# User Authentication Based On Behavioral Mouse Dynamics Biometrics

**Chee-Hyung Yoon**
Department of Computer Science
Stanford University
Stanford, CA 94305
*chyoon@cs.stanford.edu*

**Daniel Donghyun Kim**
Department of Computer Science
Stanford University
Stanford, CA 94305
*blueagle@cs.stanford.edu*

## Abstract

In this machine learning application, we try to complement the existing security system by providing another layer of user authentication protection by applying behavioral biometrics on user mouse dynamics. First we collect the user's mouse dynamics data through an application that monitors the mouse movement for the specified duration. We extract out certain signature characteristics in the patterns of a user's mouse dynamics, such as double-clicking speed, movement velocity and acceleration per direction. Using machine learning techniques, we build a Naive Bayes model of a user's mouse dynamics. Then it is possible to detect unauthorized users by finding anomalies in the measured mouse dynamics and to prevent the intruder from further accessing the system. Although there has been considerable amount of literature on fingerprinting or keystroke biometrics, behavioral biometrics based on mouse dynamics has not been very successful so far. Different from the tried methods, we used slightly more detailed features and machine learning techniques such as unsupervised learning for anomaly detection.

## 1 Mouse dynamics

Using the WinNT/XP application we implemented using C# and Win32 assembly, we collect the information on mouse actions generated as a result of user interaction with a graphical user interface by installing a global mouse hook on the user system. We consulted [1] *Detecting Computer Intrusions Using Behavioral Biometrics* by Awad E. A. Issa Traore in defining three classes of action, i.e., Mouse Movement, Point and Click, and Drag & Drop and in selecting the features to include. The collected data include movement speed (pixels/second) over traveled distance (pixels) per defined action, mouse movement speed and acceleration per movement direction, and distribution of double click speed.

### 1.1 Movement speed over traveled distance per defined action

Mouse Movement is simply defined as the user moving mouse from point A to point B without pressing any mouse button. One movement ends when there is no mouse input for the pre-defined duration. Point and Click is nearly identical to Mouse Movement, but ends with the user pressing any mouse button. Drag & Drop is defined as pressing a button followed by movement and release of the button. We measure the speed of mouse movement at each time slice of 50 ms and the distance at each action. Since speed varies much even for specific distance (think that when you start to move a mouse, the speed is about 0, but when you are at middle of movement, the speed would reach the maximum), each user's unique pattern can be found by focusing more on the data points near the maximum values. This is discussed in more detail in the data analysis section. Our collected data show that the speed-over-distance patterns are somewhat different for each action.

## 1.2 Movement speed per direction of movement

We measure the speed of mouse movement at each time slice of 50 ms along with the angle of movement. Although the movement speed somewhat depends on what application the user is running, there is a noticeably detectable pattern that is unique to each user. This unique pattern can be found by focusing more on the data points near the maximum values by almost same reason of 1.1.

## 1.3 Acceleration per direction of movement

This is identical to the movement speed per direction except that we measure acceleration instead of the movement speed. Note that the data set includes both negative and positive values since acceleration is a vector.

## 1.4 Distribution of double click speed

Based on our collected data, users have fairly unique and stable distribution in their double click speed or the time duration between two clicks in a double click action, more precisely.

# 2 Data analysis and machine learning

## 2.1 Discretization

After collecting enough data for a user, we pre-process the data using our program implemented in Java. Mainly, we discretize the very fine discrete variables the angle radian value, movement speed, and acceleration, in the increments of 0.05. Distance traveled is also discretized in the increments of 100 pixels so that the resulting curve is affected less by the noise in the data. Double clinking speed is also discretized in the increments of 0.01 sec. These increment values were chosen manually after trying a number of different values.

## 2.2 95 percentile

Our initial assumption was that the unique characteristics of a user's mouse movement speed per direction lay in the distribution of different values. As we analyzed the data, however, we found that the values near the maximal value are more stable over different sessions and also unique over different users. As mentioned in the mouse dynamics section, mouse moving is generally composed of acceleration, constant speed moving, and deceleration. Thus maximal speed tends to represent user's unique characteristic. We also believe that this was the case because whereas the distribution of different values may get affected by what type of application the user is using, the maximal speed values represent the user's behavioral character. Therefore, we extracted the high 95 percentile values at different angles as the signature vector of each user. This approach is also resilient to the outliers, which are considered just noises from a few jerky movements. Figure 1 is the result of taking 95 percentile values from the movement speed by movement angle data set.

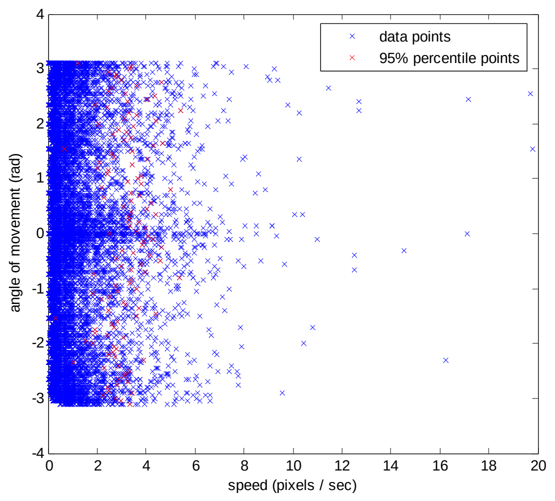

**Figure 1 Movement speed by movement angle and 95 percentile values**

## 2.3 Naive Bayes

In order to detect anomalies, we used the Naive Bayes assumption such that the values of movement speed or acceleration by angle are independent given the identity of the user. Furthermore, we assumed Gaussian distribution of the 95-percentile-values and used this for computing the likelihood of a test set value, given the identity of the user. Thus, our log likelihood is computed as

$$\ell = \sum_i \log(p(x^{(i)} \mid y)) = \sum_i \log(\frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{(\mu_y^{(i)} - x^{(i)})^2}{2\sigma^2}))$$

The value of standard deviation was hand-tuned for different features to guarantee a large difference in the resulting likelihood value depending on whether it was the data from the same user. This approach gave us much better results compared to simply comparing vector distance over the datasets. Figure 2 shows the movement speed by traveled distance values of three different users including the data over two different sessions for one user. The signature line for different users is readily observed from the figure. Table 1 shows the calculated log likelihood values where the mean values are taken from session 1 of user A and tested against the other three test sets.

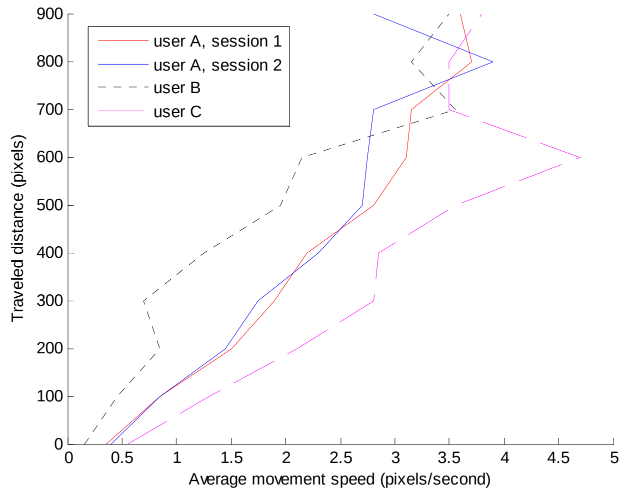

**Figure 2 Traveled distance by movement speed when performing Point and Click**

| | Against user A, session 1 |
|---|---|
| User A, session 2 | -5.253 |
| User B | -56.3763 |
| User C | -57.5950 |
| User D | -517.3138 |

**Table 1 Log likelihood for different user identities**

In processing the acceleration data, we observed a strong symmetry between the positive (acceleration) and negative (deceleration) over all users. Thus, we decided to use the absolute value of the acceleration for simplicity. Figure 3 shows the signature acceleration vectors of selected users. Here, the blue and red curves represent the signature acceleration vector of the same user over two different sessions. The overlap between the two curves is easy to observe.

Figure 4 is the signature double click speed distribution vectors.

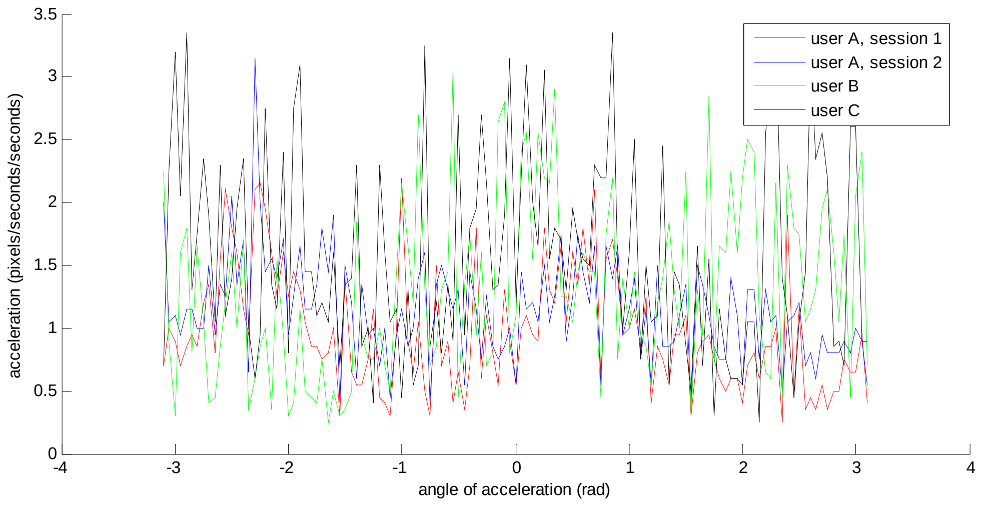

**Figure 3 Acceleration by angle for selected users**

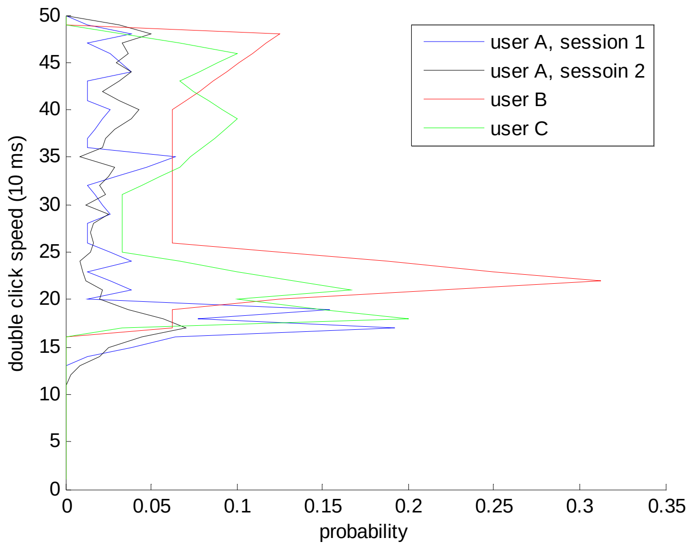

**Figure 4 double click speed distribution**

## 2.4 Final weight adjustments and anomaly detection

After computing the six log likelihoods for different features, we calculated the linear combination of the log likelihood values to get the final valuation. The weights for each likelihood was adjusted manually; it could have been done using some automatic technique such as gradient descent, but our dataset contained only about a dozen datasets from different sessions, which was not sufficient for using an automatic convergence algorithm.

After determining the weights, a threshold value for the final log likelihood value is set for each user by running some training session values against the trained session value so that each training set's final valuation is higher than the threshold. If a test set value is higher than the threshold, the set is distinguished as legitimate user, and the set is distinguished as an intruder, otherwise. If the user tends to be very stable over multiple sessions, the threshold value is set higher and the detection is likely to be more accurate unless the user deviates much from her profile, in which case a false positive occurs. On the other hand, if the user tends to show unsteady signature vectors over multiple sessions, the threshold value will be set lower in order to prevent frequent false warnings.

In order to confirm the performance of our algorithm, we tested the trained model against the data sets not used in the training process, both from the same user over a different session and from other users including some new users. Although the limited number of training sets and test sets does not allow us to draw ultimate conclusion on the performance, the algorithm worked surprisingly well on our test sets. It was able to correctly classify all the test sets fed to it. We plan to further collect large set of data and run more rigorous tests.

Another aspect we would like to see improvement on in the future is the duration of data collection time. Current span of an hour to two hours is probably too long to be actually deployed for security extension, as an intruder may harm or exploit the system in much shorter duration.

## 3 Implementation

### 3.1 Data collection

Data collection program `MouseRecorder.exe` was written in C# and Win32 assembly code. Using WinNT/XP low level global mouse hook support, it stores the features described in mouse dynamics section in four different files.

### 3.2 Data pre-processing

Data pre-processing program `Rounder.java` was implementing using Java 5.0. It reads in the files generated by the data collection program and performs discretization on the dataset.

### 3.3 Data analysis and learning

Data analysis was done exclusively in Matlab 7.0. `extractsig.m` performs 95 percentile value extractions, sorting and smoothing of the data using linear interpolation. `likelihood.m` performs the log likelihood calculation described in section 2.3.

## 4 Experiments

We collected data from eight different subjects over multiple sessions by distributing the `MouseRecorder.exe` application. Each subject ran the program in background for several hours, and each session generated four data files each ranging in size from 300KB to 1MB. In general, having larger data file resulted in smoother and more stable signature vectors and final likelihood value differed more from the likelihood obtained from the other test sets.

### References

[1] Ahmed Awad E. A. and Issa Traore (2005) Detecting Computer Intrusions Using Behavioral Biometrics, University of Victoria.