ACCessory: Password Inference using Accelerometers on Smartphones*

Emmanuel Owusu, Jun Han, Sauvik Das, Adrian Perrig, Joy Zhang
{eowusu, junhan, sauvik, perrig, sky}@cmu.edu
Carnegie Mellon University

ABSTRACT

We show that accelerometer readings are a powerful side channel that can be used to extract entire sequences of entered text on a smartphone touchscreen keyboard. This possibility is a concern for two main reasons. First, unauthorized access to one's keystrokes is a serious invasion of privacy as consumers increasingly use smartphones for sensitive transactions. Second, unlike many other sensors found on smartphones, the accelerometer does not require special privileges to access on current smartphone OSes. We show that accelerometer measurements can be used to extract 6-character passwords in as few as 4.5 trials (median).

Keywords

Mobile Phone, Sensor Malware, Side-Channel Attack, Keystroke Inference, Accelerometer

1. INTRODUCTION

Smartphones are ubiquitous. An ever-expanding consumer base carries their handsets everywhere. However, this rapid growth comes with new risks. While the proliferation of smartphones equipped with high-resolution sensors has afforded developers an opportunity to create highly interactive applications, users now rely on their smartphones to perform many privacy-sensitive tasks, such as online financial transactions and personal communications, that can be eavesdropped or exploited. In this paper, we argue that current security measures in mobile platforms do not adequately address the malware that exploits these high-resolution sensors.

Current smartphone platforms allow developers access to certain hardware sensors (e.g., accelerometers) without requiring special privileges or explicit user consent. The security risks posed by microphones and cameras have been well documented [18, 21]. However, the security risks of accelerometers have so far been largely ignored. This lack of concern might be because accelerometer values are perceived as benign. However accelerometers can expose privacy-sensitive information to malicious applications, as the *TouchLogger* [5] and *ACComplice* [22] systems demonstrate.

The ability to eavesdrop on activity occurring in the foreground is clearly a serious security breach that undermines the OS's process isolation architecture. As in the work by Cai and Chen [5], we focus on inferring user keystrokes using only data acquired from the accelerometer in an Android smartphone device.

Contributions: We show that accelerometer readings are sufficient to extract sequences of entered text on smartphones. We create and evaluate a predictive model, trained only on acceleration measurements, of the security-sensitive task of password entry. We present findings on the inference accuracy as a function of the sampling frequency of the accelerometer, the on-screen location of the keypress, and the size of the predicted screen region. Additionally, we present measures for mitigating this side channel.

Threat Model: An application running in the background can collect accelerometer output which violates the process isolation architecture that comprises much of Android OS's security policy. In fact, the accelerometer is a side channel that a background application can exploit. Malware can observe user actions within a sensitive context, such as account login, to deduce what context was active and subsequently infer the user's interactions.

We assume that the adversary can execute applications on the mobile device without special privileges beyond the permission to send information over the network. This access may be easily acquired by emulating a popular application that many users download, which argues for a legitimate reason to obtain access to network communication. For example, users may not be concerned about permitting network connectivity to a game application that uploads high scores or downloads ad content. In fact, there have been reports of malicious apps that were mere copies of existing apps with virus code appended [1, 20]. The accelerometer side-channel attack is more insidious since there exists no discernible code that clearly violates the OEM's terms of use within the malicious app.

We assume that the OS is not compromised so that the malicious application simply executes as a standard application. We also assume that the adversary's application executes in the background and logs accelerometer readings. Based on the leaked information, the adversary can infer the foreground context and entered text from the compromised device through data analysis.

2. BACKGROUND

To investigate the link between cyber security and physical security, we transform a seemingly harmless application on Android Marketplace into a potential spying device. We study these issues using the accelerometer on Google's Android OS application architecture.

2.1 Accelerometers

The accelerometer measures the acceleration of the device on the *x*

*This research was supported by CyLab at Carnegie Mellon under grants DAAD19-02-1-0389 and W911NF-09-1-0273, from the Army Research Office, and by support from NSF under TRUST STC CCF-0424422, IGERT DGE-0903659, and CNS-1050224, and by a Google research award. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of ARO, CMU, Google, NSF or the U.S. Government or any of its agencies.

HotMobile'12, February 28–29, 2012, San Diego, California, USA.

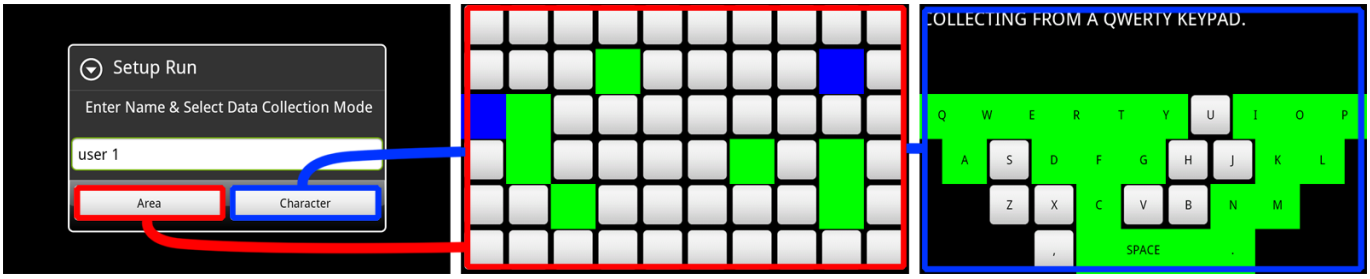

Figure 1: These three images are screen captures of the Android data collection application. The user is prompted to select one of two data collection modes: *area* or *character* mode. We analyze *area* mode data first to inform the subsequent *character* mode analysis.

(lateral), *y* (longitudinal), and *z* (vertical) axes. The most commonly used accelerometer within smartphones is the LIS311DLH. At 3x3x1 mm, it is a tiny, low power, high performance linear accelerometer. It senses the forces of acceleration in the *x*, *y*, and *z* planes to a precision of six decimal places in units of m/s^2. Other accelerometers embedded within Android devices are of a similar resolution. Given the high resolution of these devices, it is possible to discern patterns in the acceleration magnitudes for a variety of activities.

Researchers have used accelerometers in a diverse set of applications. *Smart-Its Friends* [9] and *Are you with me*? [11] use accelerometers for device pairing (i.e., to identify when users are holding their phones together). Activity recognition via acceleration data has been a well researched topic [4, 12, 14, 17]. Additionally, many popular smartphone applications utilize accelerometers. *Bump* allows two users to easily connect and share information by tapping their smartphones together.[1] *Smart Alarm Clock* identifies an ideal time for a user to wake up by monitoring his or her sleep cycle.[2]

2.2 Android Platform

On the Android platform, each application runs as a dedicated Linux process, and each process has its own virtual machine. This isolation architecture sandboxes one application from all others. For additional security controls, the Android platform maintains a list of protected device features. An application that needs to access one of these protected features (e.g., sending a SMS) or interact with another application must declare its intentions by specifying the corresponding "permission" in an Android Manifest file. Users are then notified about the permissions requested by the application at install time. The desired security goal for this isolation architecture is that no application can adversely affect other applications or the user.[3]

However, the accelerometer is not considered a "protected feature." Consequently, applications do not require user permissions to access the sensory outputs of the accelerometer. Further, the user need not be made aware of an application's solicitation of the accelerometer. The fact that no special permissions are needed to access accelerometer data indicates that acceleration readings are perceived to abide by the isolation architecture's policy that no application can infer information that is not specifically permitted. Yet, as we show in this paper, accelerometer readings can potentially reveal highly sensitive information.

3. IMPLEMENTATION

We present a principled machine learning approach for extracting text entered into a smartphone via accelerometer measurements. This attack is a multi-step process involving keypress segmentation, probabilistic keypress classification, and finally sorting keystroke sequences by maximum likelihood. In this paper, we focus on describing the latter two steps of the attack because key segmentation from an acceleration stream is relatively straightforward using root-mean-square anomalies for spike detection. Using this simple method, we were able to obtain 94-96 percent segmentation accuracy and estimate that we can diminish the error further by using more sophisticated segmentation techniques. Thus, assuming segmented keypresses, we present how to identify the affected region of the touchscreen and map the predicted region to an on-screen keyboard. Then, we present the use of a probabilistic error model constructed on training data to sort keystroke sequences by maximum likelihood.

One major challenge we face in this attack is that an error at any step in the process cascades, ultimately corrupting predictive accuracy. To test the feasibility of this attack, we implement a virtual testbed application to obtain ground truth acceleration streams corresponding to key presses at specific screen regions, and analyze these streams for predictive precision, recall, and accuracy.

3.1 Application Design

We implement an Android application for collecting accelerations while a user types. The application is compliant with the Android developer terms of use in that our application does not do anything explicitly unauthorized. The application only requires network access for offloading the collected data.

Our application has two collection modes: *area* and *character*. The *area* mode interface is completely populated by a 10x6 array of buttons. The buttons are similar in size to the buttons found on the standard Android OS soft keyboard. We develop the *area* mode data collection screen for two main purposes: (i) to evaluate the inference accuracy at varying levels of granularity to gain insight about the amount of information noisy accelerometer data streams reveal about keystrokes; and (ii) to quantify our intuition that certain regions of the screen leak more information about keystrokes given a typing style. The *character* mode interface consists of a QWERTY keyboard arranged similarly to the Android soft keyboard screen. The *character* mode is the testbed for our keystroke reconstruction attack. Figure 1 shows screen captures of the data collection application.

As the application runs, a new record is appended to a log file for every updated accelerometer reading. Each record includes a timestamp, acceleration measurement, and a label unique to the pressed key, if any. The accelerometer readings are triaxial[4] and are measured in units of meters per second squared (m/s^2). We monitor

[1]The *Bump* app sends motion sensor data to a server where a matching algorithm pairs phones experiencing the same "bump" motion. `http://bu.mp/faq`

[2]The *Smart Alarm Clock* app tracks body movements during sleep via the accelerometer to calculate the optimal time to wake an individual. `http://blog.smart-alarm-clock.com/`

[3]Android Security Architecture. `http://developer.android.com/guide/topics/security/security.html`

[4]On smartphones running Android, the coordinate axes are defined relative to the screen in its default orientation. For this discussion, we define the x-axis, y-axis, and z-axis as the axis along the short side, along the long side, and going through the screen, respectively.

key-pressed and key-released touch events as they are dispatched by each button to establish the ground truth for our subsequent analyses.

As our test data consists of acceleration readings that are logged while users complete simulated typing tasks, we implement an evolving color-coded scheme as a measure of progress for the user and to ensure that every region of the screen receives a sufficient number of samples: a button will turn green after the first press, blue after five presses, and black after ten presses.

3.2 Data Analysis

At a high level, our data analysis consists of three main phases: *preprocessing*, *feature subset selection*, and *classification*. We perform classification for *area* and *character* mode separately–analyzing *area* mode data first to inform subsequent *character* mode analysis.

Data Collection: Keypress data for both experiments was collected from 4 participants using the HTC ADR6300 handset model. Participants were between the ages of 18 and 30, and were all regular smartphone users. For the *area* mode inference experiment, we collected data for about 1300 keypresses, where each one of the 60 screen areas received approximately 20 positive samples. Before each data collection run, participants were instructed to press keys in any order until all of the keys received at least one press. Testing was done using k-fold cross-validation. For the *character* mode inference experiment, we collected training data for about 2700 keypresses–again, ensuring that all keys received a similar number of positive samples. Participants were instructed to either enter sentences or to press keys in any order until all keys had received at least one press. For testing, we logged data for 99 6-character passwords.

In practice, one would need to account for differences in typing styles; however, addressing this is beyond the scope of our present research. To ensure consistency across tests, we instructed all participants to: (i) hold the device in the landscape orientation using both hands and (ii) enter text using thumbs. We discuss several sources of variability, including typing styles, in more detail in §5.

Preprocessing: Since the Android platform dictates when an application receives updates for acceleration measurements, we use linear interpolation to obtain consistent sampling intervals throughout the dataset. Interpolation also provides data smoothing which mitigates baseline signal noise in the data by suppressing moment-to-moment spikes. We find that the average sampling rate of our devices' accelerometer is about $50Hz$ and continue our analyses with this baseline sampling rate. Preprocessing the raw acceleration measurements generates a modified dataset $M = \{x_i, y_i, z_i, m_i\}_{i=1}^{s}$ where s is the number of samples in our dataset, x_i, y_i, z_i represent the acceleration along the respective axes, and m_i is the Euclidean magnitude of the acceleration.

Feature Selection: For each preprocessed acceleration stream, we generate the set of 46 features detailed in Table 1. The first eleven features in Table 1 summarize the acceleration stream information for each dimension separately, and thus comprise 44 features in total. The last two features, *total time* and *window size*, are summarized meta-information about the key stream–the duration of the stream and the number of samples in the stream, respectively.

We chose these features partially based on the results of prior work [2, 5] that deals with triaxial accelerometer feature extraction and partially based on insights gained from exploratory observations of the data. To ensure the selection of an optimal set of features for our classification task we employ the *Wrapper* feature subset selector [10]. The *Wrapper* algorithm is coupled with a classification algorithm and exhaustively searches through the feature space for the set of features that maximizes a pre-specified "evaluation measure"–in our case, the "evaluation measure" is the cumulative classification accuracy of our model. Because we execute *Wrapper* for multiple classification tasks we omit a final list of selected features as these vary by the task domain.

Feature	Description	D/M
RMS	The Root-Mean-Square value	D
RMSE	The Root-Mean-Square error	D
Min	The minimum value	D
Max	The maximum value	D
AvgDeltas	The average sample-by-sample change	D
NumMax	The number of local peaks	D
NumMin	The number of local crests	D
TTP	The average time from a sample to a peak	D
TTC	The average time from a sample to a crest	D
RCR	The RMS cross rate	D
SMA	The Signal Magnitude Area	D
Total Time	The Total Time of the window	M
Window Size	The number of samples in the window	M

Table 1: This table contains the list of features used to summarize acceleration stream values by window. Dimensional features (D) are calculated separately for each dimension (x, y, z) as well as the Euclidean magnitude of acceleration (m). Meta features (M) describe the window features of the acceleration stream and are calculated only once per feature vector.

It is worth noting that our analysis took on two forms: "continuous" and "aggregate." In the continuous form, we employ feature extraction for each sample of each acceleration stream and treat the analysis as a sequence classification problem. In the aggregate form, we treat each acceleration stream corresponding to a keypress as a single entity and extract features that describe the entity as a whole. Empirically, we find that the aggregate form yielded substantially better results. We therefore advance this discussion with only aggregate features in mind.

Classification: At the highest level we want to identify which acceleration streams correspond to specific touch events. We take as input a feature vector, F, and output a prediction label, P, that corresponds to the ground truth label, G. For all instances where $P = G$ we consider our model to have accurately "classified" the instance. For the *area* mode analysis, we formulate the task of identifying keystrokes as a hierarchical classification problem. We recursively partition each area of the screen into two parts, and then classify individual keys within each new subarea. We can calculate the per-key accuracy of each subdivision by taking the cumulative product of the accuracies for each subdivision and the final per-key classification of any given subarea. To ensure optimal accuracy at every division, we perform a new feature subset selection at each division. We perform these divisions at the $1/2$, $1/4$, and $1/8$ granularities and compared the per-key accuracy findings to discern the optimal scheme for maximum classification accuracy.

We find that the Random Forest [8] classification algorithm yields the best results among a variety of candidates, including a multilayer perceptron artificial neural network, sequential minimal optimization trained Support Vector Machine [15], and C4.5 decision tree [16]. The Random Forest algorithm creates a set of n C4.5 decision trees, training each with a random subsample of the given dataset. A split at each node in a C4.5 trees is determined by looking at c randomly selected features of the given feature space. A classification for a given instance (in our case, the feature vector of an acceleration stream, F) is made by pushing the instance down every tree in the forest and counting the label of the training instance that the testing instance matches as the "vote" for a particular tree. The votes are then tallied, and the most frequent vote among the trees is considered the prediction of the forest. We suspect that Random Forests performed well because of the propensity for significant variability of feature values between instances of the same label. Random Forests create multiple trees based on a varying subset of features, making it more robust than most other classifiers to intraclass variability.

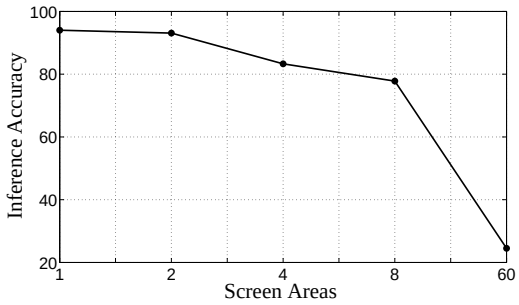

Figure 2: Inference accuracy by screen region granularity. The screen surface is partitioned into successively smaller blocks and evaluated for classification accuracy.

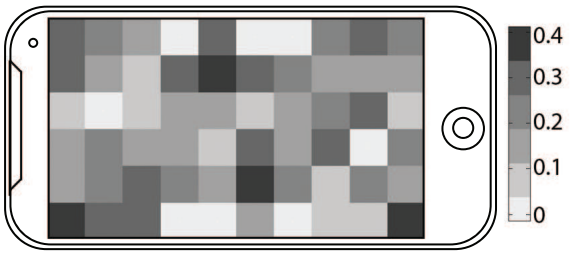

Figure 3: This heat map shows the relative prediction accuracies for each region on the device screen, where darker shades imply greater predictive accuracy.

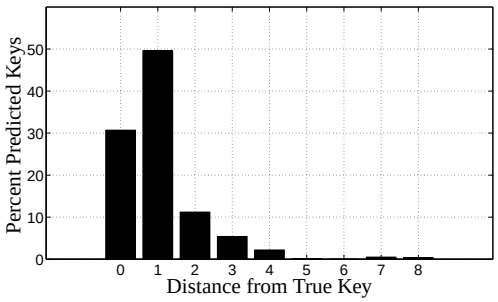

Figure 4: This histogram shows the distribution of distances of the predicted screen region from the actual screen region. Notice that mispredictions are localized.

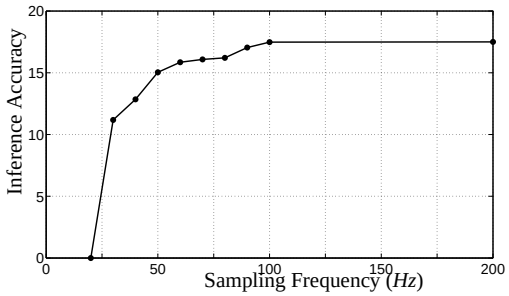

Figure 5: Inference accuracy by accelerometer sampling frequency. Note that these values are not best case estimates, as we do not optimize our model for predictive accuracy in this analysis.

4. EVALUATION

4.1 Study 1: Area Mode Inference

In this study, we logged accelerometer data for about 1300 key presses. We recorded approximately 20 samples for each screen region. Below, we report on the results of our hierarchical classification model.

Hierarchical Classification: The results of our hierarchical classification scheme are summarized in Figure 2. The classification accuracies reported for each recursive screen region split are obtained by averaging the results over ten runs of stratified 5-fold cross-validation. Notice that we obtain high fidelity classification accuracies for large screen region splits (e.g., 93% for halves) that tapers off as we increase the granularity of our predictions. By multiplying the successive division probability, we empirically find that splitting the screen into eighths and then classifying individual region keys yields the greatest average individual key accuracy of approximately 24.5%.

Figure 3 shows a heat map of the relative predictive accuracies of our model across the surface of the device screen. Darker shades in the heat map imply greater predictive accuracy for a specific key region. We can see that, for the two-handed typing style, our model obtains relatively high accuracy along the diagonal regions of the screen, with other regions of high accuracy concentrated in the center area of the screen. From this figure, we know which areas of the screen yield the highest predictive accuracy, which helps inform a probabilistic ranking of keystroke sequences.

Error Localization: Figure 4 shows the distribution of classification errors in units of "key distance" from the ground truth. We define a unit of key distance as the level of adjacency of one key to another. For example, if a key *b* is directly adjacent to another key *a*, we say that *b* is 1 key distance away from *a*. However, any key *c* that is adjacent to *b* but not *a* is said to have a key distance of 2 from *a*. We can see from Figure 4 that the vast majority of our model's predictions are concentrated within an error of 0 or 1 key distance (80%). Furthermore, 91%, 96%, and 99% of our predictions are within 2, 3, and 4 keys, respectively. Thus, we conclude that most of our model's mispredictions are localized around the ground truth. This localization informs a probability distribution of likely mispredictions that we exploit to rank-order keystroke sequence predictions.

Sampling Rate: Figure 5 shows how the predictive accuracy of our model varies in relation to the simulated resampling frequency of our dataset. We observe that as the sampling rate increases to 100*Hz*, we obtain significantly higher individual key accuracies. This finding informs a potential resolution to the security concerns raised by our findings: by enforcing a low sampling rate on the accelerometer for untrusted applications or background applications, one can substantially mitigate the predictive accuracy of keystroke inference models.

4.2 Study 2: Character Mode Inference

In a second study, we logged acceleration readings during simulated password entry tasks. We then performed data analysis using the approach described in §3.2 to probabilistically generate a ranked list of the entered text.

Data Set: We collected the dataset with measurements logged from the *character* mode screen. We collected a training set consisting of about 2700 keypresses. The training dataset is evenly split between pangram tests (where we instruct participants to enter sentences containing all letters of the alphabet), and coverage tests (where we instruct participants to randomly touch keys until all keys have received at least one press) collected from three participants. The testing dataset consist of 99 6-character passwords collected from the remaining participant. The passwords are based on a public list of common passwords leaked from several databases.[5]

Prediction Ranking: The translation model produces a conditional probability mass distribution, $P(P_i|G_j)$, of predicted keys P by actual keys G–a matrix containing the frequency of the actual keys pressed for every prediction made by the model during training. We

[5]`http://blog.jimmyr.com/Password_analysis_of_databases_that_were_hacked_28_2009.php`

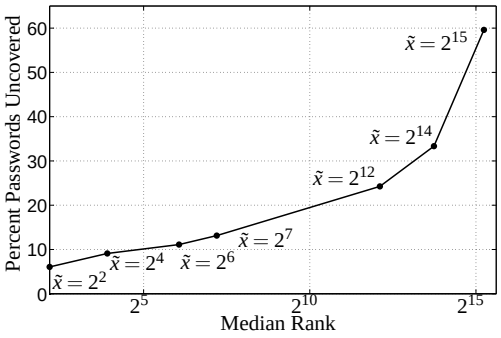

Figure 6: The percentage of passwords cracked plotted against the median number of trials required to extract those passwords.

utilize the probability mass distributions to generate a ranked list of candidate passwords by running a maximum likelihood search for the most probable classification errors for each keystroke sequence.

We operationalize the term "trial" as a single attempt at cracking a password. Thus, in the case of our attack, a trial would refer to traversing down a single step of the rank-ordered list produced by our maximum likelihood search. As evident in Figure 6, our model is able to correctly deduce 6 of the 99 passwords in a median 4.5 trials. Compare these results to the assumed case, that accelerometer data provides little information about texts entered on the phone and therefore does not violate the process isolation property. A brute force attack on a 6 character length password takes approximately 2^{28} trials on average. The model presented here cracked 59 of 99 passwords in approximately 2^{15} median trials. From these results we conclude that accelerometers can be used to significantly reduce the search space for text entered on smartphones, resulting in the ability to extract sensitive information, such as account passwords, from the user.

5. DISCUSSION

Given that the data collection presented in this report is controlled, the question remains: How feasible would an accelerometer attack be in a real world setting? We discuss how our model can be extended to handle several sources of uncertainty present during keystroke inference, as well as a few potential mitigation strategies.

Sources of Variability: A real-world implementation of this attack would have to address several sources of variability such as different hand sizes, typing styles, screen sizes, and keyboard user interfaces. Nevertheless, we do not claim that our models are generalizable to address these issues. Individual calibration will likely always be a necessity, but we believe our model can be extended to address these and other sources of variability given the appropriate training data.

In general, though there certainly exists subtle variabilities, most people enter text on smartphones in a similar fashion. We observe several categories of typing styles. For example, some people prefer to hold a phone with one hand while using the index finger of the other hand to enter text, while others prefer to hold a phone with both hands and enter text using their thumbs. Typing style also depends on whether the phone is held in a portrait or landscape orientation. A sample of the acceleration measurements can be used to identify the holding style of the user. This enables the adversary to switch to the appropriate translation model.

Some variables can be identified directly by the running application (e.g., the screen size and keyboard UI). In the case of variability that is not easily identified from the acceleration measurements or by the application, the adversary can train the model separately for each configuration and use the results of the model that produces the most sensible output. We make simplifying assumptions because the goal of our work is to show that such attacks are feasible due to the nature of information leaked by accelerometers.

Real-World Threat: Major websites typically limit the number of retries for entering a password. Our results indicate that a small fraction of passwords can be cracked in a limited number of trials (e.g., 1 of 99 passwords was cracked in 1 attempt and 6 of 99 in 4.5 median attempts). Attackers can perform this attack in a scalable manner where cracking just 1% of passwords can be lucrative.

Our model makes no assumptions about the text being entered– it simply maps sensor data to screen locations to keys. This attack can be used to extract other types of text, such as text messages and e-mail, where a perfect translation is not necessary to produce an approximate text representation and enable the extraction of sensitive information. Furthermore, the structured form of natural-language text makes it vulnerable to further analysis (e.g., analysis that uses a dictionary-backed sequencing model, such as the *n*-gram language model, to further disambiguate the entered text.

There do exist simpler methods for attackers to obtain sensitive information from smartphones. We discuss some of these threats in §1 and §6. However, the accelerometer is a particularly interesting case since it presents a physical side channel that cannot be easily shielded or detected. It is prudent to consider an adversary with more resources that is willing to invest the extra time needed to develop a robust eavesdropping application with these stealthy properties. Our model represents a proof-of-concept design to demonstrate that this is a real threat.

Mitigation Strategies: Several counter measures can be taken to defend against accelerometer-based side-channel attacks. OEMs can take steps to increase the accountability of application publishers in addition to increasing the likelihood that device owners identify suspicious apps. The first step is to force every application to declare their intention to access the accelerometer and other motion sensors. The next step is to inform users about applications that request potentially dangerous combinations of permissions (in our case, network access and fine-grained accelerometer data).

Furthermore, design solutions exist that complicate keystroke inference. Enforcing a limit on the sampling frequency can certainly help to mitigate the attack. As shown from §4.1, inference accuracy drops significantly at lower sampling frequencies, thus reducing the effectiveness of a keystroke inference attack. This approach, however, poses potential problems for legitimate applications requiring fine-grained acceleration data. Another approach is to prevent background applications from accessing fine-grained accelerometer readings, e.g., by reducing the sampling rate to 2 samples per second. Other design solutions include varying the keyboard layout during sensitive tasks such as password entry (e.g., altering the placement of individual keys, or the entire keyboard) or to automatically initiate the phone vibrator at random intervals during text entry. These strategies would provide protections at the expense of usability.

6. RELATED WORK

We discuss related research in the areas of mobile sensor malware, keystroke inference, and accelerometers.

Researchers have studied malicious code on mobile devices that learn information about the device owner using other embedded sensors. For example, Schlegel et al. exploited an application that can access the smartphone's microphone to eavesdrop on sensitive communications such as speech, the destination caller during tone dialing, or menu selections based on sound emanations [18]. Xu et al. considered malicious applications that used the camera on smartphones [21]. Cai, Machiraju, and Chen studied privacy risks that stem from mobile phone sensors, but they only considered microphone, camera, and GPS data, not accelerometer data [6].

Several researchers have studied the extraction of entered text from PC and laptop keyboards–e.g., based on sound [3, 23] and inter-

keystroke timing observations [19]. *(sp)iPhone* uses motion sensors in the iPhone 4 to infer keystrokes of nearby laptop users [13].

Researchers have used accelerometers to infer other types of information about a device owner. Chang et al. used accelerometers embedded in television remote controls (as well button press sequence features) to distinguish between household members [7]. Liu et al. developed *uWave*–software that uses triaxial accelerometer data on mobile phones for highly accurate gesture recognition [12]. Our prior work, *ACComplice*, uses accelerometers on smartphones to infer the location of a moving vehicle within several minutes of drive time [22].

The most closely related work is *TouchLogger*, which also proposes to infer keystrokes based on accelerometer readings [5]. While a promising first step, it remains unclear how powerful accelerometers readings really are, due to *TouchLogger*'s lack of sequencing inference, and the coarse level of granularity by only distinguishing amongst 10 large screen areas. We evaluate this side channel at the granularity needed to make inferences about entered text on standard smartphone keyboards–*Touchlogger* distinguishes only 10 keys whereas *ACCessory* distinguishes 29-60 categories. Additionally, we evaluate the efficacy of inferring a sequence of keys instead of the per-key accuracy and present an approach for mitigating errors during keystroke sequencing.

7. CONCLUSION

As we demonstrate in this paper, accelerometer readings are highly security sensitive. We show how a background application can use the accelerometer as a side channel to spy on keystroke information during sensitive operations such as account login. We are able to break 6 of 99 six-character passwords in as few as 4.5 attempts (median). Allowing for more attempts, we successfully break 59 of 99 passwords using only accelerometer measurements logged during text entry. Given our results, it is clear that future versions of mobile devices need to restrict access to accelerometer information as strictly as access to microphone and camera sensors' information.

8. REFERENCES

[1] Android Hit by Rogue App Malware. BBC: `http://www.bbc.co.uk/news/technology-12633923`, March 2011.

[2] Felicity R Allen, Eliathamby Ambikairajah, Nigel H Lovell, and Branko G Celler. Classification of a Known Sequence of Motions and Postures from Accelerometry Data using Adapted Gaussian Mixture Models. In *Physiological Measurement*, volume 27, 2006.

[3] Dmitri Asonov and Rakesh Agrawal. Keyboard Acoustic Emanations. In *Proceedings of IEEE Symposium on Security and Privacy*, 2004.

[4] 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. Springer Berlin / Heidelberg, 2004.

[5] Liang Cai and Hao Chen. TouchLogger: Inferring Keystrokes on Touch Screen from Smartphone Motion. In *USENIX Workshop on Hot Topics in Security (HotSec)*, 2011.

[6] Liang Cai, Sridhar Machiraju, and Hao Chen. Defending Against Sensor-Sniffing Attacks on Mobile Phones. In *Proceedings of ACM Workshop on Networking, Systems, and Applications for Mobile Handhelds (MobiHeld)*, 2009.

[7] Keng hao Chang, Jeffrey Hightower, and Branislav Kveton. Inferring Identity using Accelerometers in Television Remote Controls. In *Proceedings of the International Conference on Pervasive Computing*, 2009.

[8] Tin Kam Ho. C4.5 Decision Forests. In *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, 1995.

[9] Lars Erik Holmquist, Friedemann Mattern, Bernt Schiele, Petteri Alahuhta, Michael Beigl, and Hans-W. Gellersen. Smart-Its Friends: A Technique for Users to Easily Establish Connections between Smart Artefacts. In *Proceedings of Ubicomp*, 2001.

[10] Ron Kohavi and George H. John. Wrappers for Feature Subset Selection. In *Artifical Intelligence*, volume 97, pages 273–324, 1997.

[11] Jonathan Lester, Blake Hannaford, and Gaetano Borriello. "Are You with Me?" - Using Accelerometers to Determine If Two Devices Are Carried by the Same Person. In *Pervasive Computing*, volume 3001 of *Lecture Notes in Computer Science*, pages 33–50. Springer Berlin / Heidelberg, 2004.

[12] Jiayang Liu, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. uWave: Accelerometer-Based Personalized Gesture Recognition and Its Applications. *Pervasive and Mobile Computing*, 5(6):657 – 675, 2009.

[13] Philip Marquardt, Arunabh Verma, Henry Carter, and Patrick Traynor. (sp)iPhone: Decoding Vibrations From Nearby Keyboards Using Mobile Phone Accelerometers. In *Proceedings of ACM Conference on Computer and Communications Security*, CCS, 2011.

[14] Uwe Maurer, Anthony Rowe, Asim Smailagic, and Daniel P. Siewiorek. eWatch: A Wearable Sensor and Notification Platform. In *IEEE Workshop on Wearable and Implantable Body Sensor Networks*, 2006.

[15] John Platt. Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. Technical report, Microsoft Research, 1998.

[16] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993.

[17] Nishkam Ravi, Nikhil D, Preetham Mysore, and Michael L. Littman. Activity Recognition from Accelerometer Data. In *Proceedings of Conference on Innovative Applications of Artificial Intelligence (IAAI)*, pages 1541–1546, 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 *Proceedings of Network and Distributed System Security Symposium (NDSS)*, February 2011.

[19] Dawn Xiaodong Song, David Wagner, Song David, and Xuqing Tian. Timing Analysis of Keystrokes and Timing Attacks on SSH, 2001.

[20] Vanja Svajcer. Malicious cloned games attack Google Android Market. Naked Security: `http://nakedsecurity.sophos.com/2011/12/12/malicious-cloned-games-attack-google-android-market/`, December 2011.

[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 ACM Conference on Wireless Network Security (WiSec)*, 2009.

[22] Jun Han; Emmanuel Owusu; Thanh-Le Nguyen; Adrian Perrig; Joy Zhang. ACComplice: Location Inference using Accelerometers on Smartphones. In *Proceedings of the International Conference on Communication Systems and Networks (COMSNETS)*, January 2012.

[23] Li Zhuang, Feng Zhou, and J. D. Tygar. Keyboard Acoustic Emanations Revisited. In *Proceedings of ACM Conference on Computer and Communication Security (CCS)*, November 2005.