An Efficient User Verification System via Mouse Movements

Nan Zheng, Aaron Paloski, and Haining Wang
Department of Computer Science
The College of William and Mary
Williamsburg, VA 23185, USA
{nzheng,appalo,hnw}@cs.wm.edu

Abstract

Biometric authentication verifies a user based on its inherent, unique characteristics—who you are. In addition to physiological biometrics, behavioral biometrics has proven very useful in authenticating a user. Mouse dynamics, with their unique patterns of mouse movements, is one such behavioral biometric. In this paper, we present a user verification system using mouse dynamics, which is both accurate and efficient enough for future usage. The key feature of our system lies in using much more fine-grained (point-by-point) angle-based metrics of mouse movements for user verification. These new metrics are relatively unique from person to person and independent of the computing platform. Moreover, we utilize support vector machines (SVMs) for accurate and fast classification. Our technique is robust across different operating platforms, and no specialized hardware is required. The efficacy of our approach is validated through a series of experiments. Our experimental results show that the proposed system can verify a user in an accurate and timely manner, and induced system overhead is minor.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*authentication*

General Terms

Security, Human Factors

Keywords

User verification, mouse dynamics, angle-based metrics

1. INTRODUCTION

In today's Internet-centered world, the tasks of user authentication and verification have become more important than ever before [2, 7, 19, 24]. For highly sensitive systems such as online banking, it is crucial to secure users' accounts and protect their assets from malicious hands. Even in less critical systems such as desktop machines in a computing laboratory, online forums, or social networks, a hijacked session can still be misused to spread viruses or post spam, possibly damaging a user's reputation and other systems. The most common approach to securing access to systems is the use of a password [9, 11]. Unfortunately, passwords suffer from two serious problems: password cracking and password theft [25, 32]. Once a password is compromised, an adversary can easily abuse a victim's account. Thus, there is a great demand to quickly and accurately verify that the person controlling a given user's account is who the user claims to be, termed re-authentication [24].

Existing user verification and re-authentication methods require human involvement, such as providing secret answers to agreed-upon questions. Unfortunately, they only provide one-time verification, and the verified users are still vulnerable to both session hijacking and the divulging of the secret information. To achieve a timely response to an account breach, more frequent user verification is needed. However, frequent verification must be passive and transparent to users, as continually requiring a user's involvement for re-authentication is too obtrusive and inconvenient to be acceptable.

In this paper, we propose a biometric-based approach to verifying users based on passively observable mouse movement behaviors. In general, in order for a re-authentication system to be practical, it must have the following features:

- Accuracy. Not only must the system accurately identify an impostor, it must also have the probability of rejecting a true user close to zero, to avoid inconvenience to true users.
- Quick response. The system should make a quick verification decision. In other words, it should be able to distinguish a user in a timely manner.
- Difficult to forge. Even if a user's profile template is known by an impostor, it will be very hard to mimic the normal biometric behaviors in a consistent manner and then evade the verification system.

Our new approach meets all of these challenges, delivering an accurate and quick verification based on biometrics which are difficult to forge. The basic working mechanism of our approach is to passively monitor the mouse movements of a user, extract angle-based metrics, and then use Support Vector Machines (SVMs) for accurate user verification. The key feature of our approach is to exploit the point-by-point

CCS'11, October 17–21, 2011, Chicago, Illinois, USA.

angle-based metrics of mouse movements, which are relatively unique from person to person and independent of the computing platform, for user verification.

Current biometric approaches are limited in applicability: physiological biometrics, such as fingerprints and retinal scans, provide accurate one-time authentication but require specialized hardware which may be expensive or unavailable on all users' machines. On the other hand, behavioral biometrics such as keystroke and mouse dynamics hold promise, because they can be obtained from common user interface (UI) devices that nearly every user can be assumed to own. Compared with keystroke dynamics [16, 20, 23], mouse dynamics has its own advantage for two reasons. First, keystroke monitoring can record sensitive user credentials like usernames and passwords, raising much more serious privacy concerns than mouse movement monitoring. Second, keyboard is much more complex than mouse in structure, and thus keystroke dynamics are more easily affected by different kinds of keyboards in terms of shape, size, and layout.

However, to date the existing mouse-based user verification approaches have either resulted in unacceptably low accuracy or have required an unacceptably long amount of time to reach a decision, making them unsuitable for online re-authentication. In contrast to previous research, our approach introduces a novel way—point-by-point angle-based metrics—to characterize users' mouse movements, which significantly reduces verification time while keeping high accuracy.

We perform a measurement-based study, derived from 30 *controllable* users and a corpus of more than 1,000 real users in the field. Based on these two sets of mouse movement data, we evaluate the effectiveness of the proposed system through a series of experiments, using the set of angle-based metrics specifically chosen for being both platform-independent and widely variant from user to user.

In summary, the major contributions of this paper include:

- We model behavioral biometrics using mouse dynamics, and develop an efficient user verification system. It achieves high accuracy and significantly outperforms existing systems in terms of verification time.
- We propose a novel measurement strategy involving a carefully chosen set of angle-based metrics, which is relatively independent of the operating environment and capable of uniquely identifying individual users.
- We conduct an experiment involving sessions from over 1,000 unique users, which is able to re-authenticate a user within just a few clicks with a high accuracy. This promising result could lead to a practical user verification system, suitable for online deployment in the future.

The remainder of the paper is structured as follows. Section 2 reviews the background and related work in the area of mouse dynamics. Section 3 describes our data collection and measurement, including our choice of angle-based metrics over more traditional metrics. Section 4 details the proposed classifier for user verification. Section 5 presents our experimental design and results. Section 6 discusses issues which arise from the details of our approach, and Section 7 concludes.

2. BACKGROUND AND RELATED WORK

The underlying principle of biometric-based user authentication is centered on "who you are". This is very different from conventional user authentication approaches, which are mainly based on either "what you have" or "what you know". Unfortunately, a physical object such as a key or an ID card can be lost or stolen. Similarly, a memorized password could be forgotten or divulged to malicious users. Conversely, a biometric-based approach relies on inherent and unique characteristics of a human user being authenticated. The biometrics can never be lost or forgotten, nor can another user easily steal or acquire them. This makes biometrics very attractive for user authentication.

Biometrics are categorized as either physiological or behavioral [31]. Physiological biometrics, like fingerprint and facial recognition, have attracted considerable attention in research [13, 18]. The downside of these biometrics is that they need specialized hardware, which can be problematic for wide deployment. For user authentication over the Internet, one cannot always rely on the existence of hardware at the client side. In contrast, behavioral biometrics using human-computer interaction (HCI) can record data from common input devices, such as keyboards and mice, providing user authentication in an accessible and convenient manner.

Behavioral biometrics first gained popularity with keystroke dynamics with Monrose et al.'s work on password hardening in 1999 [19]. Later on, Ahmed and Traore [1] proposed an approach combining keystroke dynamics with mouse dynamics. Mouse dynamics for re-authentication have been previously studied as a standalone biometric by Pusara and Brodley [24]. Unfortunately, their study is inconclusive with only eleven users involved, prompting the authors to conclude that mouse biometrics are insufficient for user re-authentication. Our study relies on an improved verification methodology and far more users, leading us to reverse their hypothesis.

In Ahmed et al.'s work [1, 2, 21], while achieving very high accuracy, the number of mouse actions needed to verify a user's identity[1] is too high to be practical. Specifically, their experiment requires as many as 2,000 aggregate mouse actions before a user can be recognized, and is not practical for real-time deployment. Conversely, we aim to provide a system suitable for online re-authentication. We first employ a finer-grained data collection methodology, allowing us to collect far more data in less time. We also employ support vector machines (SVMs), which are considerably faster than the neural networks employed in [2, 21]. Thus, our system can make a decision in just several mouse clicks.

More recently, a survey covering the existing works in mouse dynamics has been conducted with a comparative experiment [15]. It points out that mouse dynamics research should be more aware to reduce verification time and take the effect of environmental variables into account. It can be seen later that, compared to other works, our approach also achieves high accuracy but only requires a small amount of biometric data. Moreover, we explore the effects of environmental factors (different machines, mice, and time) and show that our approach is relatively robust across different operating environments and times.

Graphical passwords [7, 27] are a related form of user authentication, relying on HCI through a pointing device to

[1]referred to as *session length* in [2, 21].

authenticate a user. Mouse dynamics differ in that they differentiate between users by *how* the users move and click the mouse, rather than *where* the users click. Graphical passwords record where the user clicks on the screen, and subsequently use this sequence as a substitute password. Systems such as these are complementary to our work, and can be deployed together. For instance, one might employ a graphical password system while passively recording a user's mouse dynamics, utilizing the passively recorded measurements as a secondary fail-safe to verify the user's identity. This is similar in spirit to using keystroke dynamics with password hardening as in [19].

3. MOUSE MOVEMENT MEASUREMENT AND CHARACTERIZATION

3.1 Data Collection

We collect two sets of data. The first data set is from a controllable environment, referred to as the controllable set; while the second data set is from an online forum in the field, referred to as the field set. We have obtained approval from the Institutional Review Board (IRB) of our university, which ensures the appropriate and ethical use of human input data in our work.

For the controllable set, 30 users are invited personally to participate in the data collection. They are from different ages, educational backgrounds, and occupations. We intentionally set a normal environment for these users and inform them to behave as naturally as possible. Mouse movement data are recorded during their routine computing activities. These activities range among word processing, surfing the Internet, programming, online chatting, and playing games. We make use of a logging tool RUI [17] to record their mouse movement activities.

For the field set, more than 1,000 unique forum users' mouse movements are recorded by JavaScript code, and submitted passively via AJAX requests to the web server. On one hand, these users are anonymous but identifiable through unique login names. However, there is no guarantee on the amount of data collected for a certain user. A forum user could be logged in for a long time with frequent mouse activities, or could perform just one click and then leave. On the other hand, the breadth of this corpus of users is utilized to serve as the base profile for both training and testing purposes.

The raw mouse movements are represented as tuples of timestamp and Cartesian coordinate pairs. Each tuple is in the form of $\langle \text{ACTION-TYPE}, t, x, y \rangle$, where ACTION-TYPE is the mouse action type (a *mouse-move* or *mouse-click*), t is the timestamp of the mouse action, x is the x-coordinate, and y is the y-coordinate. Timestamps in our data collection are collected in milliseconds.

3.1.1 Data Processing

The purpose of preprocessing is to identify every point-and-click action, which is defined as continuous mouse movements followed by a click. *Continuous* mouse movements are series of mouse movements with little or no pause between each adjacent step. Within the ith point-and-click action for a user c, we denote the jth mouse move record as $\langle \text{MOUSE-MOVE}, t_i, x_i, y_i \rangle_{c,j}$, where t_i is the timestamp of the ith mouse movement. Based on the record that belongs to each point-and-click action, we calculate angle-based metrics.

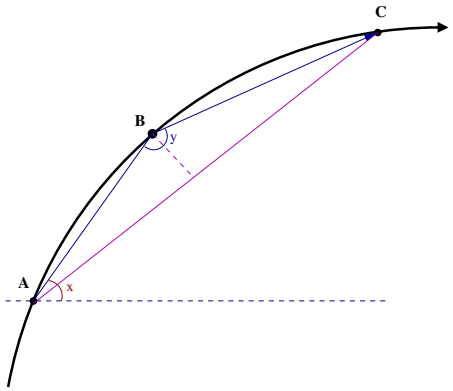


Figure 1: Illustration of angle-based metrics.

3.1.2 Metrics

To analyze the mouse movement data, we define three fine-grained angle-based metrics: direction, angle of curvature, and curvature distance. These newly-defined metrics are different from the conventional metrics, such as speed and acceleration, and can accurately characterize a user's unique mouse moving behaviors, independent of its running platform.

- Direction. For any two consecutive recorded points A and B, we record the direction traveled along the line $\overrightarrow{AB}$ from the first point to the second. The direction is defined as the angle between that line $\overrightarrow{AB}$ and the horizontal (see angle x in Figure 1).
- Angle of Curvature. For any three consecutive recorded points A, B, and C, the angle of curvature is angle $\angle ABC$; i.e., the angle between the line from A to B ($\overrightarrow{AB}$) and the line from B to C ($\overrightarrow{BC}$) (see angle y in Figure 1).
- Curvature Distance. For any three recorded points A, B, and C, consider the length of the line connecting A to C ($\overrightarrow{AC}$). The curvature distance is the ratio of the length of $\overrightarrow{AC}$ to the perpendicular distance from point B to the line $\overrightarrow{AC}$ (see the perpendicular lines in Figure 1). Note that this metric is unitless because it is the ratio of two distances.

As a comparison, we list the definition of two traditional mouse movement metrics, speed and pause-and-click, as follows.

- Speed. For each point-and-click action, we calculate the speed as the ratio of the total distance traveled for that action divided by the total time taken to complete the action.
- Pause-and-Click. For each point-and-click action, we measure the amount of time between the end of the movement and the click event. In other words, this metric measures the amount of time spent pausing between pointing to an object and actually clicking on it.

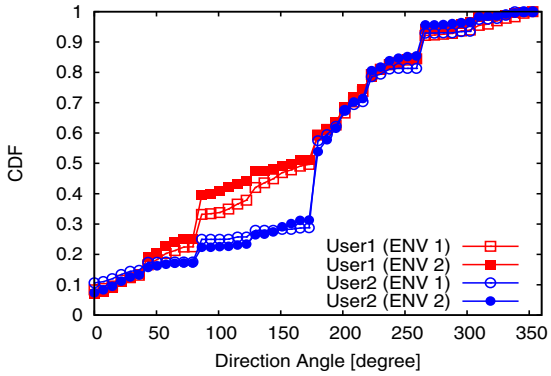


Figure 2: Direction Angle metric plotted for two different users on two different machines each.

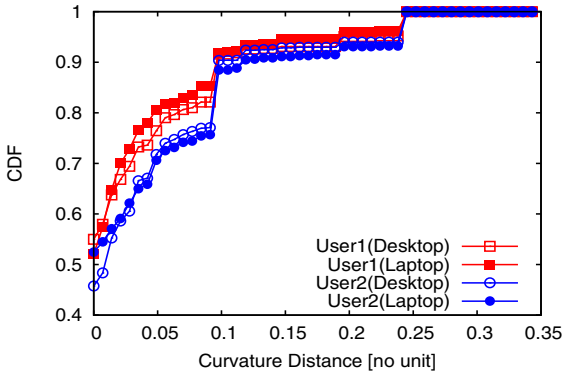


Figure 4: Curvature Distance metric plotted for two different users on two different machines each.

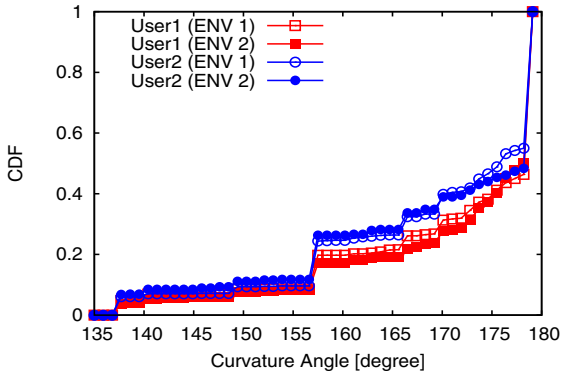


Figure 3: Angle of Curvature metric plotted for two different users on two different machines each.

3.2 Mouse Movement Characterization

3.2.1 *Dependence on Different Platforms*

One problem we came across in analyzing our data is that it may be difficult or meaningless to compare two users who are using very different machines. The entire user's environment can affect its data: the OS used, screen size and resolution, font size, mouse pointer sensitivity, brand of mouse, and even the amount of space available on the desk near the mousepad. Metrics such as speed and acceleration, then, are poor choices for comparison between users of arbitrary platforms. This is because these two metrics can be skewed by differences in screen resolution and pointer sensitivity. On the other hand, metrics such as pause-and-click are highly dependent on the content a user is reading. For example, a user tends to pause longer before clicking a link on a rich content page such as a wiki article, and hesitates for a much shorter time before clicking a "submit" button.

This makes a good case to use angle-based metrics for arbitrary user comparison instead. Direction and angle of curvature are not based on screen size or any other element of the user's environment, and thus are relatively platform-independent. Likewise, curvature distance is a ratio of distances on the screen, and thus self-adjusts for the user's specific environment. A ratio can be compared to another user's ratio across platforms.

Figures 2, 3, and 4 show the comparison of two users with angle-based metrics. We can see that the cumulative distribution function (CDF) curves for the same user's individual data are very similar and well synchronized in shape, even across platforms. This indicates that angle-based metrics are relatively stable on different platforms.

3.2.2 *Uniqueness of Angle-Based Metrics Across Users*

The other distinctive feature of angle-based metrics is that they are unique across users. Not only does the same user have very similar angle-based results on different platforms, but different users have clearly different angle-based results, even on similar platforms.

Again, as Figures 2, 3, and 4 show, even though each user's CDF is consistent across different platforms, there is a distinct gap between different users' CDF curves, even on the same platform. As a comparison, Figure 5 shows the CDF curves with respect to the speed of the two users, a more commonly-used metric. Figure 6 shows the CDF curves with respect to the pause-and-click of the two users. While the different users' CDF curves in both speed and pause-and-click are closely coupled on the same environment, there is a distinct gap between the same user's two curves for different environments. Since the closest matching curve for either user is the curve of the *other* user under the same environment, it can be very hard to uniquely differentiate people using these metrics.

Together with the platform independence discussed above, this makes angle-based metrics superior to speed and pause-and-click for user verification. Note that for easy presentation, we only compare the difference of the mouse dynamics between a pair of users. However, the similar observation holds for the other users.

3.2.3 *Distance Between Distributions*

Using the *distance* between two probability distributions, we further verify if the angle-based features of a user remain relatively stable across different types of mice, platforms, and time, in comparison with those of the other controllable users.

We define the distance between two probability distributions as follows. Since angle-based features are continuous

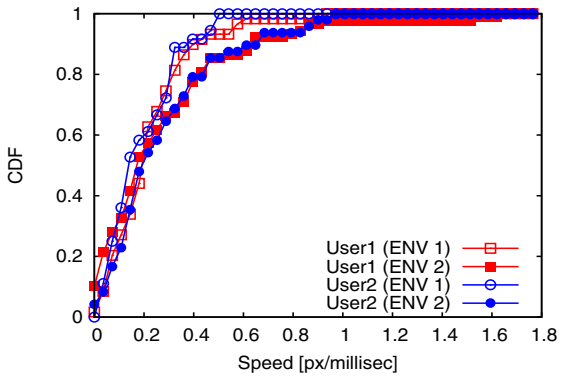


Figure 5: Speed metric plotted for two different users under two different environments each.

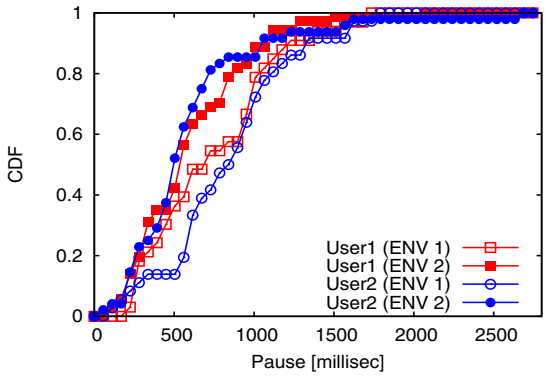


Figure 6: Pause-and-click metric plotted for two different users under two different environments each.

variables, we divide their whole range into discrete intervals, called *bins*, and calculate the probability density functions (PDFs) regarding to each bin. Consider two distributions: the first is expressed as PDF $\{p_1, p_2, ..., p_n\}$, where p_i represents the probability of falling into the ith bin; the second distribution is expressed similarly as PDF $\{q_1, q_2, ..., q_n\}$. The *distance* between the two distributions is the accumulated deviation from each other over all bins:

$$D(p, q) = \sum_i |p_i - q_i|.$$

Of course, the distance here is dependent on the size of each interval. The smaller we divide the interval, the more subtle differences the distance reflects. However, the bin should not be so small as to enlarge noise.

Figure 7 plots the distances of a user from the other users, as well as from itself using a different mouse on a different machine at very different time. Each user's PDF is computed over 1,000 curvature angles randomly selected from its data, and loops for 10 times. The height of each bar is the average distance from the target user (labeled as user 1) in setting A (indexed by 1-A in the figure), and each error bar is standard deviation over the 10 times. Data 1-A, 1-B, and 1-C are all from user 1. The details of these three settings are listed in Table 1. Moreover, data 1-B are recorded

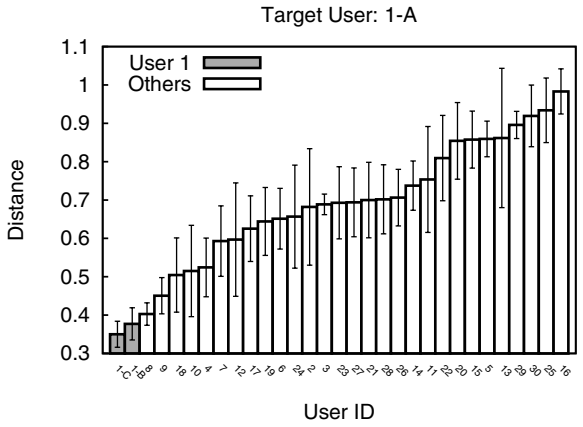


Figure 7: Distances from one user's curvature angle distribution to those of others, as well as to itself in different settings.

Setting	Machine Type	Mouse Type
1-A	Dell Precision T3500	Dell MOC5UO Two-Button Scroll-Wheel
1-B	Apple Macbook MB990LL/A	Apple A1152 One-Button Trackball
1-C	Apple Macbook MB990LL/A	Dell MOC5UO Two-Button Scroll-Wheel

Table 1: Setting Details

two and half months later than data 1-A, and data 1-C are recorded two days later than data 1-B.

It is clear that the distances from user 1 to itself using different mice, at different machines, and over different times are the two smallest in the figure, i.e., both are smaller than the distances from user 1 to any other users with the same setting. This implies that the angle-based behavior of a user has its own inherent pattern which is relatively stable across different settings and times; meanwhile, it is also distinguishable from the behaviors of other users. Note that in Figure 7, the distance values between user 1-A and some users are very close to each other (e.g. the distance value of user 23 and that of user 27). However, it does not indicate that the behaviors of those users are similar, because the definition of distance here is an accumulation of deviations at different bins. The dynamics of two users' PDFs could be totally different, but at the same time both deviate equally from a third user.

Note that to achieve accurate measurement results, there are two prerequisites in characterizing mouse movement under different environments. First, the polling rates of mouse recorders at different platforms should be configured to the same level. Second, prior to characterizing a user's mouse movement, sufficient mouse events must be collected to create a profile of the user's mouse movement. In particular, we observed that 1,000 mouse actions (which on average can be collected in 2 hours) are large enough to profile a user's mouse behavior well.

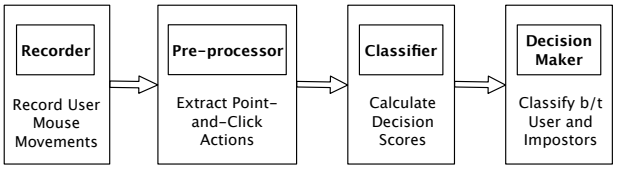


Figure 8: System Architecture

3.2.4 *Number of Mouse Clicks in a Real Session*

For the field set, drawn from 1,074 real users on an online forum over the course of an hour, we recorded an average of 15.14 clicks per user session. Note that because this data was gathered over a one-hour window, this value is a lower bound on the actual number of clicks in an average user session. Any user who was logged in before the window began or stayed logged in after the window ended (including users who stayed logged in longer than an hour) would necessarily have more clicks than recorded. Thus, the actual average is almost certainly higher, and probably much higher.

The number of mouse clicks per user session is closely related to verification time. With the average number of mouse clicks per session being about 15, a verification system based on mouse dynamics must be able to identify a user in fewer than 15 clicks in order to ensure that, on average, a decision is made within one user session. As shown in Section 5, our approach can verify a user's identity with high accuracy in only 15 clicks, so our system can reliably make the right decision before the user logs off in a majority of cases.

4. SYSTEM ARCHITECTURE

As shown in Figure 8, our proposed user verification system consists of four components—recorder, preprocessor, classifier, and decision maker.

The design of the first two components is straightforward. The main task of the recorder is to record users' mouse movements, while the preprocessor computes the angle-based metrics based on the recorded raw data. The focus of this section is on the design of the classifier and that of the decision maker.

4.1 SVM Classifier

We choose Support Vector Machines (SVMs) as our classifier to differentiate users based on their mouse movement dynamics. SVMs have been successfully used in resolving real-life classification problems, including handwritten digit recognition [29], object recognition [22], text classification [14], and image retrieval [28]. In general, SVMs are able to achieve comparable or even higher accuracy with a simpler and thus faster scheme than neural networks.

In the two-class formulation, the basic idea of SVMs is to map feature vectors to a high dimensional space and compute a hyperplane, which separates the training vectors from different classes and further maximizes this separation by making the margin as large as possible. SVMs classify data by determining a set of support vectors, which are members of the set of training inputs outlining a hyper plane in feature space [30].

For a binary classification problem, given l training samples $\{\mathbf{x}_i, y_i\}, i = 1, \ldots, l$, each sample has d features, expressed as a d-dimensional vector $\mathbf{x}_i$ ($\mathbf{x}_i \in \mathbb{R}^d$), and a class label y_i with one of two values ($y_i \in \{-1, 1\}$). A hyperplane in d-dimensional space can be expressed as $\mathbf{w} \cdot \mathbf{x} + b = 0$, where $\mathbf{w}$ is a constant vector in d dimensions, and b is a scalar constant. We aim to find a hyperplane that not only separates the data points but also *maximizes* the separation. As Figure 9 shows, the distance between the dashed lines is called the *margin*. The vectors (points) that constrain the width of the margin are the *support vectors*. The formulation of our binary class SVM problem is to:

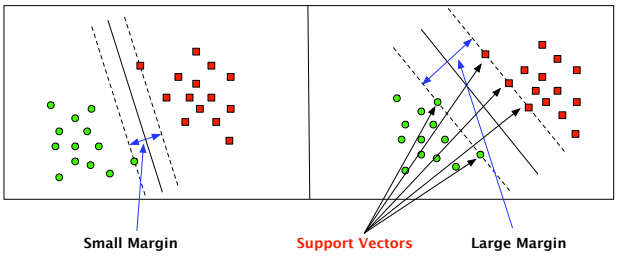


Figure 9: SVM hyperplane in two dimensional feature space [10].

$$\text{minimize:} \quad W(\alpha) = -\alpha^T \mathbf{1} + \frac{1}{2}\alpha^T H \alpha,$$
$$\text{such that:} \quad \alpha^T y = 0, \qquad \mathbf{0} \leq \alpha \leq C\mathbf{1},$$

where matrix $(H)_{ij} = y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$, α is the vector of l non-negative Lagrangian multipliers to be determined, and C is a constant. This minimization problem is known as a *Quadratic Programming Problem (QP)*, which is well studied with many proposed efficient algorithms.

In reality, not all data points can be linearly separated as we assumed. To handle this issue, SVMs use a "kernel trick". The data are pre-processed in such a way that the problem is transformed into a higher dimension, where they are linearly separable in the new feature space. Given a mapping $\mathbf{z} = \phi(\mathbf{x})$, and defining the *kernel function* as $K(\mathbf{x}_a, \mathbf{x}_b) = \phi(\mathbf{x}_a) \cdot \phi(\mathbf{x}_b)$, our classifier would be

$$f(\mathbf{x}) = \text{sign}\left(\sum_i \alpha_i y_i (K(\mathbf{x}_i, \mathbf{x})) + b\right).$$

A popular choice of kernel function is the Gaussian Radial Basis Function (RBF) $K(\mathbf{x}_a, \mathbf{x}_b) = \exp\left(-\gamma \left\|\mathbf{x}_a - \mathbf{x}_b\right\|^2\right)$, where $\gamma > 0$, and is a tunable parameter. In practice, RBF is a reasonable first choice among other kernels, due to its generality and computational efficiency [6].

Thus, the procedure to resolve a classification problem using SVMs is: (1) choosing a kernel function, (2) setting the penalty parameter C and kernel parameters as well, if any, (3) resolving the quadratic programming problem, and (4) constructing the discriminant function from the support vectors. In particular, we view the user verification problem as a two-class classification problem, and the learning task is to build a classifier based on the user mouse movements.

In our proof-of-concept implementation, we used the open source SVM package LIBSVM 3.0 [6] for building the prototype. LIBSVM is an integrated tool for support vector classification. We used the default RBF kernel and the cross-validation to find the best parameter C and γ. In our study, all impostors are classified as +1, and normal data are classified as -1. The detailed experiment setups will be discussed in Section 5.

4.2 Decision Making

In the design of the decision maker, we use two mechanisms, threshold and majority vote, to further improve verification accuracy.

4.2.1 Threshold

The threshold determines how SVMs' output is interpreted: a value over the threshold indicates an impostor, while a value under the threshold indicates a true user. To make a user verification system deployable in practice, minimizing the probability of rejecting a true user is sometimes even more important than lowering the probability of accepting an impostor.

By default, in a binary classification problem with labels in $\{+1, -1\}$, LIBSVM outputs a score called a *decision value* for each testing sample. If the decision value is greater than 0, the sample is classified as $+1$, otherwise it is classified as -1.

4.2.2 Majority Votes

To build the profile for an authorized user, in training, we randomly pick $m/2$ samples that belong to the user, labeled as negative (non-impostor), and another random $m/2$ from the field set, labeled as positive (impostor). We employ a simple *majority vote* decision making scheme in order to improve and stabilize the verification accuracy. Specifically, before verifying if a sample belongs to the target user, we train the user's profile $2n + 1$ times. Each time the training samples are different since they are randomly selected. In this way, there will be $2n + 1$ votes about the predicted label for each testing sample.The label that is voted by the majority, i.e., with greater than n votes, will be the final predicted label. With majority votes, the decision maker can significantly reduce the randomness of the results and improve verification accuracy.

5. EXPERIMENTAL EVALUATION

In this section, we evaluate the effectiveness of our mouse movement based verification system through a series of experiments, in terms of verification accuracy, verification time, and system overhead. The verification accuracy of our system is measured using (1) the false reject rate (FRR), which is the probability that a user is wrongly identified as an impostor, and (2) the false accept rate (FAR), which is the probability that an impostor is incorrectly identified as a user. Here we define a block as follows:

Block (or detecting block) A *block* is composed of mouse movements in a group of point-and-click actions. Statistical features are calculated based on all mouse movements in one *block*.

Note that choosing different sizes (that is, choosing different number of point-and-click actions contained) of a *block* greatly affects verification accuracy, in terms of FAR and FRR.

The verification time is the mean time needed to detect an identity mismatch. This corresponds to the sum of the time for the user to generate mouse actions needed to make a decision, the time to extract the features for this session, and the time to classify the identity. In our approach, the number of mouse actions needed to make a decision equals to the number of clicks contained in one *block*. As described before, a *block* corresponds to one sample in either training or testing. Verification time is determined by the total number of actions needed to make decisions and the average time cost per action. In general, the larger the number of actions required for decisions and the higher the average time cost per action, the longer the verification time becomes.

5.1 Experimental Setup

As described in Section 3.1, our experiments are based on two sets of data. The first data set is collected in controllable environments. A total of 81,218 point-and-click actions are captured, with an average 5,801 point-and-click actions per user. Overall, 150 hours of raw mouse data are collected. The second data set is recorded from 1,074 anonymous users in an online forum for one hour.

These two data sets serve different purposes. A *target user* is selected from the first data set as the user to be verified, while forum users from the second data set are used as the background. Whereas we can identify a forum user based on its unique login name, the lack of guarantee on its collected data makes it unsuitable to be the verified target. The preprocessor extracts each user's point-and-click actions and computes the angle-based metrics corresponding to each point-and-click. Each of those generated files containing point-and-click actions is divided into two halves. The training data will be extracted from the first half, while the testing data will be from the second half. Therefore, there is no overlap between the training data and the testing data.

5.2 Verification Results

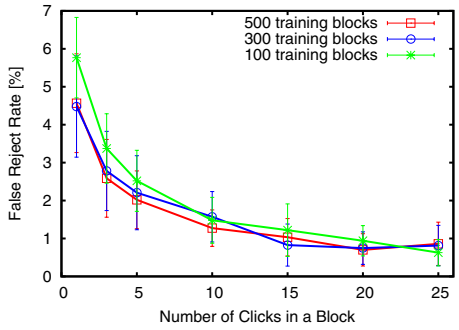
We construct our classification model based on self and non-self discrimination. That is, for a given user, its profile is learned from a certain number of its own mouse movement samples and an equal number of others' mouse movement samples. Therefore, the training data is composed of positive samples and an equal number of negative samples. We train a separate model for each user in the controllable set. In the training file for a given user, a negative case is a block of point-and-click actions that belongs to itself, while a positive case is a block of clicks that belongs to *others*. Here, *others*' mouse movement blocks are randomly chosen from the forum set, due to its large user population.

There are four configurable parameters in our system: the size of a detecting block, the size of the training data, the threshold, and the number of votes. The first two are associated with the SVM training process. Increasing the threshold value directly lowers false reject rate (FRR), but at the cost of raising false accept rate (FAR). Increasing the number of votes improves verification accuracy in terms of both FRR and FAR, but increases the verification time.

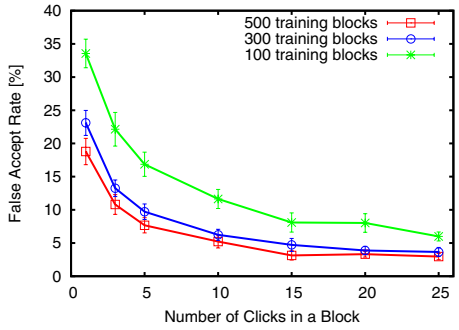
To fully evaluate verification accuracy, we conduct two sets of experiments. In the first set of experiments, we test our classification model trained in the same environment. In the second set of experiments, we test the classification model trained in a different environment.

5.2.1 Self and Non-Self Discrimination in Same Environment

We first configure the number of mouse clicks per block and the size of the training data. The FRR and FAR with different sizes of detecting blocks and training data are shown in Figures 10(a) and 10(b), respectively. These tests are

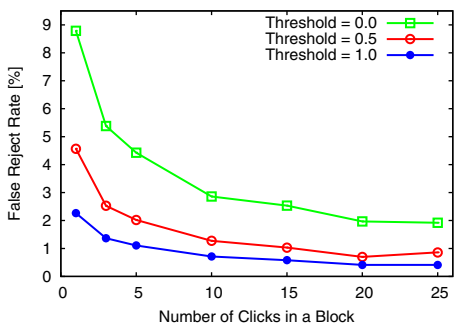


(a) False Reject Rate

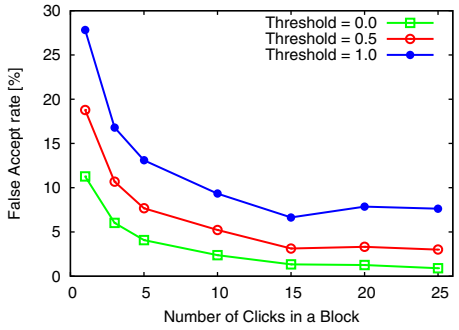


(b) False Accept Rate

Figure 10: Variation of FRR and FAR with the number of clicks. Error bars indicate standard deviation.



(a) False Reject Rate



(b) False Accept Rate

Figure 11: Variation of FRR and FAR with threshold.

performed with the default threshold of 0.5 and 5 out of 9 (5/9) majority votes. As larger detecting block size and training data are provided, the SVM classifier becomes more accurate, but we see diminishing returns in accuracy as the number of actions increases, e.g., going from 10 clicks to 25 clicks requires 150% more input but only provides a relatively small increase in verification accuracy; also going from 100 training samples to 300 training samples requires 200% more data, but only returns a relatively small increase in verification accuracy.

With the SVM classifier configured, the last two parameters, the threshold and the number of votes, determine the overall performance of the system.

The threshold can be increased or decreased from the default value of 0.0 to bias the classifier towards authentic users or impostors, lowering the FRR or FAR, respectively. As mentioned in Section 4.2.1, this is a tradeoff between user inconvenience level and system security level. After multiple tests, we observe that setting the threshold value to 0.5 yields a false reject rate 1% on average. Therefore, throughout this paper, we only show results with a threshold value of 0.5. Setting the threshold value affects both FRR and FAR. Figure 11 shows that increasing the threshold value greatly lowers FRR at different sizes of a detecting block.

The use of majority votes increases the verification accuracy of the system, in terms of both FRR and FAR. Figure 12 shows the improvement of FRR by majority votes. However, increasing the number of votes means longer verification time, since for n votes the classifier needs to be run n times. Figure 12 indicates that the improvement by 2/3 majority votes is comparable to that of 3/5 majority votes.

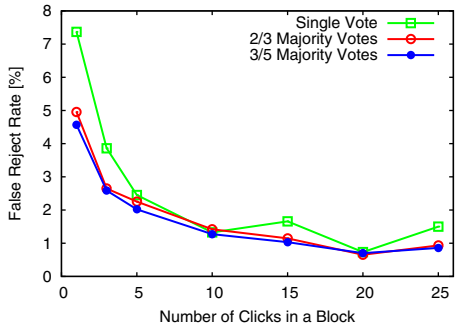
With a fully configured system (500 training blocks, a threshold of 0.5, and 3/5 majority votes), Table 2 lists FRR and FAR averaged over 30 users in the controllable set. It can be seen that, if there are 25 clicks in one *block*, the average false reject rate is 0.86%, so there is only a little chance that an authenticated user is misclassified as an impostor. Meanwhile, we achieve an average false accept rate of 2.96%.

Number of Clicks	FRR	FAR
1	4.57%	18.79%
3	2.59%	10.81%
5	2.02%	7.67%
10	1.27%	5.23%
15	1.03%	3.13%
20	0.70%	3.32%
25	0.86%	2.96%

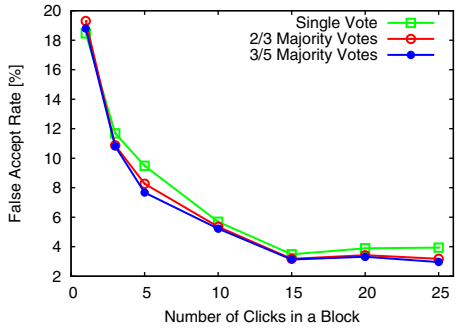
Table 2: Variation of FRR and FAR with different number of clicks in one block

5.2.2 *Self and Non-Self Discrimination in Different Environment*

To verify that our approach also works in different environments with different machines, we conduct another self vs. non-self discrimination experiment on two different machines. More specifically, the user's profile is trained from its mouse movements in a work environment on a desktop,



(a) False Reject Rate



(b) False Accept Rate

Figure 12: Variation of FRR and FAR with majority votes.

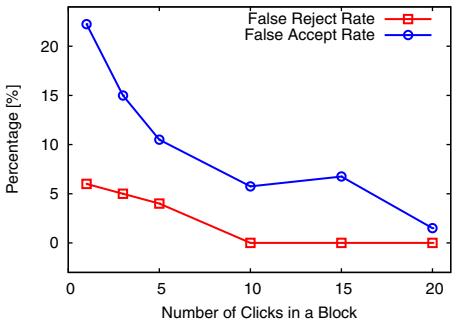


Figure 13: FRR and FAR for one user profiled on one platform and tested on another platform.

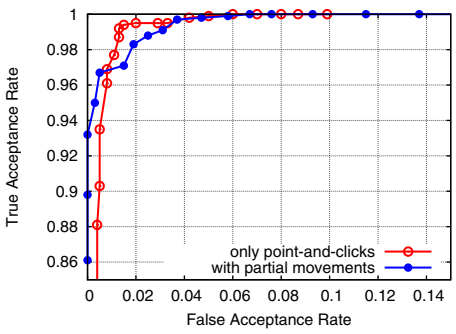


Figure 14: ROC curves with and without partial movements.

while its mouse movements in a home environment on a laptop are tested. The testing user base includes 5 users. The corresponding FRR and FAR are shown in Figure 13, each represented by a single curve, respectively. Note that a user is profiled on one platform but tested on a different one, and hence a single plot shows the result of the test.

It can be seen that our approach works well across different environments and platforms. It further confirms that our classification model indeed captures those features that are intrinsic to a user and not affected by environmental factors.

5.2.3 *Partial Movements*

Partial movements are a series of continuous mouse movements *without* ending in a click. On one hand, unlike point-and-click actions that have a certain object to reach as a target, some partial movements could be aimless. For example, a user may move its mouse just to stop the screen saver when watching a video. Thus, we observe that the movements of point-and-clicks demonstrate a more consistent pattern than those of partial movements. However, most partial movements are intentionally performed. For example, a user may often move its mouse to aid reading. In addition, some partial movements are just as well-motivated as point-and-clicks. A user could start moving the mouse to a link, but then decide not to click on it.

Moreover, in a real user session, partial movements occur much more frequently than point-and-clicks. From the forum data we collected, there are only 0.53 mouse clicks per minute on average, but 6.58 partial mouse movements per minute. Figure 14 shows the comparison of ROC (Receiver Operating Characteristic) curves with and without partial movements for a randomly selected user (other users' ROC curves are similar).

Suppose we choose 20 mouse clicks in a detecting block. On one hand, without partial movements − that is, when only point-and-clicks are included − the EER (equal error rate) is 1.3%; with partial movements, the EER increases to 1.9%. On the other hand, using partial movements can lower the average verification time by one order of magnitude (about 12 times) in our experiments. Therefore, using partial movements will significantly reduce verification time, but at the cost of accuracy degradation.

5.2.4 *Subtleties on Verification Time*

The verification time is the time required by a verification system to collect sufficient behavioral data and then make a classification decision. The value of the verification time heavily depends on two factors: (1) the number of required mouse clicks (or mouse movements if partial movements are included) in a detecting block, and (2) how frequently a user generates mouse actions. If the number of mouse actions in a detecting block is already configured, the verification time will be mainly determined by the latter.

There are two verification scenarios, static and continuous, when estimating the number of mouse actions a user generates per unit time. In the scenario of *static* verification, a user is required to perform a series of mouse movements and its mouse data is verified within a certain amount of time (e.g., login time). A good example of this scenario is a click-

based graphical password for user login, where five clicks are estimated to be made in no more than 25 seconds [4, 8]. This implies that the verification time will be less than 100 seconds if 20 mouse clicks are needed in a testing block. By contrast, in the scenario of *continuous* verification, a user's mouse data is continuously collected and verified throughout the entire session. This is non-intrusive to users and meets the goal of passive monitoring. However, the frequency of user mouse actions varies significantly in different sessions. In general, the average frequency of user mouse actions will be much lower than that of the static scenario. The reason is that often there is a period of silence between a user's previous and next mouse actions while a user is reading or typing. An observation from the forum data we collected indicates, on average, it takes 1.89 minutes for one mouse click to happen. If we choose 20 mouse clicks in a detecting block, the verification time could be as long as 37.73 minutes; however, the verification time will be reduced to 3.03 minutes if partial movements are used.

5.2.5 System Overhead

The verification system can be deployed in two different scenarios. In the first scenario, for example, it can be used for access control in a computer lab or on a personal computer. In this case, it will be installed at the client side by the system administrator. In this standalone scenario, normally only a single user is present for verification at any one time, and the end host has a plenty of resources to fulfill the verification tasks. Thus, the performance impact caused by the verification on the host is minor.

In the second scenario, for an online application such as banking account verification, the system will be placed at the server side, with JavaScript embedded inside users' account home page. Deployed at the server side, our system needs to be able verify hundreds or even thousands of users simultaneously in real-time. Thus, system overhead becomes an issue, and the system must be efficient in terms of memory and CPU costs.

We first estimate the memory overhead of the verification process. We profile the verification process using the Linux tool `valgrind`, and find out that it only consumes 3.915 KBytes of memory per testing block during the operation. The prototype of our system is designed to use a single-thread multiple-client model with time-multiplexing, and thus only one process is used. The primary memory cost is to accommodate the accumulated user-input actions and SVM outputs for each online user. A single user-input action consumes 12 bytes, 4 bytes each for the 3 angle-based metrics. A detecting block of 10 user-input actions consumes 120 bytes, and this is the per-user memory requirement. If 120 bytes is scaled to 1,000 online users, it is only 117.19KBytes in total, which is negligible considering that online websites currently store the user name, password, IP address, security questions and literally dozens of other attributes for each user.

The computational overhead is the sum of CPU costs in pre-processing and detecting (including classifying and decision making). The pre-processing on 15 minutes' user inputs from more than 1,000 users in a typical website, including 5,270 point-and-click actions, takes only 20.937 seconds. The CPU cost is measured on a Pentium 4 Xeon 3.0Ghz, using the Linux command `time`. Note that the forum trace is collected during 15 minutes from about 1,000 online users, implying that it takes about 23.3 milliseconds to process data generated in one second. At the same time, the verification takes only 229 milliseconds over 5,801 point-and-click actions. In comparison to pre-processing, this is negligible. Therefore, the induced computational overhead is minor on the server.

In terms of disk space for storing user profiles, the signature of a single user profile generated by the training process consumes 203.32KBytes. If it is scaled to 100,000 users, that is 19.4GBytes, which is very affordable at a personal computer, let alone a high end server.

5.3 Comparison with Existing Works

We compare our evaluation results with those of existing works in terms of verification accuracy and time, which are listed in Table 3. As described in Section 5.2.4, the verification time is highly dependent on the number of mouse events needed to make a decision, the type of mouse events used (mouse click, mouse move, or drag-and-drop), as well as how fast a user generates mouse events. Even for the same user at different times, the number of mouse events per unit time varies a lot. However, to the best of our knowledge, our work is the first to achieve high accuracy with a reasonably small number of mouse events.

6. DISCUSSION

Since our verification system records users' mouse movements and clicks, privacy concerns may arise. However, compared to keystroke dynamics, the amount of personal information included in mouse dynamics is minimal. In the process of recording keystrokes, the system would record the user's passwords, user names, and other sensitive textual information. By contrast, recording mouse dynamics only reveals the physical movements of a mouse and its clicks within a certain period of time, giving away little to no information about user credentials. Even with the perfect knowledge of a user's mouse movements, the only things an adversary can figure out are when the user clicked and on which position of the screen. Thus, we believe that our verification system will not cause any privacy violations.

In general, mouse-dynamics-based re-authentication techniques are robust against online forgery. A person's unique mouse dynamics are similar to its signature, and like a signature, it is difficult to mimic even with the complete knowledge of the original. In fact, a user's mouse dynamics is a continuous process, making it much harder to forge than a signature. Unlike forging a signature, which only has to be accomplished once, the adversary of our verification system would need to mimic the true user's mouse patterns continuously for the entire length of the session. It is extremely difficult for one user to force itself to consistently move the mouse in such a mechanical way that it matches specific angles, even if those metrics are known ahead of time. Thus, mouse dynamics generally and our fine-grained angle-based approach in particular, are very robust against online forgery. However, how vulnerable mouse-dynamics-based approaches are to offline attacks, especially generative attacks which create concatenative synthetic forgeries [3], is still an open question and will be investigated in our future work.

The retraining of our classifier is necessary to deal with sudden, perhaps temporary, changes in a user's mouse profile. If the user's behavior suddenly changes, due to an unex-

Source	FRR	FAR	Data required	Settings	Notes
[2]	2.4649%	2.4614%	2000 mouse actions	Continuous	Free mouse movements
[21]	0%	0.36%	2000 mouse actions	Continuous	Free mouse movements
[12]	2%	2%	50 mouse strokes	Static	Mouse movements from a memory game
[24]	1.75%	0.43%	Not specified	Continuous	Applies to a certain application
[26]	11.2%	11.2%	3600 mouse curves	Continuous	Free mouse movements
Ours	1.3%	1.3%	20 mouse clicks	Continuous	Free mouse movements

Table 3: Comparison with Existing Works

pected complication such as a sprained wrist, the difference in mouse usage could be large enough for the user to be unrecognizable by the verification system. The system would classify the user as an impostor and prevent that user from accessing its own account. While these sorts of occurrences are relatively rare, to avoid the possible rejection, the user can easily appeal to a system administrator to retrain the classifier with the user's new movement patterns. Once the user's behavior returns to normal (for example, the user's wrist heals), we can either retrain the system again, or simply reuse the previous classifier if a backup is available.

Although our approach is relatively independent of the running environments, it is sensitive to the polling rate of mouse movement recording. In the mouse data collection, a continuous mouse movement is discretized to a set of mouse coordinates, which are sampled at a certain rate. Thus, the measured resolution of mouse movements is dependent on the polling rate of a recorder. The faster the polling rate is, the more fine-grained movements we capture. For example, given a mouse movement curve, a high polling rate can render a smooth accurate shape, but a low polling rate more likely profiles it as a zigzag path. For this reason, in our data collection on different environments, the polling rates of the recorders are configured to the same value. In fact, it is not difficult to maintain a given polling rate under different running environments. Through I/O methods provided in most common programming languages, we are able to set timers and capture mouse cursor position at a fixed interval.

It is true that with an increase in user population, there is a higher chance that two users share the similar mouse movements. In fact, known as "the scalability problem", this is a common problem for almost all biometrics approaches. In face recognition, if more people are tested, it is more likely that two users' faces are similar and could make the classifier fail. The same thing happens in keystroke dynamics, and it has been determined that the accuracy of keystroke dynamics decreases with the increase in sample size [5, 23].

Though promising, our accuracy is unable to reach the European Standard for Access Control Systems, which requires a false acceptance rate (FAR) of under 0.001% and a false rejection rate (FRR) of under 1%. Therefore, we believe that our scheme is more suitable to work together with other authentication methods for user verification, instead of working as a stand-alone authentication system.

7. CONCLUSION

In this paper, we present a new approach to user reauthentication using the behavioral biometrics provided by mouse dynamics. Our approach focuses on fine-grained angle-based metrics, which have two advantages over previously studied metrics. First, angle-based metrics can distinguish a user accurately with very few mouse clicks. Second, angle-based metrics are relatively independent of the operating environment of a user, making them suitable for online reauthentication.

Our system mainly consists of a recorder, which gathers a user's mouse dynamics, and a support vector machine (SVM) classifier, which seeks to verify a user as either an impostor or an authenticated party. We gathered two sets of data: one set of 30 users under controlled circumstances, and another set of over 1,000 users on a forum website. We evaluated the system performance in terms of verification accuracy and time, resulting in a equal error rate (EER) of 1.3% with just 20 mouse clicks. We also showed that, for a system deployed at server side, the overhead required for online verification is negligible.

Acknowledgments

We would like to thank Aaron Koehl, Steven Gianvecchio, and Xin Ruan for their invaluable assistance in data collection and technical support. We would also like to thank the anonymous reviewers for their insightful comments.

8. REFERENCES

[1] A. A. E. Ahmed and I. Traore. Anomaly intrusion detection based on biometrics. In *Proceedings of the 2005 IEEE Workshop on Information Assurance*, 2005.

[2] A. A. E. Ahmed and I. Traore. A new biometric technology based on mouse dynamics. *IEEE Transactions on Dependable and Secure Computing*, 4(3):165–179, 2007.

[3] L. Ballard, F. Monrose, and D. Lopresi. Biometric authentication revisited: Understanding the impact of wolves in sheep's clothing. In *USENIX Security Symposium*, 2006.

[4] R. Biddle, S. Chiasson, and P. van Oorschot. Graphical passwords: Learning from the first twelve years. *ACM Computing Surveys*, 2011. (to appear).

[5] T. Buch, A. Cotoranu, E. Jeskey, et al. An enhanced keystroke biometric system and associated studies. In *Proceedings of Student-Faculty Research Day, CSIS, Pace University*, pages C4.2–C4.7, 2008.

[6] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[7] S. Chiasson, A. Forget, E. Stobert, P. van Oorschot, and R. Biddle. Multiple password interference in text passwords and click-based graphical passwords. In *ACM Conference on Computer and Communications Security (CCS)*, 2009.

[8] S. Chiasson, P. C. V. Oorschot, and R. Biddle. Graphical password authentication using cued click-points. In *12th European Symposium On Research In Computer Security (ESORICS), 2007*. Springer-Verlag, 2007.

[9] S. Chiasson, P. van Oorschot, and R. Biddle. A usability study and critique of two password managers. In *USENIX Security Symposium*, 2006.

[10] DTREG. SVM - Support Vector Machines. `http://www.dtreg.com/svm.htm`, Feb 2011.

[11] D. Florencio and C. Herley. A large scale study of web password habits. In *Proceedings of WWW 2007*, 2007.

[12] H. Gamboa and A. Fred. A behavioral biometric system based on human-computer interaction. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 5404, pages 381–392, Aug. 2004.

[13] P. Gupta, S. Ravi, A. Raghunathan, and N. K. Jha. Efficient fingerprint-based user authentication for embedded systems. In *Proceedings of the 42nd annual Design Automation Conference (DAC)*, pages 244–247, 2005.

[14] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proc. of European Conference on Machine Learning*, pages 137–142, 1998.

[15] Z. Jorgensen and T. Yu. On mouse dynamics as a behavioral biometric for authentication. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, pages 476–482, 2011.

[16] K. Killourhy and R. Maxion. Why did my detector do that?!: predicting keystroke-dynamics error rates. In *Proceedings of RAID'10*, pages 256–276, 2010.

[17] U. Kukreja, W. E. Stevenson, and F. E. Ritter. RUI: Recording user input from interfaces under windows and mac os x. *Behavior Research Methods*, 38(4):656–659, 2006.

[18] C. Mallauran, J.-L. Dugelay, F. Perronnin, and C. Garcia. Online face detection and user authentication. In *Proceedings of the 13th annual ACM international conference on Multimedia (MM)*, pages 219–220, 2005.

[19] F. Monrose, M. K. Reiter, and S. Wetzel. Password hardening based on keystroke dynamics. In *ACM Conference on Computer and Communications Security (CCS)*, pages 73–82, 1999.

[20] F. Monrose and A. D. Rubin. Authentication via keystroke dynamics. In *ACM Conference on Computer and Communications Security (CCS)*, pages 48–56, 1997.

[21] Y. Nakkabi, I. Traore, and A. A. E. Ahmed. Improving mouse dynamics biometric performance using variance reduction via extractors with separate features. *IEEE Transactions on Systems, Man, and Cybernetics*, 40(6):1345–1353, 2010.

[22] C. Papageorgiou, M. Oren, and T. Poggio. A general framework for object detection. In *Proceedings of the International Conference on Computer Vision*, 1998.

[23] A. Peacok, X. Ke, and M. Wilkerson. Typing patterns: A key to user identification. *IEEE Security and Privacy*, 2(5):40–47, 2004.

[24] M. Pusara and C. E. Brodley. User re-authentication via mouse movements. In *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 1–8, 2004.

[25] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. Mitchell. Stronger password authentication using browser extensions. In *USENIX Security Symposium*, 2005.

[26] D. Schulz. Mouse curve biometrics. In *Biometric Consortium Conference, 2006 Biometrics Symposium*, pages 1–6. IEEE, 2006.

[27] E. Stobert, A. Forget, S. Chiasson, et al. Exploring usability effects of increasing security in click-based graphical passwords. In *Annual Computer Security Applications Conference (ACSAC)*, 2010.

[28] S. Tong. Support vector machine active learning for image retrieval. In *Proceedings of the ninth ACM international conference on Multimedia*, 2001.

[29] V. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[30] V. N. Vladimir. *The Nature of Statistical Learning Theory*. Springer, Berline Heidelberg, New York, 1995.

[31] R. V. Yampolskiy and V. Govindaraju. Behavioural biometrics: a survey and classification. *Int. J. Biometrics*, 1(1):81–113, 2008.

[32] Y. Zhang, F. Monrose, and M. K. Reiter. The security of modern password expiration: An algorithmic framework and empirical analysis. In *ACM Conference on Computer and Communications Security (CCS)*, 2010.