Available online at www.sciencedirect.com

ScienceDirect

Procedia Technology 11 (2013) 989 – 995

Procedia Technology

The 4th International Conference on Electrical Engineering and Informatics (ICEEI 2013)

On the Best Sensor for Keystrokes Inference Attack on Android

Ahmed Al-Haiqi*, Mahamod Ismail, Rosdiadee Nordin

Department of Electrical, Electronic & Systems, Faculty of Engineering, UKM, Bangi, 43600, Selangor, Malaysia

Abstract

One of the most recently exposed security threats on smartphone platforms is the potential use of motion sensors to infer user keystrokes. Exploited as side channels, few researchers have demonstrated the ability of built-in accelerometers and gyroscopes in particular, to reveal information related to user input, though the practicality of such an attack remains an open question. This paper takes further steps along the path of exploring the aspects of the new threat, addressing the question of which available sensors can perform best in the context of the inference attack. We design and implement a benchmark experiment, against which the performances of several commodity smartphone-sensors are compared, in terms of inference accuracy. All available Android motion sensors are considered through different settings provided by the OS, and we add the option of fusing several sensors input into a single dataset, to examine the amount/lack of improvement in the attack accuracy. Our results indicate an outstanding performance of the gyroscope sensor, and the potential improvement obtained out of sensors data fusion. On the other hand, it seems that sensors with magnetometer component or the accelerometer alone have less benefit in the adverted attack.

Selection and peer-review under responsibility of the Faculty of Information Science & Technology, Universiti Kebangsaan Malaysia.

Keywords: Smartphone security; Sensor-based attack; Side-channel attack; Touchscreen taps inference; Motion sensors; Android;

1. Introduction

Key logging attacks have been, and continue to be, a major security concern in traditional computing platforms. PC operating systems, e.g. Windows, allow system message interception, thereby enabling background applications to capture and log the key strokes of active applications in the foreground. Such Trojan applications are very

* Corresponding author. Mobile: +60123034880
E-mail address: ahmadalhaiqi@gmail.com

2212-0173
Selection and peer-review under responsibility of the Faculty of Information Science & Technology, Universiti Kebangsaan Malaysia.
doi:10.1016/j.protcy.2013.12.285

common in traditional PCs, and although usually called keyloggers, many are capable of capturing more information than mere keyboard input, such as screen shots and mouse clicks. Even when keyloggers are not easy to install or hide, hardware keyboards allow for backdoor channels that could be resorted to in order to guess the actual input. Such channels are commonly termed side channels, and include physical phenomena, like electromagnetic and acoustics emanations.

Moving to smartphones, the situation is quite different. Smartphone OSes, like Android, restrict the interception of keystrokes to the current view that has the focus, and no direct method can be used by Trojan apps to log user taps (but for a workaround example, see [1]). Moreover, the lack of physical keyboards limits the side channel avenues at the disposal of attackers. For instance, the rich heritage of research on electromagnetic and acoustic emanations is not applicable on smartphones. Yet, innovative side channels to sniff on user input have been proposed in the literature, including the analysis of finger smudges left on the touch screen surface [2], employing the dial tone sounds [3], that resembles to more traditional acoustic emanations attacks, and even the old-school low-tech shoulder-surfing [4].

Embedding sensors into smartphones had made them an unprecedented platform, combining communications, computing and sensing capabilities. User interface, gaming, and healthcare are but a few domains in which sensors found instant applications [5], while a key idea around which many more applications evolve is context awareness. On the flip side, sensors bring along many serious implications, especially related to user privacy. Researchers have studied the potential threat of more traditional sensors, namely, GPS, camera and recorder, on user privacy and/or security (e.g. [6]). Less traditional sensors, such as motion sensors (e.g. accelerometers and gyroscopes), have received little attention, until recently. The key observation that moved motion sensors into the threat spot, is the correlation between user taps on touch screen and vibrations or motion changes to the body of the smartphone itself. Accelerometers, for example, can sense the linear displacements caused by the force of user taps, while gyroscopes can measure angular displacements around specific axes. Obviously, the original benign purpose of theses sensors are far from such vicious uses, and that makes them a surreptitious side channels.

Few authors in academia have demonstrated the feasibility of such a side channel attack, dealing with the task as a classification problem to map sensors reading into key labels. Section 3 presents a brief survey of these works. The availability of several sensors on consumers electronics devices, in particular smartphones, raises a seemingly interesting question of which sensor, or collection of sensors thereof, is of greater potential in the context of the new threat. In this paper, we set off to experiment with the performance of different sensors that are supported by Android operating system, and integrated in most Android-powered phones. Section 2 explains shortly the considered sensors, and how it is possible to capture their data with Android help. For the purpose of the comparison, we actually implemented the attack, collecting sensors data through an Android app. Details of the experiment are given in section 4. It is important to notice that the aim of this paper is not to improve the accuracy of the attack, or evaluate its practicality thereof, but is more about comparing the performance of sensors under the same conditions, which are well controlled. Our experiment in this light could focus on the relative performance between different datasets on the same setting, rather than looking for the best among several settings. The latter is the focus of most recent works discussed in section 3, while the early works were concerned with the bare feasibility of the attack.

2. Technical background

Android supports a variety of sensors, of which the relevant to the current attack are motion and position sensors. Table 1 lists the supported motion sensors, as of Android 4.2.2. Besides those sensors, Android also provides a synthetic sensor based on the values from the accelerometer and magnetometer, through a method call (getOrientation ()). Accelerometer and gravity sensors contain the Earth's gravity force, which is more of a bias to our experiment, and therefore not considered, as the linear accelerometer can take their role.

Table 1. Motion sensors that are supported on Android platforms (source [7]).

Sensor	Description	Units of measurement
TYPE_ACCELEROMETER	Acceleration force along the x, y and z axes (including gravity).	m/s^2
TYPE_GRAVITY	Force of gravity along the x, y and z axes	m/s^2
TYPE_LINEAR_ACCELEROMETER	Acceleration force along the x, y and z axes (excluding gravity).	m/s^2
TYPE_GYROSCOPE	Rate of rotation around the x, y and z axes	rad/s
TYPE_ROTATION_VECTOR	Rotation vector component along the x, y and z axes (axis * sin (θ/2)).	Unitless

3. Related work

Authors in [8]suggested first the use of motion sensors to infer keystrokes on touch screens. They developed an Android application, named TouchLogger, to demonstrate the attack. The application used numbers-only soft keypad in the landscape mode. TouchLogger utilized the synthetic Orientation sensor, which relies on accelerometer and magnetometer hardware sensors. Orientation sensor was deprecated in Android 2.2 (API level 8).

Following was another work[9], where an Android application, ACCessory, was built to evaluate a predictive model, trained only on acceleration measurements. ACCessory attempted to infer area zones on the screen as well as character sequences (to construct typed passwords).

The next work[10], adopted an online processing, where the training and classification were performed on the smartphone itself through a Trojan application, TapLogger, to stealthily monitor the movement changes of the device and try to log the number pad passwords, and screen lock PINs. Two sensors were used: the accelerometer for taps detection, and the Orientation sensor for tap positions inference.

The same authors of TouchLogger published another work again[11]. The purpose of the study was to provide a more thorough investigation on the practicality of such an attack, and to compare the performance of different classification schemes, and the impact of different devices, screen dimensions, keyboard layouts or keyboard types. This paper examined the use of gyroscopes output on mobile devices for the attack, and indicated that inference based on the gyroscope is more accurate than that based on the accelerometer.

TapPrints[12], the framework presented in another paper was evaluated across several platforms including different operating systems (iOS and Android) and form factors (smartphones and tablets). It also showed a combined approach that uses both the accelerometer and gyroscope for achieving better accuracy.

Finally, and most recently, the authors of [13] focused solely on the accelerometer sensor to further investigate the practicality of sensors side channels in inferring Android four-digit PINs and password pattern (swiping). Contrary to previous last two works, they found that accelerometer based techniques perform nearly as well, or better, than gyroscopic based techniques.

To the best of our knowledge, no previous study addressed the relative performance of all relevant Android sensors in conducting the inference attack. In addition, we also consider fusing data from more than one sensor for that matter.

4. Methodology

Keystrokes inference can be viewed as a machine learning problem, in particular, a classification task that maps collected patterns of raw sensor signals data into corresponding key classes. Abiding by typical machine learning process sequence, raw data are collected from the source, and pre-processed, then features are selected. Part of the resulting dataset is labeled with the correct class to form a training examples subset, and the rest is kept for evaluation purposes as a test subset. In the following subsections we present those steps in more detail.

4.1. Data collection

An application was built for the purpose of acquiring sensors data out of an Android smartphone. Our raw data are the readings of a set of four sensors, two of which are synthetic sensors, derived from a set of up to three hardware sensors, as noted in section 2. The particular device used in the experiment is a Samsung Galaxy S2. One user was utilized to type all data sets to ensure consistent typing and holding style factors that could affect the inference performance. Fig 1(a) lists the main hardware specifications of the device and its built-in sensors, while Fig 1(b) shows the typing profile of the user throughout the experiment.

a

Phone	Model number		Android version
Samsun Galaxy S2	GT-I9100G		4.0.3
sensor	**Name**	**Min delay**	**Vendor**
ROTATION_VECTOR	MPL rotation vector	10000 µs	Invensense Technology
LINEAR_ACCELEROMETER	MPL linear accl	10000 µs	Invensense Technology
GYROSCOPE	MPU3050 Gyroscope sensor	10000 µs	Invensense Technology
ACCELEROMETER	KXTF9 3-axis Accelerometer	10000 µs	Kionix Corporation
MAGNETIC_FIELD	AK8975 Magnetic field sensor	10000 µs	Asahi Kasei Microdevices

b

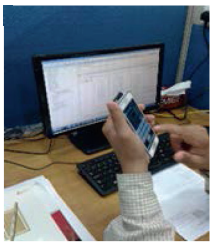

Fig. 1. Experiment settings (a) Hardware specsand (b) user typing profile.

The UI of the application, depicted in figure 2, allows for the selection of the sensor in each session, and a layout similar to the dialing soft keypad of Android 4.0.3 is presented to the user, where he was asked to key in almost the same set of 300 keys in each session. The key set covers uniformly the ten digits of the numbers soft keypad. Two datasets were generated from the same linear accelerometer, but distinct by the use of high pass filtering, an option that the user can choose by selecting a checkbox on the screen. Filtering is one of the common techniques to mitigate noise in sensors data, and in that context, low pass filters are more useful, whereas high pass filters can extract the most fluctuating components, of which we are more interested in. The interface also includes an option for fusing the data, upon which all four sensors are registered with Android, and the readings from all sensors are recorded in the same session. This is useful to inspect the case when the feature vector of a dataset example comprises components from different sensors together.

4.2. Pre-processing

Raw sensors data usually need to be processed before feeding into a learning system. One of the most important steps in the context of the current attack is the detection, and extraction of the signal segments that correspond to key strokes from the continuous stream of sensors reading. This step could form a separate research task, and several approaches might be followed. Some of the previous works on this attack regard this task as a straightforward anomaly detection problem [9], or a simple classification problem [12], while others treat it as a significant part of their whole system[10]. Yet, some authors leave this step as a separate undertaking that lies on the shoulder of the attacker, and assumes knowledge of the keystrokes delimitations [13]. We follow the same suit, as the course of extracting this knowledge is independent of the eventual performance, assuming consistency among all datasets. We isolate the keystrokes sensors data by matching their timestamps to the start and end time of each button click event, which we also collect during the experiment, using Android-provided motion events, through onTouch method. The latter method is implemented as per the onTouchListener interface requirement. It is these events that supply the experiment with the labels necessary for the training examples.

Other pre-processing techniques are also possible and usually crucial for successful learning, including normalization and calibration. Normalization is needed when different features of input data belong to different scales, of several order of magnitude discrepancy. Rescaling might be necessary to ensure that no single feature has influence that may not reflect their real relative importance in deciding the outcome. In our case, all sensors reading in all three axes are of the same or only one order of magnitude variance. Calibration is usually also needed, for example, to remove the projection of Earth gravity from accelerometer data and initial orientation from gyroscope

data. Using the linear accelerometer in our experiment ensures the already gravity-less acceleration readings, while initial bias in other sensors are not of much concern since the whole experiment is conducted under the exact same conditions, including any initial biases.

In our data sets we have encountered no missing values, as we have collected all the data ourselves programmatically. Very few outliers could be seen, probably because of abrupt unintended motion of the user hands. We simply got rid of those outliers, as they play no representative role in the input data. We also did not need any dimensionality reduction techniques, as our data are already of relatively low dimension (18, in most cases, except when applying sensors fusion, where every sensor contributes 18 features).

The other significant pre-processing we applied, besides keystrokes extraction, is the aggregation of sensors data by each key, as explained in the next section, to create the features. Finally, some works in the literature had to normalize the sampling rate of sensors, termed de-jittering in [11], to compensate for non-uniform sensors sample intervals. However, this is basically needed for the purpose of standard signal analysis methods, whereas our features are mainly simple statistics that use aggregation of few samples every another key, as discussed later, and no de-jittering was applied.

4.3. Feature selection

For the purpose of classification, the input dataset is a set of examples. Each example is a features vector (collection of features or attributes), which is fed to the classifier and mapped collectively to a pre-set output label (a digit key, in our case). Previous works on the keystrokes inference attack vary greatly in the set of features employed for classification, and there is no obvious evidence of which set is better, indicating an open research area in this direction. For our experiment, we have chosen to apply simple statistics on sensors data from the time domain only, though some authors include also frequency domain signal features.

The output of our application is a set of files, each recording a continuous stream of readings from one sensor, or the combination of sensors in case of data fusion. In addition, one file always contains the touch events information, namely the start time, end time, and the particular tapped key. One touch event normally spans several sensor samples. In our experiment, a key tap takes on average 80 ms, and the sampling rate of the sensors is, at most, 100 samples per second in theory. This means around 10 samples per key in the best case. In practice, however, we found that each key corresponds to an average of 5 sensor samples. Individual samples are meaningless relative to a key tap, and so we aggregate the samples for each key, producing simple standard statistics of min, max, mean, median, standard deviation and skewness. In this manner, for a dataset of 300 keys, we obtain 300 examples, each of which comprises 18 features, plus the class label (the key symbol itself).

The aggregation could be accomplished using any programming language or computational package (e.g. MATLAB or Octave), though we have written a simple script in the R language to match sensors and key data, and perform the statistical calculations.

4.4. Classification

The goal of the inference attack is, given a bunch of sensors samples, to map every pattern of readings into an output class, and the percent of correct mapping forms the accuracy of the classifier. We adopted the implementation of classification algorithms in Weka suit of machine learning software [14]. Many classifiers are available in Weka, and choosing a particular method is not critical for our experiment. However, our initial exploration revealed that the ensemble learning can give better results, with implementations in "*meta"* Weka package. In particular, "Bagging" classifier showed the best performance on average (with FT base model), though we do not aim to venture any claims related to classification algorithms performances. "Bagging" is a general technique for improving the accuracy of a given learning algorithm. As an ensemble learning method, it aggregates multiple learned models of the same type (e.g. decision trees), and uses voting to combine the output of individual models [15].

5. Evaluation and discussion

All experiment runs were conducted under 5-folds cross-validation testing option, and almost the same dataset size of 300 examples. Fig. 2 shows the results of classifying the datasets using Bagging ensemble learning, with Functional Trees base model [16]. Fig. 2(a) is an overall comparison between all settings in terms of classification accuracy. The figures 2(b)-2(g) are the confusion matrices of the individual settings, which give more detailed insight into the performance of the classifiers relative to individual keys.

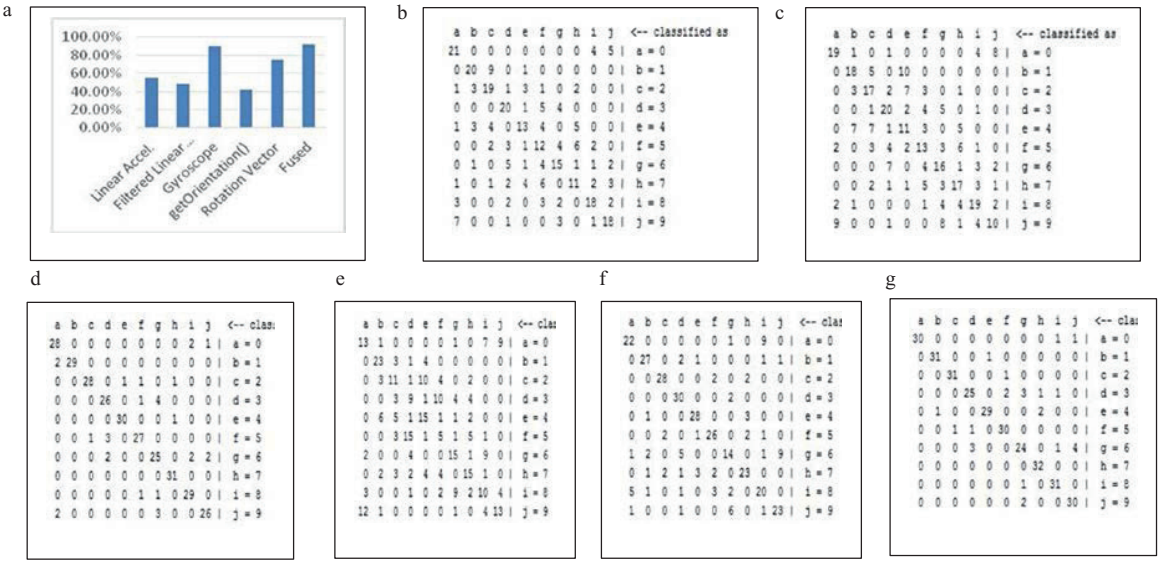

Fig. 2. Experimental result (a) accuracy comparison among all sensors (b)-(g) confusion matrices for linear accelerometer, filtered linear accelerometer, gyroscope, getOriented method output, rotation vector and the fused sensors, respectively.

Several notes are in order, based on the obtained results. It is obvious that, confirming to [11, 12], the gyroscope sensor alone has a superior performance compared to all other sensors, even when combined together. This result is actually consistent with the observation that rotations have more power to distinguish between different keys than shifts. This is also the reason for the inferior accuracy of both accelerometer variations (with and without high pass filtering), and even the rotation vector which relies on the accelerometer as one of the components in deriving its value. Another factor that affects the rotation vector performance is the magnetic field sensor, which is also one of the components in calculating the vector values. Magnetic field sensors are known for their inaccurate and noisy outputs in current commodity smartphones, especially in the presence of nearby metals. What supports this conclusion is that the accuracy of the rotation vector sensor is still much better than the synthetic sensor that is based on the getOrientation method, though both depend on the accelerometer and magnetometer. The difference in accuracy is apparently due to the inclusion of gyroscope into the derivation of the rotation vector.

It also seems that sensors fusion is not always the best option, and depends largely on the proper selection of the ingredient sensors. Filtering also depends on the context, and in our case, it appears that the already filtered linear accelerometer has less performance with more filtering; the high pass filter, despite more appropriate to extract occasional motion like keystrokes, could leave the classifier with less than enough information to discern different keys.

6. Conclusions

The introduction of sensors into smartphones is definitely a much welcomed addition that holds a lot of potential. The dark side of integrated sensors, however, is the new attack vectors that could rely solely on sensors as side channels into user privacy and security. Inferring user taps on touch screens is a recent threat that has been investigated by researchers, utilizing mainly the built-in accelerometers and gyroscopes. This paper addressed the apparently interesting question of the best performing sensor, among the commodity available sensors on Android supported platforms today. Four sensors were compared in an experiment that implements the attack, namely, the linear accelerometer, gyroscope, rotation vector sensor, and the combined accelerometer and magnetometer synthetic sensor. The results showed a greater benefit of exploiting the gyroscope sensor, or a fusion of several sensors (perhaps excluding the sensors with magnetometer component) to conduct such attacks, from the perspective of an attacker; that is.

Acknowledgements

This research is supported by the Ministry of Higher Education under research grant LRGS/TD/2011/UKM/ICT/02/02.

References

[1] International, j. *Interactive Overlay*. Available: http://www.jawsware.mobi/code_OverlayView/. 2013.
[2] Aviv, A. J., Gibson, K., Mossop, E., Blaze, M.,Smith, J. M. "Smudge attacks on smartphone touch screens," in Proceedings of the 4th USENIX conference on Offensive technologies, 2010. p. 1-7.
[3] Schlegel, R., Zhang, K., Zhou, X., Intwala, M., Kapadia, A.,Wang, X. "Soundcomber: A stealthy and context-aware sound trojan for smartphones," in Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS), 2011. p. 17-33.
[4] Maggi, F., Volpatto, A., Gasparini, S., Boracchi, G.,Zanero, S. "A fast eavesdropping attack against touchscreens," in Information Assurance and Security (IAS), 2011 7th International Conference on, 2011. p. 320-325.
[5] Lane, N. D., Miluzzo, E., Lu, H., Peebles, D., Choudhury, T.,Campbell, A. T. "A survey of mobile phone sensing," Communications Magazine, IEEE, vol. 48, 2011. p. 140-150.
[6] Cai, L., Machiraju, S.,Chen, H. "Defending against sensor-sniffing attacks on mobile phones," in Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds, 2009. p. 31-36.
[7] *Motion Sensors*. Available: http://developer.android.com/guide/topics/sensors/sensors_motion.html
[8] Cai, L.,Chen, H. "TouchLogger: inferring keystrokes on touch screen from smartphone motion," in Proceedings of the 6th USENIX conference on Hot topics in security, 2011. p. 9-9.
[9] Owusu, E., Han, J., Das, S., Perrig, A.,Zhang, J. "ACCessory: password inference using accelerometers on smartphones," in Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications, 2012. p. 9.
[10] Xu, Z., Bai, K.,Zhu, S. "Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors," in Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks, 2012. p. 113-124.
[11] Cai, L.,Chen, H. "On the practicality of motion based keystroke inference attack," in *Trust and Trustworthy Computing*, ed: Springer, 2012. p. 273-290.
[12] Miluzzo, E., Varshavsky, A., Balakrishnan, S.,Choudhury, R. R. "Tapprints: your finger taps have fingerprints," in Proceedings of the 10th international conference on Mobile systems, applications, and services, 2012. p. 323-336.
[13] Aviv, A. J., Sapp, B., Blaze, M.,Smith, J. M. "Practicality of accelerometer side channels on smartphones," in Proceedings of the 28th Annual Computer Security Applications Conference, 2012. p. 41-50.
[14] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P.,Witten, I. H. "The WEKA data mining software: an update," ACM SIGKDD Explorations Newsletter, vol. 11, 2009. p. 10-18.
[15] Breiman, L. "Bagging predictors," Machine learning, vol. 24, 1996. p. 123-140.
[16] Gama, J. "Functional trees," Machine Learning, vol. 55, 2004. p. 219-250.