Side Channel Attacks for Keyboard Snooping

I’ve been working on my latest project with the Darwin Deason Institute for Cybersecurity since April of last year. The Deason Institute does research on sensitive projects for various clients, and as a result there are limits on how much information we can disclose about our projects. This project is a bit more public because we’re submitting a paper about it to a journal, so I can talk about most of what I worked on.

Essentially, the project examines the feasibility of using mobile phone sensor to determine what someone is typing on a nearby keyboard. We designed a setup that involved a number of phones sitting on a conference table, as would be common in a meeting. We had one person sit at a keyboard, interviewing another person and recording their responses. By using a keylogger we determined which keys were typed, and we collected sensor data from the phones with the goal of using this sensor data to predict the keystrokes.

Side Channels

We focused on using mobile phones because of their ubiquity. Smartphones are everywhere, and the sensors on these phones have been improving rapidly. A side channel attack takes advantage of these sensors, repurposing a benign sensor for a malicious purpose. In our research we looked at the microphone, the accelerometer, and the gyroscope. These sensors are active all the time; apps like FitBit use the accelerometer to count steps, apps like Google Cardboard use the gyroscope to navigate a 3D environment, and camera apps use the microphone to record sound.

The use cases for these sensors are so varied that some of the sensors (like the accelerometer and gyroscope) are available to all programs – the user does not have to grant permission for an application to use these sensors. This means an attacker could access the accelerometer data and the user would never find out.

System Design

Our overall system design is as follows: collect sensor data from a certain number of phones, combine and parse this data, and use the data to predict which keystrokes were typed. This is a classic signals processing/machine learning problem. Essentially, we’re performing speech recognition, except instead of listening for people talking we listen for people typing, and instead of predicting spoken words we predict type words.

There are a number of steps involved in processing a particular sensor stream, such as microphone data. Step 1 is collecting microphone data from the phones. Step 2 is combining this microphone data together. This step is harder than it seems at first glance; sometimes different phones keep time differently because their clocks are slightly incorrect. Once we correct for the clocks, however, we have combined microphone data from multiple phones. With this data, we proceed through three levels: first, a convolutional neural network; second, a recurrent neural network; and third, a language model. After these three levels the microphone data has been turned into predicted keystrokes.

Convolutional Neural Network

Convolutional Neural Networks (CNNs) became popular because of their performance in image processing. The basic idea is that you can analyze an image by breaking the image into smaller sections. Then, you “convolve” over the image, meaning that you slide your window over and down until you’ve scanned the entire image.

Cnn.gif

CNNs blew the competition out of the water in the image processing world in 2012. When this happened many data scientists became interested in deep learning because it was such an improvement over other methods for object recognition. In time CNNs were applied to signals processing as well. In the same way that CNNs use a sliding window over an image, they take a small segment of audio data at a time and move that window over across time until the entire audio signal has been processed.

IMAGE

In our case, we break the input signal into small chunks and run a CNN over each chunk. The idea is that each chunk either contains one keystroke or no keystrokes. The CNN will output a guess as to which keystroke the chunk contains, or if it even contains a keystroke at all. If we have 10 chunks that represent someone typing the word “student”, the CNN might output [s, no\_key, t, u, d, no\_key, e, n, t, no\_key].

Recurrent Neural Network

CNNs are great at taking a large input like an audio signal and condensing it down into a small representation. The issue with CNNs is that they don’t often recognize the relationship between characters. For example, in English “q” is almost always followed by “u”. It’s reasonable for the model to guess “u” as the next letter every time it sees a “q”.

To take advantage of these dependencies, we use a Recurrent Neural Network (RNN). RNNs became popular for text analysis and are used today for everything from translation to sentiment analysis. The benefit of RNNs is their natural understanding of time-dependence; not only can an RNN learn that “u” comes after “q”, it can also learn that “Giallanza” comes after “Tyler.”

Language Model

If we had enough training data, the RNN could perform all the tasks of a language model (like learning the correct spelling of different words). However, with our limited data the RNN can still make mistakes. For example, the RNN may output “for exsmple” instead of “for example”. To fix these mistakes we also built out a simple language model.

The language model looks at a dictionary of English language words and maps the output from the RNN to the nearest word. This helps fix the discrepancies between what the RNN predicts and what people actually typed, helping with “spelling errors.”

Results

At this stage in the project I cannot fully discuss the results from the system. However, I’ll run through the different ways we evaluate how well the model is performing and discuss at a high level how well the model is doing.

First, we can evaluate the character-level accuracy of the model. At the character level, the model gets a penalty if it gets a single letter wrong. Our model achieves pretty good character-level accuracy, especially after the RNN refines the results from the CNN.

Character-level accuracy has a number of issues. For example, if a model is really accurate because it guesses every vowel correctly but does not guess any consonants correctly, it will not produce useful results. Humans are generally able to read a sentence with all the vowels removed (try reading this sentence -> try rdng ths sntc) but cannot read the same sentence with all the consonants removed (try reading this sentence -> eai I eee).

To overcome these issues, we also scored our model based on word-level accuracy. Word-level accuracy only punishes a model if it gets an entire word wrong. This mirrors the way humans read a bit better. If one model gets 50% of the words correct and another model only gets 10% of the words correct, the first model will produce much more readable text. Without giving away too many technical details, our model can correctly score approximately 33% word-level accuracy.

Why it Matters

As phone sensors become better side channel attacks will only increase in potency. Even more worrying than mobile phones is the growing prevalence of the Internet of Things (IoT). New devices like home security cameras, Amazon Echos, and even smart appliances are all loaded with sensors, including the sensors we exploited in this study.

We have demonstrated that these sensors can be repurposed maliciously in ways that are quite unexpected; it is possible for your Amazon Echo to record not only your spoken words but also the words you type into your computer. A number of other studies in the field have shown similar results for other forms of personal data, including using smart watches to determine your phone password. We expect that the incidence and potency of these attacks will only increase in the future.

In Summary

I helped develop a system that can use mobile phone sensors to determine what someone typed on a nearby keyboard.

The system can correctly identify about one third of the words a person types.

I have written a paper that explains in technical detail how our system works. The paper will be submitted to the ACM Journal of Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT) by their November 15 deadline.