroscope vibrations and record only when the noise is under a threshold.

**Extraction:** Our trojan Android application uses a custom keyboard with the standard Android keyboard layout for tap detection. The keyboard injects an event into the gyroscope data when a key is pressed. Our system uses this as the start of the sample and uses a constant time difference to compute the end of the sample.

**Resampling:** The filtered gyroscope data are of different sizes as the Android SDK does not allow setting a fixed sampling rate for sensors. However, most machine learning toolkits are very specific about file formats and require fixed length training and test samples, therefore the data has to be resampled before consolidation. Another reason for resampling is to increase the size of the data such that

---

In the Android environment, the gyroscope sensor is used to capture motion data. The Android Sensor API returns the rate of change of orientation along the x and y axes. We use this rate along with the gyroscope recordings to extract features for machine learning.

The data collected from the gyroscope are the smartphone’s time, gyroscope accuracy, x axis orientation, y axis orientation, and the z axis orientation at the maximum sampling rate for the device. We additionally collected the key press time using a custom keyboard having the same layout and capabilities as the standard Android QWERTY and Number keyboard. We chose to discard the z axis orientation as vibrations caused by keystrokes mainly affected the x and y planes.

The data collected from the microphones are the amplitudes received by the two microphones. We collected raw audio data instead of pre-processed data. A sampling rate of 48kHz was chosen because Android currently does not support higher sampling rates. During the initial phase of the experiment, we also collected accelerometer data from the smartphone. Subsequently, we did not consider this sensor because the accuracy was significantly lower than the gyroscope. The reason was that the accelerometer combines acceleration and gravity and the gravity component varied significantly even when the phone was placed stationary on a table. Even when the gravity vibrations were removed by filtering techniques, there was not much improvement in accuracy.

**4.1.3 Synchronization**

The gyroscope and audio data are synchronized by injecting a microphone start event in the gyroscope data. The trojan application invokes two threads to start the gyroscope and microphones, however, the microphones are started only after the gyroscope has completely initialized. This is done to ensure that the gyroscope sensor starts at the highest sampling rate and does not get reduced to a lower rate due to resource consumption by the microphones. Once the microphone starts recording, the application generates a microphone start event that gets injected into the gyroscope data. The two are then recorded simultaneously.

**4.2 Pre-processing**

**4.2.1 Gyroscope**

Contrary to previous works [29], [21], [5] that extracted features from the gyroscope recordings, we use the raw gyroscope x and y axes orientations as the feature set for machine learning. The Android Sensor API returns the rate of change of orientation in radians/second. We use this rate to construct the gyroscope recording and then use this as our raw data. The raw data is filtered, extracted, resampled and converted to a machine learning toolkit compatible format. Figure 6 shows the stages of pre-processing for a gyroscope sample, these stages are described below.

**Filtering:** The gyroscope noise during a typing session occurs mainly because of the instability of the hand and this noise is typically high frequency. When the change in orientation is much higher than the noise, i.e. high signal-to-noise ratio, the noise can be removed by simple frequency filtering techniques. We use a Fast Fourier Transform filter [11] to detect frequencies corresponding to the sample values (the tap). We keep these frequencies unchanged and remove the amplitudes at the other frequencies. We then apply an Inverse Fast Fourier Transform to obtain the filtered data. The technique works quite well, however, for noisier data advanced filtering schemes such as Kalman Filtering [28] can be applied. When the signal-to-noise ratio is low, these filters may remove significant tap specific information reducing the accuracy. One option is to use the unfiltered raw data in such cases but we observed that the accuracy with filtered data is much better than raw data. This is because the noise in the data changes the wave-form and makes them dissimilar even for the same location. Another option is to observe the gyroscope vibrations and record only when the noise is under a threshold.

**Extraction:** Our trojan Android application uses a custom keyboard with the standard Android keyboard layout for tap detection. The keyboard injects an event into the gyroscope data when a key is pressed. Our system uses this as the start of the sample and uses a constant time difference to compute the end of the sample.

**Resampling:** The filtered gyroscope data are of different sizes as the Android SDK does not allow setting a fixed sampling rate for sensors. However, most machine learning toolkits are very specific about file formats and require fixed length training and test samples, therefore the data has to be resampled before consolidation. Another reason for resampling is to increase the size of the data such that
minute changes in data are identified by machine learning algorithms. We use a technique known as Cubic Spline Interpolation [20] to resample the data without changing the waveform of vibrations. Our evaluation results confirm that greater number of samples increase the inference accuracy.

4.2.2 Microphones

Instead of extracting specific properties from the audio samples and using them as features, we use the raw audio received at the two microphones. The raw audio data is first filtered to remove noise, then extracted to obtain the tap, re-sampled and converted to a machine learning toolkit compatible format. Figure 7 shows the stages of pre-processing for an audio sample, described below.

Filtering: The noise in audio signals can be present due to several factors such as background music, conversations, and moving traffic. Audio noise is typically high frequency and can be removed using frequency filtering techniques. We use a bandpass filter to pass all frequencies in the range of 1500 to 3500 Hz and stop the remaining frequencies. Bandpass filtering is an effective technique when the tap and noise frequencies don’t overlap. The frequency range was obtained by analyzing recordings on the HTC One such that background noise was removed while retaining the frequencies of the tap sound. A low signal-to-noise ratio can have a significant impact on the inference accuracy and even when filters exist to remove noise, the noise removal algorithm may also change the original waveform and decrease inference. One option is to observe the microphones data and record only when the noise is under a threshold.

Extraction & Resampling: We use the peak amplitude at the two microphones to detect the start of the sample. The channel that has the higher amplitude becomes the base channel and about two wavelengths are extracted to ensure that the peak of the other channel is extracted as well. The audio samples are interpolated so that changes in data are detected as features by the machine learning algorithms.

4.3 Training Process

The system uses the specifics of Android QWERTY and number keyboard and a number of steps and algorithms to develop adequate training models.

Table 1: Accuracies of elementary algorithms for some sample sets.

<table>
<thead>
<tr>
<th>Keyboard</th>
<th>Sensor</th>
<th>DT</th>
<th>NB</th>
<th>NN</th>
<th>10-NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTC One</td>
<td>Number</td>
<td>70%</td>
<td>72%</td>
<td>66%</td>
<td>60%</td>
</tr>
<tr>
<td>HTC One</td>
<td>Number</td>
<td>68%</td>
<td>73%</td>
<td>78%</td>
<td>71%</td>
</tr>
<tr>
<td>Samsung S2</td>
<td>Number</td>
<td>60%</td>
<td>61%</td>
<td>58%</td>
<td>52%</td>
</tr>
<tr>
<td>Samsung S2</td>
<td>Number</td>
<td>74%</td>
<td>74%</td>
<td>82%</td>
<td>72%</td>
</tr>
</tbody>
</table>

Consolidation: The filtered training data are consolidated into a single database file in the machine learning toolkit format. The file can contain only the gyroscopic data (for non-stereo microphones), the microphone data or the microphone + gyroscopic data combined (for stereo-microphones). Unlike [21], we do not use any techniques to combine the microphone + gyroscopic data but simply append their contents. We reason that these sensors have different properties that may be lost if combined.

Area Division: The keyboards were broken down into areas such that all keys in a particular area are distinguishable from another using at least one feature. The area division for a standard QWERTY keyboard in portrait mode is shown in Figure 5. They are chosen such that the x-orientation remains constant and y-orientation varies for all keys in the area. For example, for the area ‘QWE’, the negative y-orientation will be higher for ‘Q’, lesser for ‘W’ and the least for ‘E’. The area division for a standard number keyboard in portrait mode is shown in Figure 5 and follows the same technique except for keys ‘8’ and ‘0’ that will have a constant y-orientation and varying x-orientation. We tested other area divisions as well but this division worked better than the others.

Elementary Algorithms: The Weka toolkit offers a large variety of machine learning algorithms. The problem of predicting the keystrokes is a supervised classification problem, therefore we eliminated algorithms that do not apply in this context (e.g., K-Means). Before developing our meta-algorithm, the algorithms we tested were Decision Trees, Naïve Bayes, K-Nearest Neighbor (k-NN), Hidden Markov Models, Support Vector Machines, Random Forest and Neural Networks. Of these algorithms, Decision Trees (DT), Naïve Bayes (NB), 1-Nearest Neighbor (NN) and 10-Nearest Neighbor (10-NN) performed better and yielded higher accuracy rates. Neural Networks also yielded high accuracy rates but was not chosen due to heavy resource consumption. In the context of our work, instance-based methods such as k-NN yield high accuracy, since they try to find the closest match between the new prediction and the training data. Table 1 shows the performance of these elementary algorithms for three QWERTY and three number keyboard sample sets. As we can see, none of these algorithms perform well on all areas of keyboard because of overlapping instances. This observation drove us to develop a meta-algorithm which considers the areas of keyboard before making predictions.

The Training Process: The goal of the training process is to build inference models based on the entire character set and areas defined for the keyboard.
In step 1, training models are built for predicting areas of the keyboard using a voting [18] algorithm. The voting algorithm uses the prediction of all the algorithms and their confidence values to determine the area of the test data. The model for voting uses ensembling techniques such as Bagging and Boosting to improve the accuracy of algorithms. The Ensembling technique builds multiple models from subsets of the training data, analyzes the accuracy of the subsets to detect incorrectly classified instances, and then uses these instances again with different weights or averaging to build better predictive models.

In step 2, training models are built for the entire character set for all the algorithms. These models also use ensembling. If the training data specified is microphones + gyroscope, then training models for microphones are also built. This is because the gyroscope data for certain areas of the keyboard that are close to the actual hardware may be weaker that other areas and weak gyroscope data may reduced the overall accuracy of the model for that area.

In step 3, training models are built for all character sets within an area for each area. These models also use ensembling. If the training data specified is microphones + gyroscope, then training models for microphones are also built. Our system evaluates these models using multi-fold cross validation using varying fold values. In multi-fold cross validation, a subset of the training data is provided to the model as test data and the accuracy of the model is computed. By using multiple folds, a model can be tested multiple times with different training data and their accuracies are averaged. Our system determines which models are better for an area and uses these for predictions for that area.

In step 4, the two best algorithms for an area (determined in step 3) are combined into a single voting algorithm. This voting algorithm is used by the meta-algorithm to make a final prediction in case all previous prediction mechanisms do not conclude on a single prediction. In case the two models predict different keys, their confidence values are used to determine the final prediction.

4.4 The Area-Based Meta-Algorithm

Initially, we used elementary algorithms and voting schemes to make predictions on test data. Using these elementary algorithms, the accuracy achieved was not high (See Table 1), and different algorithms predicted different keys for the same test data. To address the problem, we developed a meta-algorithm that utilizes our area specific models at multiple levels to infer the key with a higher accuracy rate than traditional algorithms. The evaluation we performed shows that the meta-algorithm yields much better accuracy.

The Inference Process: Figure 8 shows the flow diagram of our meta-algorithm and the levels of evaluations. In step 1, the test data is evaluated using the area voting model. The goal of this step is to identify the area where a keystroke would belong to and load the appropriate models for that area. Our system evaluates the models built for every area and maintains a list of the models that have yielded high accuracy with the training data for that area. For example, an area ‘IOP’ on the QWERTY keyboard may have weak gyroscope data implying that the consolidated model will be weak. The system detects this and loads the microphone model for evaluation instead of the microphones + gyroscope combined model.

In step 2, the test data is evaluated using the loaded character set models. We use these models before the area-specific models to ensure that a prediction error by the voting model is detected and corrected. For example, the voting model may predict the area of a test sample on a QWERTY keyboard as ‘IOP’ when the key pressed was ‘K’. One reason for this may be weak or noisy gyroscope data in the voting model that is similar to keys in area ‘IOP’. Our system will load the microphone models when the gyroscope data is weak. These models can then evaluate the test inputs correctly based on the audio data. If more than 75% of the models predict the same key, then this key is chosen as the final prediction and no additional steps are performed.

In step 3, the test data is evaluated using the area-specific models. This step is only executed when the character models were not successful in predicting the keystroke. One reason can be the prediction of neighboring keys that belong to another area. For example, a character set based model may predict the key as ‘A’ when the actual key is ‘Q’. These keys are neighbors on a standard Android QWERTY keyboard and may contain similar vibrations and audio characteristics. When the test data is evaluated specifically using the models for area ‘QWE’, then they can only predict a key from the selected area and may predict the correct key.

In step 4, the test data is evaluated using a voting model consisting of the two best algorithms for that area. The model determines the final prediction based on the prediction and confidence values of the two algorithms. This step will generally be executed when the test data is quite noisy and difficult to infer. We do not discard the data but attempt to make a final prediction based on the two best algorithms for that area.

5. EVALUATION

We evaluate our keystroke inference system using the following three metrics: The performance of the gyroscope, microphones and microphone + gyroscope (combined) sensors for different areas of the keyboard, the performance of our meta-algorithm applied to different machine learning algorithms in comparison to the traditional use of these al-
algorithms, and the performance of our meta-algorithm on the sample sets that were collected.

5.1 The Meta-Algorithm Evaluation

Table 2 shows the area-wise accuracy of the gyroscope (Gyro), microphone (Mics) and combined (Comb) sensors for a sample set collected using the QWERTY keyboard in portrait mode on the HTC One. The evaluation is based upon the accuracy with which characters were predicted correctly in different areas of the keyboard. We can see that the gyroscope predictions are inconsistent across areas as compared to the microphone which is consistent throughout. The gyroscope results are location dependent because of its hardware location. The areas IOP, ASD and NM are close to the gyroscope and do not exhibit significant vibration on the y axis. The inference depends more on the x axis vibrations yielding lower accuracy for these areas. The areas XZ, ASD, and QWE are further from the gyroscope and exhibit significant vibrations in the y axis. The inference depends on both axes vibrations yielding higher accuracy for these areas. Microphone predictions, on the other hand are location independent as they rely on the speed of sound traveling over the surface. We also see that the accuracy of the microphones is higher than the gyroscope for most of the areas, the only exception being area XZ. Combining the data from the two sensors yields higher accuracy than the individual sensors in some cases when the gyroscope data for an area is not weak. In a situation where the gyroscope data is weak for an area, our system attempts to detect this and performs inference using microphone data for that area.

<table>
<thead>
<tr>
<th>Area</th>
<th>Gyro</th>
<th>Mics</th>
<th>Comb</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q, W, E</td>
<td>84%</td>
<td>90%</td>
<td>92%</td>
</tr>
<tr>
<td>R, T, Y, U</td>
<td>86%</td>
<td>86%</td>
<td>92%</td>
</tr>
<tr>
<td>I, O, P</td>
<td>79%</td>
<td>90%</td>
<td>99%</td>
</tr>
<tr>
<td>A, S, D</td>
<td>84%</td>
<td>94%</td>
<td>97%</td>
</tr>
<tr>
<td>F, G, H</td>
<td>70%</td>
<td>89%</td>
<td>92%</td>
</tr>
<tr>
<td>J, K, L</td>
<td>71%</td>
<td>84%</td>
<td>89%</td>
</tr>
<tr>
<td>X, Z</td>
<td>88%</td>
<td>80%</td>
<td>98%</td>
</tr>
<tr>
<td>C, V, B</td>
<td>83%</td>
<td>93%</td>
<td>93%</td>
</tr>
<tr>
<td>N, M</td>
<td>77%</td>
<td>90%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 2: Area-wise accuracy of QWERTY keyboard sample set.

We also evaluated our keystroke inference system in environments when such an attack would not work so well. The gyroscope inference accuracy will be affected when the vibrations recorded during typing are noisy such as when typing while on a running vehicle, trembling hands during typing or when the touch is too soft. The microphones inference accuracy will be affected when the background noise is too high or when the touch is too soft for the microphones.
to capture. In our experiments, we asked two participants to type in a noisy restaurant environment and achieved an accuracy of 42% and 56% for 212 and 226 test samples using the microphones, respectively. There were two participants who touched the screen very softly, and for them, our system achieved a low accuracy of less than 20% for both the microphones and the gyroscope. This was mainly because the keystroke could not be differentiated from the background noise. We also asked two participants to type on a tablet and achieved an accuracy of 36% and 45% for 106 and 234 test samples, respectively, using the gyroscope. These participants held the tablet in two hands and typed using their thumbs significantly reducing the vibrations caused due to typing.

5.2 \textbf{End-to-end Attack Evaluation}

To illustrate an end-to-end attack, we have also implemented a Trojan-like functionality in our Android application. The application starts a background service that queries for the foreground activity every five seconds. In Android, every UI page is known as an activity. An application may have multiple activities and every activity has a different class name. Using these class names, an adversary can determine the functionality that an application is performing. For example, for the banking application we used, the login activity is named as com.\texttt{****.mobile.****Activity} (parts of the class name hidden here for anonymity of application). The Trojan starts recording the microphones and gyroscope during the banking application login activity or when credit card input activity is in the foreground.

We collected 100 four digit random numbers and 100 sixteen digit random numbers simulating PIN numbers and credit card numbers from the Trojan service. These were recorded when users opened a particular activity of a banking application triggering the microphone and gyroscope recording. Table 5 shows the accuracy obtained by using our meta-algorithm with these PIN and credit card numbers. For four digit PIN numbers, the system correctly predicted 376 digits out of 400 digits and 8 additional keystrokes were detected by the system. Out of the 100 PIN numbers, 84 were predicted completely correctly in the first attempt. For sixteen digit credit card numbers, the system correctly predicted 1467 digits out of 1600 digits and 12 additional keystrokes were detected by the system. Out of the 100 credit card numbers numbers, 52 were predicted completely correctly in the first attempt.

Table 5: Final meta-algorithm accuracy for 100 PIN numbers and 100 credit card numbers.

<table>
<thead>
<tr>
<th>Total</th>
<th>PINs</th>
<th>Correct</th>
<th>Correct Digits</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>84</td>
<td>376</td>
<td>94%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>52</td>
<td>1467</td>
<td>91.5%</td>
<td></td>
</tr>
</tbody>
</table>

We, thus, show that by building area specific models combined with meta-techniques, it is possible to achieve high accuracy of predictions such as 90-94% for the QWERTY and Number keyboard.

6. \textbf{MITIGATION}

The Android platform uses a variety of security and defense mechanisms against application’s misbehavior, such as sandboxing and permissions. The security and effectiveness of such techniques have been studied in previous works [26], [17], [3], [12], [22], [10]. Cai et al. [6] discussed properties of a privacy protecting sensors. Although these mechanisms have proved to be effective against a large number of attacks, they are not effective against side channel attacks that bypass them.

We broadly classify mitigation techniques against side channel attacks as blocking or limiting accuracy [24], [7].

\textbf{Blocking:} When a sensitive application starts running, it will obtain a lock on mutually exclusive sensors and hardware that are only accessible to one process (app). Some sensors, such as the microphone and camera, fall into this classification. During this period when the application is using these sensors, no other process will be able to use them. Blocking is straightforward to design and implement, e.g. in the current Android SDK, this can be done by using the system calls in \texttt{android.media.AudioRecord} and \texttt{android.media.MediaRecorder}. However, this mechanism is not practical against non-mutually exclusive sensors, such as the gyroscope and accelerometer, without significant user experience degradation.

\textbf{Limiting accuracy:} The inference accuracy for both the microphones and gyroscope is highly correlated with the sampling rate. Table 6 shows the impact of the sampling rate on these sensors. We see that by reducing the sampling rate of the gyroscope from 100 Hz to 56 Hz in sample sets collected by the same user, the inference accuracy reduced from 79% to 58%. By lowering the sampling rate to 20 Hz, most keystroke vibrations were not detected yielding a low accuracy of 18%. Similarly, by reducing the sampling rate of the microphones from 48 KHz to 22.05 KHz, the inference accuracy reduced from 91% to 31%. As mentioned in Section 3, the sampling rate of the sensors can be reduced by introducing services that use more processing power, however, an adversary can still obtain high and constant sampling rate by using the Android NDK.

Table 6: Impact of sampling rate on inference accuracy for Number keyboard sample set.

<table>
<thead>
<tr>
<th>Sampling Rate</th>
<th>Gyroscope</th>
<th>Microphones</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 Hz</td>
<td>79%</td>
<td>48000 Hz</td>
</tr>
<tr>
<td>56 Hz</td>
<td>58%</td>
<td>44100 Hz</td>
</tr>
<tr>
<td>20 Hz</td>
<td>18%</td>
<td>22050 Hz</td>
</tr>
<tr>
<td>48000 Hz</td>
<td>91%</td>
<td>89%</td>
</tr>
<tr>
<td>44100 Hz</td>
<td>89%</td>
<td>31%</td>
</tr>
<tr>
<td>22050 Hz</td>
<td>31%</td>
<td></td>
</tr>
</tbody>
</table>

7. \textbf{RELATED WORK & DISCUSSION}

Cai & Chen [5] were the first to study the feasibility of number keystroke inference attacks using an Android device’s orientation sensor. They developed an Android application called TouchLogger and collected three data-sets on a HTC Evo 4G phone using a Number only keypad in Landscape mode. Their experiments achieved a successful inference accuracy of about 70% on all three data-sets and showed that such an attack was indeed feasible.

Owusu et al. [23] studied the feasibility of character and area inference using an Android device’s accelerometer sensor. They developed an Android application called ACCes-
In this paper, we investigated the feasibility of keystroke inference on a smartphone by recording the sounds of key taps by the stereo-microphone and the vibrations by the gyroscope. In the future, we plan on implementing mitigation techniques for these side-channel attacks on Android and evaluate their effectiveness on several smartphones.

9. ACKNOWLEDGMENTS

We would like to thank Professor Kevin Butler for his helpful comments on our paper. This material is based upon work partially supported by the National Science Foundation under Grant No. CNS 1409453.
10. REFERENCES


