SNARL:
Simulating Known Actors with Machine Learning

Bryan Cai, Yasyf Mohamedali
{bcai, yasyf}@mit.edu

I. Abstract

We present SNARL, a suite of tools that is able to simulate a user's typing pattern with the goal of exposing the vulnerabilities of new biometric based identification techniques. Researchers have known for many years that password based verification systems are insecure, in part due to security holes in applications, and in part due to users (e.g. password reuse). In an effort to bolster security, some applications leverage additional data; we are particularly interested in the merits and drawbacks of biometric data. Biometric identification schemes are in some aspects very secure, because a person can often be uniquely identified by some combination of biometric factors. On the other hand, if somehow this information were to be stolen, akin to a password being leaked, it is nearly impossible to change, since they are often either physical traits that exist from birth or habits developed over many years. Our paper demonstrates a system that is able to rapidly learn and simulate user password typing habits, and suggests that keypress timings should only be used as an inexpensive technique that does not significantly boost security.

II. Introduction

The past few years have seen several attempts to increase security in password-based authentication systems by adding secondary factors derived from biometric patterns in user behavior. Examples of this include Yang and Haddad's `punchTimeAuth` [4] and the commercial product being developed by UnifyID [1]. However, there has not been much focus on the barrier this presents to attackers who are capable of mimicking their targets, particularly in the age of commodity cloud hardware and effective deep learning methods.

We present an exploration into the pitfalls of relying on input-delay-based biometric data as a second factor for authentication, using `punchTimeAuth` as our standard for comparison. We show that the model introduces a prohibitively high number of false negatives when exposed to real-world usage. Furthermore, we present several adversarial models which can successfully impersonate a user typing their password, trained on background typing data that need not include the password itself. In order to collect this data, we built two collection mediums: a Google Documents look-alike and an extension for Google Chrome. We collect which keys are pressed, the length of time each is held for, and the delay between each pair of keys. Finally, we present a novel neural model for attempting to distinguish users based on the information we collect, along with some conclusions and observations about this source of data as a second factor for security.

The code for the systems described in this paper can be found at https://github.com/yasyf/snarl.

III. Threat Model

Our exploration focuses specifically on the systems which analyze the delays between a user's keypresses while she is typing her password. We assume that the system in question has been previously trained with multiple samples of the user's typing patterns. Our tests were against the seventh quantile-based `punchTimeAuth`. We assume the role of an adversary which has managed to obtain the password of a user (such as through a database breach, phishing scam, or cross-site scripting exploit), and is attempting to gain access to a system employing the second factor of security described above. We further assume that the adversary has time to build tools as we did to collect typing data on their target. Importantly, we do not require the adversary to ever witness their target typing the password in question.

IV. Approach

A. Data Collection

In order to collect sufficient typing data from the users that agreed to participate in our exploration, we built two interactive tools. In both cases, we solicited complacent users by inviting them to use our applications, relying on their lack of careful inspection when receiving a link from someone they trust.

The first is a Google Docs look-alike [2], which imitates a well-known service in a phishing-style attempt to convince a complacent user to type on the page. We overlaid a text box on a webpage containing the CSS from a Google Doc we created, allowing the user to type as if this were the application they are so familiar with.

We implemented a small client-side library in JavaScript which listens for the `KeyDown` and `KeyUp` DOM events emitted from the textbox, so as to record the length of time each key is held for and the delay between key presses, as well as the actual characters being typed. This data is stored in a

local buffer and is periodically uploaded to our servers, where a Flask app records the logged objects to a MongoDB instance, tagged with a unique identifier for the user in question.

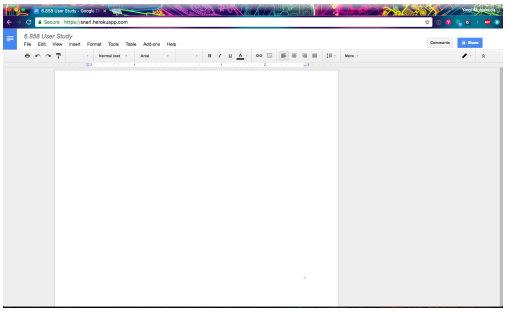

Figure 1: Google Docs look-alike

The second technique we developed for data collection is a key-logging Google Chrome extension [3]. While the data quality from a document editor was vastly superior to the flow of data from casual bursts of typing in a browser, our models demanded as much data as we could possibly collect. Thus we augmented our corpus by creating an extension that injects a script into every web page, using the same library as before. We asked participants to install the extension and made them aware of the implications; our proof-of-concept was installed with consent but could just as easily be installed through less transparent means by a bad actor.

The combined results of these two methods resulted in a data set of approximately 25000 character pairs across several users.

B. Models

1) Means Model: For this project we trained several types of models. We first focused on the problem of tricking a security system into believing that we, as a hypothetical bad actor, were the original user, by simulating the delays that the original user would have in their password typing.

To begin, we trained a simple means model to estimate the delay a specific user exhibits between any two characters, regardless of context. We simply used the average delay between a given pair of characters for the user in question, filling in missing data with a global mean. This model performed surprisingly well.

2) Neural Models: Our first neural model has the goal of mimicking how a user types, building on the simple means model and capturing additional context. For each use when given a sequence of letters, the model outputs a sequence of delays between each consecutive pair of letters that mimics how the user would have typed those letters. It does this one pair at a time; the model takes as input W letters preceding and W letters following the pair we are predicting. For each of these N letters, the model constructs a vector of length V, which represents the forward and backwards context in the word. Using these two vectors, one delay is produced. Iterating over every consecutive pair of letters gives the desired sequence of delays.

The next model, the distinguisher, tries to achieve the opposite effect; given a sequence of letters and timings, it tries to guess whether the sequence was typed by the user. This model takes as input a sequence of vectors, where each vector encodes what letter was typed, how long the key was held down for, and the delay from the last keypress.

V. Implementation

We built a Python backend in Flask with an API to support our collection and processing. This included routines to receive JavaScript objects with `KeyCodes` and translate them to one-hot vectors which could be fed into our neural models. In order to aid with supervision of the collection process, we built several tools and dashboards to explore the data being collected.

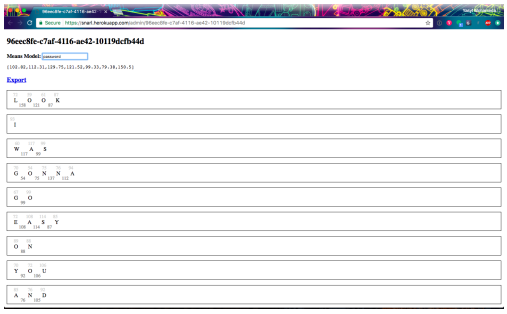

Figure 2: Data Debugging Dashboard

A. Models

1) Means Model: Our means model for a given user u was the result of a few simple NumPy operations, with the final `means` matrix defined as follows. `means` is an $N \times N$ matrix, where $N = |\mathcal{C}|$ is the cardinality of the set of characters that can appear in a string. For simplicity, we limited this set to lowercase letters, numbers, and punctuation. Element (i, j) of `means` indicates the predicted user delay between typing character i and character j.

Let y_u be the flattened length-n vector of measurements $(c_1, c_2, \texttt{delay})$ taken for user u, where `delay` is the delay between this particular instance of typing c_1 and c_2. Let $N_{c_1,c_2} = \texttt{count}(y_u : c_1 = i, c_2 = j)$.

$$\texttt{means}[i][j] := \begin{cases} \frac{\sum_{y_u : c_1 = i, c_2 = j} \texttt{delay}}{N_{c_1,c_2}} & N_{c_1,c_2} > 0 \\ \frac{\sum_{\texttt{means}:N_{i,j}>0} \texttt{means}[i][j]}{\texttt{count}(\texttt{means}:N_{i,j}>0)} & N_{c_1,c_2} = 0 \end{cases}$$

2) Neural Models: The neural models were implemented in Python, using Keras with a TensorFlow backend. Characters were represented using one-hot vectors, and delays and hold times were measured in milliseconds. We first discuss the mimic model. This model takes two $W \times A$ matrices, where W is the size of the context we consider and A is the alphabet size. The first matrix represents the forward context, and the second matrix represents the backwards context.

For example, if the word is "SECURITY," if we had a window size of 4 and we are trying to predict the delay between the "R" and the "I", the first matrix is the sequence of one-hot vectors representing ["C", "U", "R", "I"] and the second is the sequence of one-hot vectors representing ["R", "I", "T", "Y"]. If there are not enough letters, we pad the matrix with zero vectors. Each of these $W \times A$ matrices is fed through an RNN that produces the intermediate vector of length V. We then concatenate these two vectors and feed it through a fully connected layer with 50 nodes, and then then to a final output node.

We used the mean absolute error as the loss function, and standard gradient descent as the optimizer.

The distinguisher model takes as input a matrix of size $L \times (A + 2)$, where L is the maximum word length we allow, which is set to 15 but can be set higher in the future if necessary. Each row of the matrix consists of the one-hot vector of length A, with the keydown time and delay time appended. This is fed through one RNN layer, then through several dense layers, culminating in an final layer with 2 outputs and softmax activation.

VI. Results

A. Means Model

As we will discuss later, the laborious and manual process of training systems like `punchTimeAuth` left us in the difficult position of not having many options for automated testing. Thus, our testing efforts were the result of us randomly sampling words and attempting to have our model type them in the fashion of the training user. The same user would train the security system by using that word as their password for a new account, and we would inject JavaScript to simulate our predicted delays at a login screen. Using this method, our means model was able to successfully imitate several users 90% of the time. This result is somewhat surprising, given that we did not expect users to necessarily type in a document editor the same way they type a familiar password.

B. Neural Models

Our neural models for mimicking user input achieved a lower success rate on the task of imitating users to `punchTimeAuth`, succeeding on about 70% of passwords. However, there are several possible explanations. We believe that the most likely reason is because the `punchTimeAuth` is based on deviation from the median for each keypress, which the means model learns more directly. Secondly, our neural model may be overfitting to the data, and we believe results would improve if there was more data, collected over a longer span of time, to train on.

The distinguisher model had a validation accuracy of 80% to 85%, which indicates that there is an appreciable difference between how distinct users type; however, it is clearly also not enough to uniquely identify a user. Again, this analysis suffers from low sample size, as we were only able to analyze four user's typing habits.

VII. Discussion

We faced several challenges in collecting data and implementing the models for this exploration. One of the largest barriers to rigorous results was the fact that the systems we attempted to break all require significant effort to train. Concretely, in order to evaluate an imitation of a user's typing of a specific password, that user must train the security system on how she types that password. This high barrier to the end user is a shortcoming of the string-specific model used in `punchTimeAuth`, and something we tried to address with our generalized neural distinguisher. However, as we discovered, simple typing timing and language context do not provide sufficient information to produce a generalized model of a user's biometric profile.

Another particularly interesting challenge occurred in the client-side data collection library we authored. Detecting lengths of key presses in addition to delays meant we could not simply listen for `KeyPress` events, and put us at the mercy of the JavaScript event loop scheduler. We spent a nontrivial amount of time post-processing the stream of events we received to piece together the sequence of user key presses.

On the data collection front, we experienced expected challenges including a lack of willing study participants, and foreign scripts interfering with our injected collection routines in the browser. The latter resulted in the data collected from our Chrome extension being very noisy and hard to use.

As part of our analysis of user typing data, we made some interesting observations with regards to the delays between typing keys on a standard QWERTY keyboard. Our initial hypothesis was that keys which are farther apart on the standard keyboard would experience the greatest delay. However, as seen in Figure 3, this is not quite the case.

Instead, we see a pattern where keys that are typed by the same finger (on either side of the keyboard) follow each other in quick succession, with the distance between keys not rendering much information. The exception to this is the peculiarity of typing the same key multiple times. It appears that users consistently slow down when repeating a key.

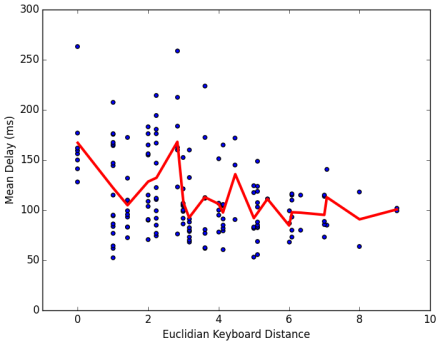

Figure 3: Keyboard Distance vs Mean Delay

VIII. Future Work

We have shown that there is significant differentiation between user's typing patterns, as evidenced by the results of our distinguisher model. However, it is not clear to the authors that there exists sufficient entropy in this data to uniquely identify a user in a large system. Certainly, timing delays and key hold times do not render a model that should be used as a second factor alone. We would like to explore how the signal from such a model can be used a heuristic for other biometric factors, as well as whether or not such signals could be aggregated to provide a secure authentication system.

IX. Conclusion

SNARL has demonstrated that timing-enhanced password authentication has approximately the same level of security as a password-only scheme, given 10,000 to 15,000 characters of data per user, which is about 4 pages of typed words. We successfully defeated the scheme presented in `punchTimeAuth` using both a very simple means model and a neural model. However, our distinguisher models show that there is differentiation in user typing habits, and typing patterns could be aggregated with other data to effectively enhance security.

References

[1] URL: https://unify.id.

[2] URL: https://snarl.herokuapp.com.

[3] URL: https://snarl.herokuapp.com/client.

[4] Joy Yang and Don Derek Haddad. *punchTimeAuth: A Biometric keystroke based authentication*. URL: http://css.csail.mit.edu/6.858/2015/projects/yangjy-ddh.pdf.