User Recognition by Keystroke Latency Pattern Analysis

Dawn Song Peter Venable Adrian Perrig

April, 8th 1997

Abstract

We analyze keystroke latency patterns to identify the person typing on the keyboard. Unlike previous work in this domain, which focused on taking one reference sample and doing user authentication based on the reference sample only, we continuously sample user input and use the data for identification and further learning and refinement of the user model.

1 Introduction

In many situations, a user may leave his computer without logging out or locking the computer. This gives an intruder a chance to use the console and the users login to break into systems and shows the insufficiency of password scheme for authentication purpose. In this project report we present a tool to provide continuous authentication of the user by continuously monitoring the users typing pattern. As soon as a different typing pattern is detected, the computer locks up and the "intruder" is asked to type in a password. This technique can be useful in many settings, for instance for notebooks. It can also be used as an additional biometric authentication method on a high security computer or for military applications.

2 Previous work

Joyce and Gupta have described an user authentication system by using keystroke in [JG90]. They are identifying a user by comparing the keystroke latencies of a fixed string, i.e. the password, with the previously stored samples.

Our work differs substantially with Joyce and Gupta's approach because we do not rely on pre-selected strings such as passwords. Further we are continuously learning and refining our model of the user. In addition we have used a finer grained keystroke event model by processing both keypress and keyrelease events. Joyce and Gupta use latencies between keypresses only.

3 Our Approach

Our approach is to validate user identity at all times by continuously monitoring keystrokes. Each keystroke is captured through by way of the X-windows server and processed either to train the model or to compute a probability that the current user is the same as the user on whom the model was trained.

We are using the keystroke delays to set up a structure similar to a Markov chain which models the mean and variance of the delay between two keystrokes. We take all the combinations of 2 subsequent keys and store the data as a user profile. To identify a user we check which user's model maximizes the likelihood of the recent key-presses.

3.1 Latency Observation

In our approach, we monitor all the key events that the user types. Typing one key triggers a pair of key events: press and release, which we call a key stroke. We get the latency between pressing and releasing a key for each key that is typed, which is called PR-latency. For each two continuous keys typed, we get the latency between the release event of the first key and the press event of the second key, which is called RP-latency. PR-latency is always positive, because a key can only be released after it's pressed. RP-latency can be negative, because the second key can be pressed before the first key is released. Because of the resolution of the X-server, the resolution of the latencies that we can get is 10 ms. The PR-latencies and RP-latencies are two disjoint data sets. The

PR-latencies and RP-latencies are grouped respectively in three different ways: bigram, trigram and word-gram. A **Key event** is a bigram event, a trigram event or a word-gram event.

Bigram: We group every two continuous key strokes into one bigram event and index it by the two keys.

Trigram: We group every three continuous key strokes into one trigram event and index it by the three keys.

Word-gram: We group every continuous set of key strokes that only contains letters, the CapsLock key and the Shift key into one word-gram event and index it by the keys.

The **Cardinality** of a key event is the number of keys in the event. A bigram event has a cardinality of two. A trigram event has a cardinality of three. A word-gram event has a cardinality of the length of the word. The **Bigram model** contains all the bigram events data. A **Trigram model** contains all the trigram events data. A **Word-gram model** contains all the word-gram events data. We call the data set of each index in the three models the **index-set**. The **Cardinality** of an index-set **C[I]** is the cardinality of the index of the index-set.

3.2 Statistics Model

For each user, we build up three models: bigram model, trigram model and word-gram model. In the training phase, we insert the data into the three models as we described above. Then we compute the mean and the standard deviation of each index-set of the three models. In the prediction phase, we use the key-strokes gathered from the X server as the input to the three models and compute the prediction.

Index-set prediction: Assume an index-set has a vector of mean $\vec{\mu}$ and a vector of standard deviation $\vec{\sigma}$. Assume the index-set has a normal distribution. (Later we will show that this assumption is valid.) Given a testing point T of $\vec{x}$, the probability of the point on the given index-set I $P(T \in I) = \frac{1}{C[I]} \sum exp(-\frac{(x_i - \mu_i)^2}{2 * \sigma_i^2})$.

Historical prediction: We are monitoring the key strokes of the user continuously. For each new key event, we compute the the index-set prediction according to the key event on the given model. Then we compute the current prediction of whether the key strokes are typed from the user of the model based on the weighted average of the previous prediction and the prediction of the new key strokes. $P_{current} = \alpha P_{new} + (1 - \alpha) P_{old}$. Because a person's typing can be irregular sometimes, it's better to give prediction based on the typing history. The average weight on the history is exponentially decreased as more key events, so any slow-occurring change of typing pattern will be captured by the prediction.

3.3 System Architecture

A program called xlisten (written in C) grabs keystrokes from the X server and for each key pressed or released outputs a line of data describing the event, including which key was pressed or released and the latency in milliseconds since the last event. These raw data are passed by the shell to the main program (written in Java) which has three distinct input modules to process the three input event types: bigrams, trigrams, and word-grams. Each keystroke can be processed simultaneously by these modules and the results combined or compared. The three input modules have a common output format which is sent to the statistics module. The statistics module has two modes: train and predict. In training mode, it incorporates each new event into its model. In predict mode (and incidentally also during training) it computes the probability of the model given the event. The user interface module, in addition to providing controls, plots a graph of these recent probability results.

3.4 Raw Input

Our system is based on a X-windows capturing program which gives us the delays between key events with a 10 millisecond resolution. This data is then passed to a Java program which performs the statistical analysis.

The xlisten output from the string "I love AI" is shown in figure 1. The meaning of the output fields is as follows:

- p = press event, r = release event
- Symbol of pressed key
- Key mask, where bit 0 = Shift, bit 1 = Caps Lock, bit 2 = Control, bit 3 = Alt
- Key-code
- Delay in milliseconds since last event

It is interesting to note that the order of the characters is interleaved. Unfortunately the X-server processes the keystrokes in batches, yielding 0 as latency between the individual events.

```
Shift_L 0 50 2015
p I 1 31 183
p space 1 65 0
r I 1 31 181
r Shift_L 1 50 0
r space 0 65 0
p l 0 46 183
r l 0 46 4
p o 0 32 184
r o 0 32 176
p v 0 55 0
r v 0 55 0
p e 0 26 183
r e 0 26 0
p space 0 65 178
r space 0 65 180
p Shift_R 0 62 0
p A 1 38 181
r A 1 38 0
r Shift_R 1 62 0
p Shift_L 0 50 182
p I 1 31 0
r I 1 31 181
r Shift_L 1 50 0
```

Figure 1: Raw xlisten output

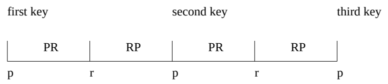


Figure 2: Keystroke Latencies in a Bigram

4 Experimental Results

4.1 Testing of Assumptions

Our mathematical model assumes that the distribution of keystroke latencies for a particular user is normal, so we tested this by plotting latency histograms. As you can see in figure 3, the assumption is essentially correct.

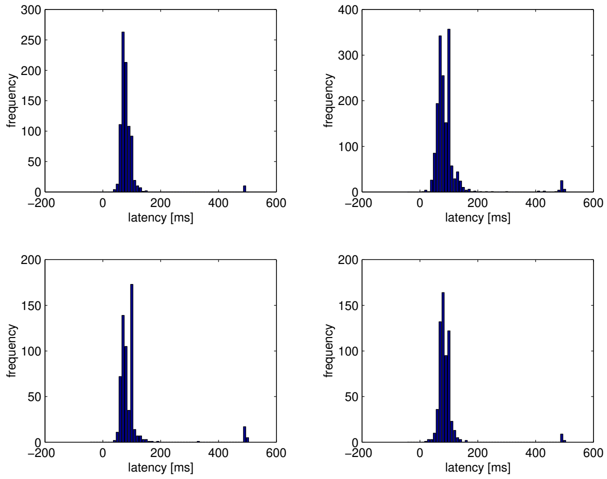


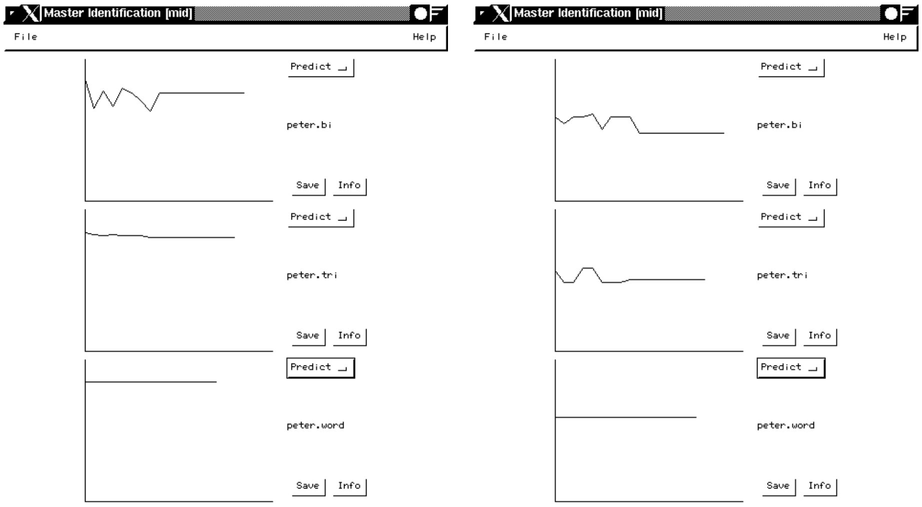
Figure 3: Keystroke Latency Distribution

4.2 An Example

For a simple test of our system, we gathered several hours worth of keystroke data from various users. We used the data from one user to train the model, and then ran predictions both on that user's typing and on the typing of another user. As illustrated by figure 4, all three graphs show significantly higher probabilities when the typist is the same person as trained the model (on the left) than when another person types (on the right).

4.3 Limitations

While the example above shows our program correctly giving higher probability to the correct user over other users, these probabilities are still not far enough apart to be conclusive for many applications. We strongly suspect that the main limitation on the effectiveness of this program in distinguishing users unambiguously lies in the gathering of the raw data from the X server. The X server reports keystroke latencies to a granularity of only



(a) Authentic User (b) Impostor

Figure 4: Predicting Identity Based on a Learned Model

10ms, and when typing is very rapid, it truncates them to zero. A granularity of 1ms would provide more accurate results.

5 Future Work

As mentioned above, a finer granularity in raw input is the first thing required. This could be done fairly quickly with a little hacking inside the X server. It would also be helpful to put some more intelligence on the outside of the running probability generator, to determine whether the results are inconclusive, authentic, or a probably indicate an intrusion. A neural net would probably be a good tool here. An application of this would be to incorporate it into a screen saver so that when the user steps away from his or her terminal without logging out nor locking the screen, if someone else were to come and begin typing, the intrusion would be detected and a password required. The authentic user, on the other hand, could resume work without the inconvenience of a password. Since the majority of security problems come from people not using security measures provided to them because of inconvenience, this added convenience would actually go a long way toward improving security of unattended terminals.

6 Availability

The program is available at `http://gs207.sp.cs.cmu.edu/~adrian/mid.tar`. There are 2 versions of xlisten, xlisten-linux and xlisten-sun. You can run the program as `xlisten-linux | java mid`. Please contact us if you feel that the usage is not intuitive.

References

[JG90] Rick Joyce and Gopal Gupta. Identity authentication based on keystroke latencies. *Communications of the ACM*, 33(2):168 – 176, February 1990.