Reconstructing non-intrusively collected keystroke data using cellphone sensors

Luis Héctor Chávez LHCHAVEZ@STANFORD.EDU

Abstract

Text input through keyboard can be intercepted by an attacker without having to compromise the system or the connection with the keyboard by relying only on side-channel information, such as acoustic emanations from keystrokes or vibrations induced on the surface it is resting upon. This project combines ideas from several previous works to make a robust interception, using only sensors available on a modern portable computing device. In particular, Mel-frequency cepstral coefficients are used to extract features from audio data, Fourier Transform is used to extract features from accelerometer data, and Support Vector Machines with a Radial Basis Function are used to classify the events when recorded.

1. Introduction

Computer systems can be eavesdropped upon without the need to compromise a specific part of the system. Side-channel attacks are possible when an attacker has an additional channel of information about the system (Kelsey et al., 2000). Magnetic fields, RF emissions, power consumption, timing and interference on other data channels can all be used for side-channel attacks (Ferguson and Schneier, 2003). The term is usually applied to cryptographic systems, but it can also be used to describe attacks on other computer systems. I will explore a side-channel attack on a computer by detecting the keystrokes a user performed that only requires physical proximity with the victim and an everyday portable computing device, such as a mobile phone or a tablet. Previous attacks on non-intrusive keystroke data collection have been made using acoustic emanations (Zhuang et al., 2005) or vibrations picked up by a cellphone accelerometer (Marquardt et al., 2011).

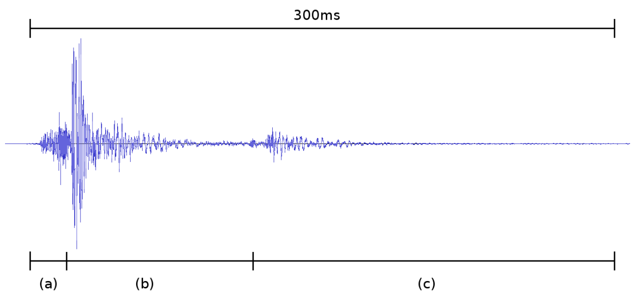


Figure 1. Sound profile of a keystroke in a mechanical keyboard: (a) finger touches the key cap and slides down the switch. Membrane keyboards emit almost no noise in this part (b) the cap exceeds the necessary force to activate the switch, which clicks against the backplate (c) the switch is released and a spring pushes the switch and the cap back to its regular position.

1.1. Acoustic emanations

Most common keyboard technologies rely on mechanically depressing a button to close a circuit that goes back to its original position when released. To do that, flexible membranes or mechanical switches are often used (Miller, 2011). Membrane keyboards are significantly quieter than mechanical switches, emitting only 47.4dB of noise. Mechanical switches on the other hand can emit up to 59.3dB (for reference, a quiet room has ambient noise of about 35dB). The sound profile of a typical keystroke can be seen in figure 1. These acoustic emanations can be easily picked up by a microphone.

1.2. Vibrations

While most of the energy used to press a key gets used either to make the keycap travel and close the circuit or dissipates as noise, some of that noise and energy is propagated to the surface the keyboard rests upon. Vibrations are produced even when typing on keyboards that have no moving parts and produce significantly less noise than membrane or switch keyboards, like virtual keyboards on tablets and phones, or the Microsoft Surface's capacitive keyboard. Vibrations can

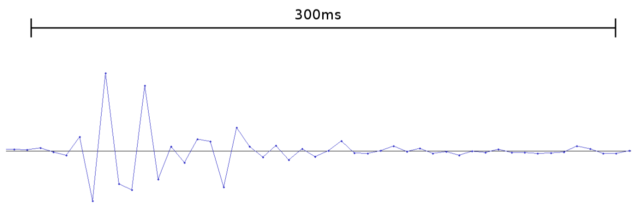


Figure 2. Although similar to the acoustic profile, the vibration profile only clearly shows the vibration produced by the switch hitting the backplate.

be picked up by a nearby accelerometer placed in the same surface as the keyboard. Unfortunately, the rate at which accelerometer data can be sampled on common devices is significantly lower than audio, the data is noisy, and the rate is not constant, so it alone cannot be used to accurately detect keystrokes. Previous approaches have complemented the collection of vibrations with dictionaries to improve detection rates (Marquardt et al., 2011). The vibration profile of a typical keystroke can be seen in figure 2 .

2. Data Set

The data set used in training and testing the system was a collection of 10,000 keystroke events distributed across all keys of a keyboard with Cherry MX Blue mechanical switches placed on a 1.5″ thick wooden table. Audio and accelerometer data were collected using a very simple app made for Android, using a Nexus 7 device placed 20 cm away from the keyboard with mono audio at 22.05 kHz and the fastest accelerometer collection rate supported of 200 Hz.

2.1. Data cleanup

The device already does applies noise reduction to the stream before providing it to the app, so no further action was necessary. Accelerometer data is returned as a timestamp (in nanoseconds since the Unix epoch), and three floating-point readings for the x, y, z accelerometers, measured in meters per second. Due to the orientation of the device, only the z accelerometer is used after normalizing it to have a bias of zero instead of g. Since the accelerometer events are polled using a software timer, the timing of the readings is not consistent, so interpolation with a 7th degree polynomial considering 4 points before and 3 points after each pair of readings was used to produce a consistent 200 Hz signal and encode it in the same format as the audio (16-bit PCM). Normalization was then used to avoid clipping and to boost the signal.

2.2. Keystroke segmentation

When both the audio and accelerometer data are collected in order to be processed, a small program identifies all the potential keystrokes in the audio file and in the accelerometer data file by splitting both the audio and accelerometer data in 100 ms windows and comparing the root mean squared (RMS) of both windows with a threshold of 10% of the maximum amplitude. This helps reduce false positives since ambient noise has little effect on the vibrations picked, and most vibrations done on the table (like hand motions or mouse movements) do not produce as much audible noise.

2.3. Training data set labelling

Labels were produced with the help of an SDL program running in the desktop computer that logged keystroke events (both key up and key down) together with timing information. Since the audio, accelerometer data, and labels come from machines and subsystems with different clocks, they are not synchronized. After the potential keystrokes were identified, another program correlated the events with the key log by using linear regression to obtain an affine transformation mapping the key log timing data to the potential events that minimized synchronization error. Once the events were synchronized, random 100 ms windows were chosen from the recording and added to the events (with roughly the same cardinality of the most frequent event: the space bar), and labelled with no key. This was then used to train the classifier.

3. Feature Extraction

Mel-frequency cepstral coefficients were chosen to extract audio features since they are widely used in speech recognition and in previous attempts to aurally recognize keystrokes (Zhuang et al., 2005). A Hamming windowing function was used in 2 ms steps with 25% overlap, then obtained the log of the FFT, and then calculated the iFFT to obtain the cepstral coefficients. That was done using Scikit-talkbox (David, 2008) using the default parameters, and then selecting the first 16 coefficients for the first 10 windows.

Since the accelerometer data showed peaks in the 50-80 Hz range, and was is sampled at 200 Hz, 160 samples (0.8 s) were needed to correctly extract the features. The accelerometer data was then passed through a Hamming window function and then the FFT was obtained, and only the buckets between 50 and 80 Hz were used as features. This made the feature vector to be 190-dimensional.

4. Classification

The data was first normalized by setting the mean of the data equal to zero and the variance equal to one in each of the dimensions of the features. Several configurations of SVM were evaluated, using the implementation provided by Scikit-learn (Pedregosa et al., 2011). Linear, polynomial, and RBF kernels were evaluated, as well as two multi-class classification strategies: one-against-one approach outlined by (Knerr et al., 1990) and a one-vs-the-rest strategy (Pedregosa et al., 2011). Several values of C were also evaluated.

5. Evaluation

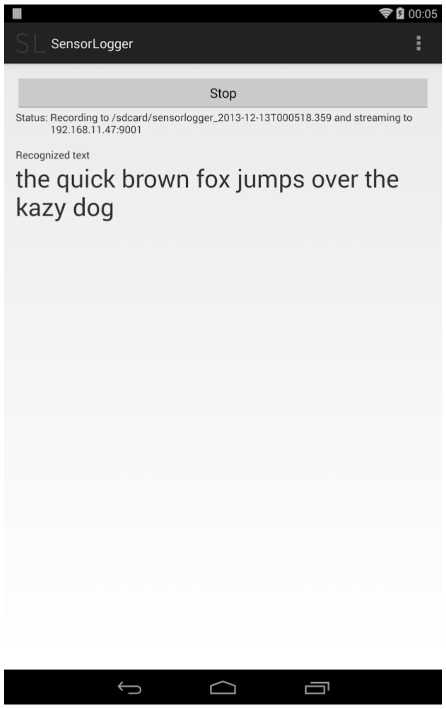
The best results were achieved using a SVM with a RBF kernel with C=1.5. When measured with 10-fold cross validation, this achieved 0.000158% training error and 4.93% validation error, which is roughly the same as the results in previous papers. Figure 3 has a graph with more details.

6. Real-time classification

The devices that were tested didn't have enough computation power to keep up with the sensor reading, segment the data, extract features and classify the keystrokes in real time. To compensate for that, the extraction program was modified to feed the uncompressed raw data to a more powerful machine through a TCP socket using a very simple framing scheme to be able to easily demultiplex the data on the server. The server uses the same code to perform training and validation, and if a key is classified, a packet with that information is sent back to the device for display Figure 4 shows a screenshot of the application running on the testing device. The volume of the data limits the applicability of this technique to local area networks.

7. Future research and conclusion

Having more spatial information through the use of stereo microphones can yield to increased accuracy when classifying keystroke events. Despite the fact that most devices have two microphones, the second one is typically used for noise reduction, and is not available for the developers to use. Other devices, such as the Microsoft Surface, do allow the recording of stereo audio, but do not allow raw audio to be captured. The audio compression used takes away most of the gains of the extra audio channel. Devices with multiple high-quality microphones are available in the market today, such as the Nokia Lumia 1520, but they are not as widespread. As the sensors of the portable computing devices improve, more and better eavesdropping attacks similar to this can be used in public environments to collect information non-intrusively and without raising suspicion.



Status: Recording to /sdcard/sensorlogger_2013-12-13T000518.359 and streaming to 192.168.11.47:9001

the quick brown fox jumps over the kazy dog

Figure 4. A screenshot of the application running and receiving recognized keyboard events from the server, with just one recognition error

References

Cournapeau David. Scikit-talkbox: a set of Python modules for speech/signal processing. 2008. 3

Niels Ferguson and Bruce Schneier. *Practical Cryptography*. Wiley, 2003. 1

John Kelsey, Bruce Schneier, David Wagner, and Chris Hall. Side channel cryptanalysis of product ciphers. *Journal of Computer Security*, pages 8(2–3):141–158, 2000. 1

S. Knerr, L. Personnaz, and G. Dreyfus. Single-layer learning revisited: A stepwise procedure for building and training neural network. *Neurocomputing: Algorithms, Architectures and Applications*, 1990. 4

P. Marquardt, A. Verma, H. Carter, and P. Traynor. (sp)iPhone: Decoding vibrations from nearby keyboards using mobile phone accelerometers. In

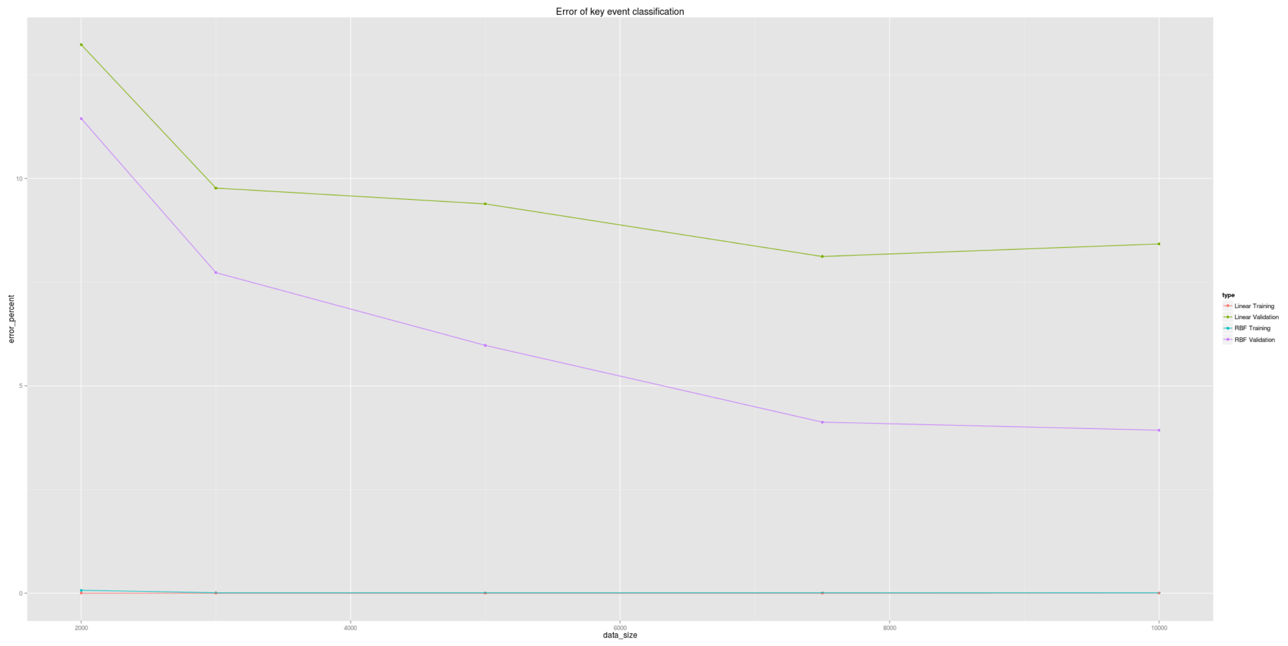


Figure 3. Training error with respect to data size

Proceedings of the ACM Conference on Computer and Communications Security 2011, pages 373–382, Chicago, IL, 2011. 1, 1.2

Patrick Miller. Mechanical keyboard FAQ: Pick the right switch, 2011. URL `http://www.pcworld.com/article/242037/mechanical_keyboard_faq_pick_the_right_switch.html`. 1.1

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011. 4

Li Zhuang, Feng Zhou, and J. D. Tygar. Keyboard acoustic emanations revisited. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, pages 373–382, Alexandria, VA, 2005. 1, 3