The Effect of Clock Resolution on Keystroke Dynamics

Kevin Killourhy and Roy Maxion

Dependable Systems Laboratory
Carnegie Mellon University
5000 Forbes Ave,
Pittsburgh PA, 15213
{ksk,maxion}@cs.cmu.edu

Abstract. Keystroke dynamics—the analysis of individuals' distinctive typing rhythms—has been proposed as a biometric to discriminate legitimate users from impostors (whether insiders or external attackers). Anomaly detectors have reportedly performed well at this discrimination task, but there is room for improvement. Detector performance might be constrained by the widespread use of comparatively low-resolution clocks (typically 10–15 milliseconds).

This paper investigates the effect of clock resolution on detector performance. Using a high-resolution clock, we collected keystroke timestamps from 51 subjects typing 400 passwords each. We derived the timestamps that would have been generated by lower-resolution clocks. Using these data, we evaluated three types of detectors from the keystroke-dynamics literature, finding that detector performance is slightly worse at typical clock resolutions than at higher ones (e.g., a 4.2% increase in equal-error rate). None of the detectors achieved a practically useful level of performance, but we suggest opportunities for progress through additional, controlled experimentation.

Keywords: Anomaly detection; Insider-attack detection; Keystroke dynamics; Digital biometrics.

1 Introduction

Compromised passwords, shared accounts, and backdoors are exploited both by external attackers and insiders. Lists of default passwords and password-cracking programs are a staple in the toolbox of external attackers. In a study of insider attacks (i.e., those conducted by people with legitimate access to an organization), Keeney et al. [11] found that the majority of insiders exploited shared or compromised passwords, as well as backdoor accounts. However, if we had some sort of "digital fingerprint" with which to identify exactly who is logging into an account, and to discriminate between the *legitimate user* of an account and an *impostor*, we could significantly curb the threats represented by both insiders and external attackers. Of the various potential solutions to this problem, one technique that has been popular within the research community is

R. Lippmann, E. Kirda, and A. Trachtenberg (Eds.): RAID 2008, LNCS 5230, pp. 331–350, 2008.

keystroke dynamics—the analysis of individual typing rhythms for use as a biometric identifier. Compared to other biometric data, typing times are relatively easy to collect. When a user logs into a computer by typing his or her password, the program authenticating the user could save not just the characters of the password, but the time at which each key was pressed and released. One could imagine a keystroke-dynamics detection algorithm that analyzes these typing times, compares them to a known profile of the legitimate user of the account, and makes a decision about whether or not the new typist is an impostor. In fact, detectors have been designed to use typing rhythms as a biometric, not just during password entry (which is our focus in this work), but also for free-text typing [17].

In terms of accuracy, the European standard for access-control systems (EN-50133-1) specifies a false-alarm rate of less than 1%, with a miss rate of no more than 0.001% [3]. In other words, in order for a keystroke-dynamics detector to be practical, it must correctly identify a legitimate user 99% of the time, and it must correctly identify an impostor 99.999% of the time. At this point, no proposed detector has obtained such numbers in repeated evaluations. When a detector comes up short in evaluation, the common strategy is to go back to the drawing board and try a new detector. However, it may be possible to boost the performance of an existing detector by giving it better data.

Imagine the effect that timing noise might have on a detector. With enough noise, subtle differences between typists will be masked, and even a good detector will be ineffective. One obvious source of noise comes from the resolution of the clock supplying timestamps for each keystroke. For instance, our testing shows that the clock used by Microsoft Windows XP to timestamp keystroke-event messages [15] (which we call the *Windows-event clock*) has a resolution of 15.625 milliseconds (ms), corresponding to 64 updates per second. Figure 1 shows how the clock resolution affects the calculation of keydown–keydown digram latencies. Specifically, if every time reported by the clock is a multiple of 15.625 ms (truncated to the nearest millisecond), then all latencies will appear to fall in bands separated by 15 ms. The calculated latencies could differ from the true latencies by as much as the resolution of the clock (approximately ± 15 ms). If two typists differ in their typing times by less than 15 ms, then the difference could be lost. This investigation empirically measures the effect of clock resolution on the performance of a detector. Specifically, we look at whether the performance of a detector can be boosted by increasing the resolution of the clock, and whether or not detectors are robust to low-resolution clocks.

2 Background and Related Work

Detectors for discriminating between users' and impostors' keystroke dynamics have been investigated for over 30 years. They were first considered in 1977 by Forsen et al. [6], who distinguished a legitimate user from an impostor on the basis of how each one typed the user's name. In 1980, Gaines et al. [7] compared

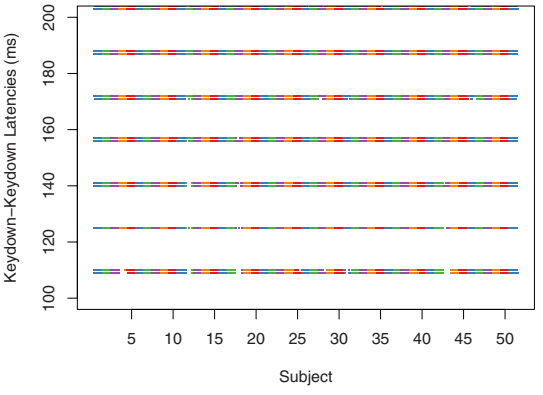


Fig. 1. The spacing between each horizontal band of keystroke latencies reveals that the Windows-event clock has a resolution of 15.625 milliseconds. Any fine-grained differences between the subjects are masked when all latencies are coarsened to a nearby multiple of the clock resolution. Data are keydown–keydown digram latencies (between 100 and 200 ms) recorded by the 15.625 ms resolution clock when each of 51 subjects typed a password 400 times. Double bands occur because Windows reports the timestamps as a whole number of milliseconds, sometimes rounding up and sometimes rounding down.

several users' keystrokes on a transcription task. Both studies presented positive findings, but cautioned that their results were only preliminary.

Joyce and Gupta [10] were some of the earliest researchers to study the keystroke dynamics of passwords. They developed a detector that compared typing characteristics of a new presentation of a password against the average (mean) typing characteristics of the legitimate user. Cho et al. [4] developed and compared two new detectors, inspired by techniques from machine learning. One was based on the nearest-neighbor algorithm, and the other used a multilayer perceptron. A full survey of keystroke-dynamics detectors has been conducted by Peacock et al. [17], but we focus here on the work of Joyce and Gupta, and Cho et al. Their work shows a diversity among available detection techniques, and our investigation uses detectors similar to theirs.

In terms of timing considerations, Forsen et al. and Gaines et al. collected data on a PDP-11. Forsen et al. reported times in 5-millisecond intervals, while Gaines et al. reported millisecond accuracy. Both Joyce and Gupta, and Cho et al. collected data on a Sun workstation. Cho et al. specified that they used X11, which provides keystroke timestamps with a 10 ms resolution. The X11 clock is typically used by researchers on UNIX-based platforms, while Windows users typically use the Windows-event clock (e.g., Sheng et al. [19]). Our testing shows that the timestamps reported through this clock have a 15.625 ms resolution (see Figure 1).

3 Problem and Approach

Keystroke-dynamics detectors—programs designed to distinguish between a legitimate user and an impostor on the basis of typing rhythms—will almost certainly be affected by the resolution of the clock that is used for timing the keystrokes. However, the extent of this effect has never been quantified or measured. In this work, we investigate the effect that clock resolution has on the performance of keystroke-dynamics detectors. We hope to boost detector performance by using better clocks, and to quantify the error introduced by typical clocks.

3.1 Investigative Approach

Our approach is outlined in the following four steps:

1. **Password-data collection:** We choose a password, and we implement a data-collection apparatus that records high-resolution timestamps. We recruit subjects to type the password. We collect keystroke timestamps simultaneously with a high-resolution clock and with a typical low-resolution clock.
2. **Derived clock resolutions:** We coarsen the high-resolution timestamps, in order to calculate the timestamps that would be generated by a range of lower-resolution clocks; we derive password-timing data at a range of resolutions.
3. **Detector implementation:** We develop three types of keystroke-dynamics detectors similar to those reported in the literature: a mean-based detector, a nearest-neighbor detector, and a multilayer perceptron.
4. **Performance-assessment method:** We construct evaluation data sets from our password-timing data, and we use them to measure the performance of the three detectors. We verify the correctness of our derivations (in step 2) by comparing a detector's performance on derived low-resolution data to its performance on data from a real clock operating at that resolution. Finally, we examine how the performance changes as a function of clock resolution.

In the end, we are able to quantify the effect that clock resolution has on several diverse detectors. We show a small but significant improvement from using high-resolution clocks. We describe the four steps of our investigation in Sections 4–7.

3.2 Controlling for Potential Confounding Factors

Our approach departs from typical keystroke-dynamics evaluations, where realism is considered to have higher importance than control. A reason for designing a controlled experiment is to remove *confounding factors*—variables that may distort the effect of the variable of interest on the experimental outcome [5].

In our investigation, the variable of interest is the clock resolution, and the experimental outcome is the performance of a detector. The clock might affect detector performance because it subtly changes the keystroke times analyzed by the detectors. All other factors that change these keystroke times are potential confounding factors that might obscure or distort this effect. They might

change a detector's performance, or even change *how clock resolution affects the detector's performance*. The presence of such a factor would compromise our investigation by offering an alternative explanation for our results.

Ideally, we would test all potential confounding factors, to see whether they actually do confound the experiment. However, to do so would require an exponential amount of data (in the number of factors). Practically, we control for potential confounding factors by keeping them constant.

4 Password-Data Collection

The first step in our investigation was to collect a sample of keystroke-timing data using a high-resolution clock. We chose a single password to use as a typing sample. Then we designed a data-collection apparatus for collecting subjects' keystrokes and timestamps. Finally, we recruited 51 subjects, and collected the timing information for 400 passwords from each one (over 8 sessions).

4.1 Choosing a Password

Password selection is the first potential confounding factor we identified. Some passwords can be typed more quickly than others. The choice of password may affect a subject's keystroke times, distorting the effect of clock resolution. To control for the potential confounding factor, we chose a single fixed but representative password to use throughout the experiment.

To make the password representative of a typical, strong password, we employed a publicly available password generator [21] and password-strength checker [13]. We generated a 10-character password containing letters, numbers, and punctuation and then modified it slightly, interchanging some punctuation and casing to better conform with the general perception of a strong password. The result of this procedure was the following password:

.tie5Roanl

The password-strength checker rates this password as strong because it contains at least 8 characters, a capital letter, a number, and punctuation. The best rating is reserved for passwords with at least 14 characters, but we decided to maintain a 10-character limit on our password so as not to exhaust our subjects' patience. (Other researchers used passwords as short as 7 characters [4].)

4.2 Data-Collection Apparatus

We wrote a Windows application that prompts a subject to type the password 50 times. Of course, in the real world, users do not type their password 50 times in a row; they might only type it a few times each day. However, the amount of practice a subject has at typing a particular password represents another potential confounding factor (see Section 3.2). Practiced typists are usually faster, and the amount of practice a subject has may affect his or her keystroke times.

By having our subjects type the password in fixed-length sessions, we controlled how much (and under what circumstances) our subjects became practiced at typing the password.

The application displays the password in a screen along with a text-entry field. In order to advance to the next screen, the subject must type the 10 characters of the password correctly in sequence and then type Return. If the subject makes a mistake, the application immediately detects the error, clears the text-entry field, and after a short pause, it prompts the subject to type the password again. For instance, if a subject typed the first three characters of the password correctly (.ti) but mistyped the fourth (w instead of e), the application would make the subject type the whole password over again. In this way, we ensure that the subject correctly types the entire password as a sequence of exactly 11 keystrokes (corresponding to the 10 characters of the password and the Return key). Forcing subjects to type the password without error is a typical constraint when analyzing keystroke dynamics [4, 19].

When a subject presses or releases a key, the application records the event (i.e., whether a key was pressed or released, and what key was involved), and also the time at which the event occurred. Two timestamps are recorded: one is the timestamp reported by the 15.625 ms resolution Windows-event clock; the other is the timestamp reported by a high-resolution external reference clock. The resolution of the reference clock was measured to be 200 microseconds by using a function generator to simulate key presses at fixed intervals. This clock reported the timestamps accurately to within ±200 microseconds. We used an external reference instead of the high-precision performance counter available through Windows [16] because of concerns that factors such as system load might decrease the accuracy of the timestamps.

The data-collection application was installed on a single laptop with no network connection and with an external keyboard. We identified keyboard selection as another potential confounding factor (see Section 3.2). If subjects used different keyboards, the difference might affect their keystroke times. We control for the potential confounding factor by using one keyboard throughout the experiment.

4.3 Running Subjects

We recruited 51 subjects, many from within the Carnegie Mellon Computer Science Department, but some from the university at large. We required that subjects wait at least 24 hours between each of their 8 sessions, so each session was recorded on a separate day (ensuring that some day-to-day variation existed within our sample). All 51 subjects remained in the study, contributing 400 passwords over the 8 sessions.

Our sample of subjects consisted of 30 males and 21 females. We had 8 left-handed and 43 right-handed subjects. We grouped ages by 10-year intervals. The median group was 31–40, the youngest group was 11–20, and the oldest group was 61–70. The subjects' sessions took between 1.25 minutes and 11 minutes, with the median session taking 3 minutes. Subjects took between 9 days and 35

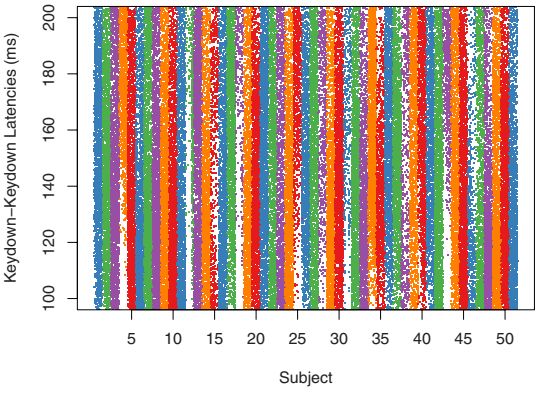


Fig. 2. The absence of horizontal bands demonstrates that the high-resolution clock has a resolution of less than 1 millisecond (200 microseconds, specifically). The keystrokes are the same as in Figure 1, but the latencies in this figure are based on the high-resolution clock.

days to complete all 8 sessions. The median length of time between the first and last session was 23 days.

5 Derived Clock Resolutions

The second step in our investigation was to use the high-resolution data to reconstruct the data that would have been collected with lower-resolution clocks. We developed a procedure to derive the timestamp of a low-resolution clock from the corresponding timestamp of a high-resolution clock.

First, we examine the keydown–keydown latencies based on the high-resolution timestamps. The latencies are shown in Figure 2. Compare these latencies to the equivalent latencies from Figure 1. Whereas the horizontal bands in Figure 1 reveal that the Windows-event clock cannot capture any timing variation smaller than 15.625 milliseconds, the absence of such bands in Figure 2 demonstrates that very subtle variations (smaller than 1 millisecond) can be captured by the high-resolution clock.

Next, to determine what would have happened if the data had been collected with a lower-resolution clock, we need to artificially decrease the resolution of this clock. Consider how timestamps are normally assigned to keystroke events:

1. The operating system is notified of the pending key event by an interrupt from the keyboard controller.
2. The operating system reads the key event from the keyboard device into memory.
3. During the handling of the key event, the operating system queries a clock for the current time.
4. The timestamp returned by the clock is included in the description of the keystroke event and is delivered to any applications waiting on the event.

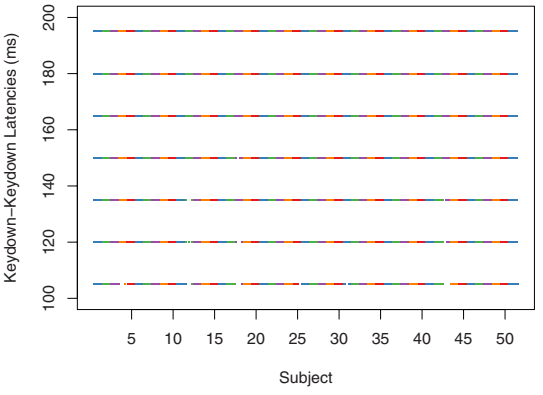


Fig. 3. The presence of horizontal bands 15 ms apart suggests that the derived 15 ms clock exhibits the same behavior as a real clock with a 15 ms resolution. The keystrokes are the same as in Figures 1 and 2, but the derived 15 ms clock was used to calculate the latencies. The bands resemble those of the real 15.625 ms clock in Figure 1, but without double bands because the 15 ms clock resolution has no fractional part being rounded to a whole millisecond.

For example, if we have a clock with a resolution of 15 ms (i.e., it is updated every 15 ms), then the timestamp returned by the clock will be divisible by 15 ms. Specifically, it will be the largest multiple of 15 ms smaller than the actual time at which the clock was queried. In general, if the clock was queried at time $t_{\text{hi-res}}$, and we want to reproduce the behavior of a lower-resolution clock (with a resolution of r), the low-resolution timestamp would be

$$t_{\text{lo-res}} \leftarrow \lfloor t_{\text{hi-res}}/r \rfloor \times r$$

where $\lfloor x \rfloor$ is the largest integer smaller than x (floor function).

Finally, with this formula and the high-resolution data, we can derive the timestamps that would have been collected with lower-resolution clocks. For instance, Figure 3 shows keystroke latencies calculated from a clock with a derived 15 ms resolution. Note the similarity to Figure 1, which shows latencies calculated from a real Windows-event clock with a 15.625 ms resolution. (The fractional part of the real clock's resolution accounts for the slight differences.)

One limitation of this procedure is that we can only derive clock resolutions that are evenly divisible by that of our high-resolution clock. This criteria allows the small but non-zero inaccuracy of our high-resolution clock to be absorbed into the inaccuracy of the lower-resolution clock. For instance, we should be able to accurately derive a 1 ms clock resolution since 1 ms is evenly divisible by 200 microseconds (the resolution of the high-resolution clock). However, we could not accurately derive a 1.5 ms clock resolution (or a 15.625 ms resolution) because it is not evenly divided. Regardless of this limitation, the accuracy of results obtained with these derived clock resolutions will be established by comparing detector performance on derived 15 ms resolution data to that on the 15.625 ms resolution Windows-event clock data. We derive data at the following 20 clock resolutions:

Milliseconds: 1 2 5 10 15 20 30 50 75 100 150 200 500 750
Seconds: 1 2 5 10 15 30

The specific resolutions were chosen arbitrarily, but with the intent of including a range of typical values (on the first line), and a range of extremely low-resolution values (on the second line) in order to identify the point at which detector performance degrades completely. In total, we have data at 22 different clock resolutions: the 20 derived clocks, the high-resolution clock, and the 15.625 ms resolution Windows-event clock.

6 Detector Implementation

The third step in our investigation was to create detectors to test using our data. We identified three different types of detector from the literature, and implemented a detector of each type:

1. a mean-based detector,
2. a nearest-neighbor detector, and
3. a multilayer-perceptron detector.

By ensuring that we have diversity in the set of detectors we evaluate, we can examine whether or not an observed effect is specific to one type of detector or more generally true for a range of detectors.

6.1 Detector Overview

We constrained our attention to detectors that behave similarly in terms of their input and output. For instance, each of our detectors must analyze password-timing data, and aims to discriminate between a legitimate user and an impostor. Each of the detectors expects the password data to be encoded in what is called a *password-timing vector*. A password-timing vector is a vector of hold times and intervals. A hold time is the difference between the key-press timestamp and the key-release timestamp for the same key. An interval time is the (signed) difference between the key-release timestamp of the first key in a digram and the key-press timestamp of the second key.

The password-timing vector is 21 elements long for the password we chose (.tie5Roanl). Each element is either a hold time for one of the 11 keys in the password (including the Return key), or the interval between one of the 10 digrams, arranged as follows:

Index	Element name
1	Hold(period)
2	Interval(period-t)
3	Hold(t)
4	Interval(t-i)
5	Hold(i)
⋮	⋮
19	Hold(l)
20	Interval(l-Return)
21	Hold(Return)

where Hold(period) is the hold time of the period key, and Interval(period-t) is the interval between the period key-release and the t key-press.

Each detector has two phases: training and testing. During training, a set of password vectors from a legitimate user is used to build a *profile* of that user. Different detectors build this profile in different ways, but the objective of a successful detector is to build a profile that uniquely distinguishes the user from all other typists (like a fingerprint). During testing, a new password-timing vector (from an unknown typist) is provided, and the detector compares the new vector against the profile. The detector produces an *anomaly score* that indicates whether the way the new password was typed is similar to the profile (low score) or different from the profile (high score). The procedure by which this score is calculated depends on the detector.

In practice, the anomaly score would be compared against some pre-determined threshold to decide whether or not to raise an alarm (i.e., whether or not the password-typing rhythms belong to an impostor). However, in our evaluation, we will use these scores directly to assess the detector's performance.

The three detectors are implemented using the R statistical programming environment (version 2.4.0) [18]. The nearest-neighbor detector leverages an implementation of Bentley's *kd*-trees [1] by Mount and Arya [14]. The multilayer perceptron uses the neural-network package AMORE [12].

6.2 Mean-Based Detector

A mean-based detector models a user's password-timing vectors as coming from some known distribution (e.g., a multidimensional normal distribution) with an unknown mean. During training, the mean is estimated, and during testing, a new password-timing vector is assigned an anomaly score based on its distance from this mean. Joyce and Gupta [10] used a detector that fits this description, and the detector we implemented is similar to theirs, but not precisely the same.[1]

During training, our mean-based detector estimates the mean vector and the covariance matrix of the training password-timing vectors. The mean vector is a 21-element vector, whose first element is the mean of the first elements of the training vectors, whose second element is the mean of the second elements of the training vectors, and so on. Similarly, the covariance matrix is the 21-by-21-element matrix containing the covariance of each pair of elements in the 21-element training vectors. These mean and covariance estimates comprise the user's profile.

During testing, the detector estimates the *Mahalanobis distance* of the new password-timing vector from the mean vector of the training data. The

[1] Our detector differs from that proposed by Joyce and Gupta in both its mean-vector calculation and the distance measure used. We calculated the mean vector using all the training data while Joyce and Gupta preprocessed the data to remove outliers. We used the Mahalanobis distance while Joyce and Gupta used the Manhattan distance. Our mean-based detector was intended to be simple (with no preprocessing) while still accommodating natural variances in the data (with the Mahalanobis distance).

Mahalanobis distance is a measure of multidimensional distance that takes into account the fact that a sample may vary more in one dimension than another, and that there may be correlations between pairs of dimensions. These variations and correlations are estimated using the correlation matrix of the training data. More formally, using the matrix notation of linear algebra, if $\mathbf{x}$ is the mean of the training data, $\mathbf{S}$ is the covariance matrix, and $\mathbf{y}$ is the new password-timing vector, the Mahalanobis distance (d) is:

$$d \leftarrow (\mathbf{x} - \mathbf{y})^{\mathsf{T}} \mathbf{S}^{-1} (\mathbf{x} - \mathbf{y})$$

The anomaly score of a new password-timing vector is simply this distance.

6.3 Nearest-Neighbor Detector

Whereas the mean-based detector makes the assumption that the distribution of a user's passwords is known, the nearest-neighbor detector makes no such assumption. Its primary assumption is that new password-timing vectors from the user will resemble one or more of those in the training data. Cho et al. [4] explored the use of a nearest-neighbor detector in their work, and we attempted to re-implement their detector for our investigation.

During training, the nearest-neighbor detector estimates the covariance matrix of the training password-timing vectors (in the same way as the mean-based detector). However, instead of estimating the mean of the training data, the nearest-neighbor detector simply saves each password-timing vector.

During testing, the nearest-neighbor detector calculates Mahalanobis distances (using the covariance matrix of the training data). However, instead of calculating the distance from the new password-timing vector to the mean of the training data, the distance is calculated from the new password-timing vector to each of the vectors in the training data. The distance from the new vector to the nearest vector from the training data (i.e., its nearest neighbor) is used as the anomaly score.

6.4 Multilayer-Perceptron Detector

Whereas the behaviors of the mean-based and nearest-neighbor detectors allow for an intuitive explanation, the multilayer perceptron is comparatively opaque. A multilayer perceptron is a kind of artificial neural network that can be trained to behave like an arbitrary function (i.e., when given inputs, its outputs will approximate the function's output). Hwang and Cho [8] showed how a multilayer perceptron could be used as an anomaly detector by training it to *auto-associate*—that is, to behave like a function that reproduces its input as the output. In theory, new input that is like the input used to train the network will also produce similar output, while input that is different from the training input will produce wildly different output. By comparing the input to the output, one can detect anomalies. Cho et al. [4] used an auto-associative multilayer perceptron to discriminate between users and impostors on the basis of password-timing vectors. We attempted to re-implement that detector.

During training, the password-timing vectors are used to create an auto-associative multilayer perceptron. This process is a standard machine-learning procedure, but it is fairly involved. We present an overview here, but we must direct a reader to the works by Hwang, Cho, and their colleagues for a comprehensive treatment [4, 8]. A skeleton of a multilayer perceptron is first created. The skeleton has 21 input nodes, corresponding to the 21 elements of the password-timing vector, and 21 output nodes. In general, a multilayer-perceptron network can have a variety of structures (called hidden nodes) between the input and the output nodes. In keeping with earlier designs, we had a single layer of 21 hidden nodes. This skeleton was trained using a technique called back-propagation to auto-associate the user's password-timing vectors. We used the recommended learning parameters: training for 500 epochs with a 1×10^{-4} learning rate and a 3×10^{-4} momentum term.[2]

During testing, the new password-timing vector is used as input to the trained multilayer perceptron, and the output is calculated. The Euclidean distance of the input to the output is computed and used as the anomaly score.

7 Performance-Assessment Method

Now that we have three detectors and data at a variety of clock resolutions, the final step is to evaluate the detectors' performance. First, we convert the data to password-timing tables. Then we devise a procedure for training and testing the detectors. Last, we aggregate the test results into overall measures of each detector's performance at each clock resolution.

7.1 Creating Password-Timing Tables

As mentioned in Section 5, we have 22 data sets that differ only in the resolution of the clock used to timestamp the keystroke events: the high-resolution clock, the 15.625 ms Windows-event clock, and the 20 derived clocks. For each clock, we have timing information for 51 subjects, each of whom typed the password (.tie5Roanl) 400 times.

We extract password-timing tables from the raw data. Hold times and digram intervals are calculated. We confirm that 50 password-timing vectors are extracted from each one of a subject's 8 sessions, and that a total of 20,400 password-timing vectors are extracted (50 passwords $\times$ 8 sessions $\times$ 51 subjects).

7.2 Training and Testing the Detectors

Consider a scenario in which a user's long-time password has been compromised by an impostor. The user is assumed to be practiced in typing her password,

[2] Note that our learning rate and momentum are 1000 times smaller than those reported by Cho et al. This change accounts for a difference in units between their password-timing vectors and ours. (We record in seconds; they used milliseconds.)

while the impostor is unfamiliar with it (e.g., typing it for the first time). We measure how well each of our three detectors is able to detect the impostor, discriminating the impostor's typing from the user's typing in this scenario.

We start by designating one of our subjects as the legitimate user, and the rest as impostors. We train and test each of the three detectors as follows:

1. We train the detector on the first 200 passwords typed by the legitimate user. The detector builds a profile of that user.
2. We test the ability of the detector to recognize the user herself by generating anomaly scores for the remaining 200 passwords typed by the user. We record these as *user scores*.
3. We test the ability of the detector to recognize impostors by generating anomaly scores for the first 5 passwords typed by each of the 50 impostors. We record these as *impostor scores*.

This process is then repeated, designating each of the other subjects as the legitimate user in turn. After training and testing a detector for each combination of subject, detector, and clock-resolution data set, we have a total of 3,366 sets of user and impostor scores (51 subjects $\times$ 3 detectors $\times$ 22 data sets).

It may seem that 200 passwords is an unrealistically large amount of training data. However, we used 200 passwords to train because we were concerned that fewer passwords might unfairly cause one or more detectors to under-perform (e.g., Cho et al. [4] trained the multilayer perceptron on up to 325 passwords). Likewise, an unpracticed impostor might seem unrealistic. If he knew that his keystroke dynamics would be scrutinized, he might practice first. However, as we argued in Section 4.2, the amount of practice a subject has had represents a potential confounding factor. Consequently, all impostors in our experiment were allowed the same level of practice. Our intuition was that the effect of clock resolution on detector performance might be seen most clearly with unpracticed impostors, and so we used their data (with plans to use practiced impostors' data in future investigations).

7.3 Calculating Detector Performance

To convert these sets of user and impostor scores into aggregate measures of detector performance, we used the scores to generate a graphical summary called an ROC curve [20], an example of which is shown in Figure 4. The hit rate is the frequency with which impostors' passwords generate an alarm (a desirable response), and the false-alarm rate is the frequency with which the legitimate user's passwords generate an alarm (an undesirable response). Whether or not a password generates an alarm depends on how the threshold for the anomaly scores is chosen. Over the continuum of possible thresholds to choose, the ROC curve illustrates how each one would change hit and false-alarm rates. Each point on the curve indicates the hit and false-alarm rates at a particular threshold.

The ROC curve is a common visualization of a detector's performance, and on the basis of the ROC curve, various cost measures can be calculated. Two common measures are the *equal-error rate* and the *zero-miss false-alarm rate*.

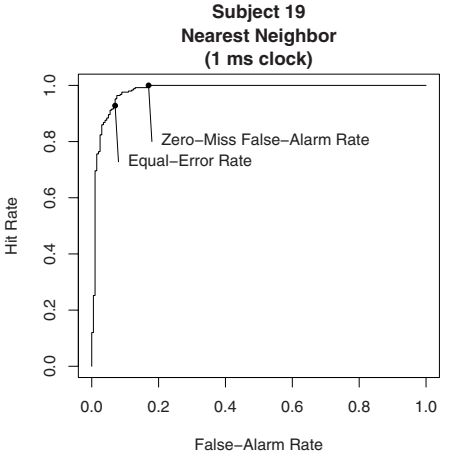


Fig. 4. An example ROC curve depicts the performance of the nearest-neighbor detector with subject 19 as the legitimate user and data from the derived 1 ms resolution clock. The curve shows the trade-off between the hit rate and false-alarm rate. The proximity of the curve to the top-left corner of the graph is a visual measure of performance.

The equal-error rate is the place on the curve where the false-alarm rate is equal to the miss rate (note that miss rate = 1 − hit rate). Geometrically, the equal-error rate is the false-alarm rate where the ROC curve intersects a line from the top-left corner of the plot to the bottom right corner. This cost measure was advocated by Peacock et al. [17] as a desirable single-number summary of detector performance. The zero-miss false-alarm rate is the smallest false-alarm rate for which the miss rate is zero (or, alternatively, the hit rate is 100%). Geometrically, the zero-miss false-alarm rate is the leftmost point on the curve where it is still flat against the top of the plot. This cost measure is used by Cho et al. [4] to compare detectors.

For each combination of subject, detector, and clock resolution, we generated an ROC curve, and we calculated these two cost measures. Then, to obtain an overall summary of a detector's performance at a particular clock resolution, we calculated the average equal-error rate and the average zero-miss false-alarm rate across all 51 subjects. These two measures of average cost were used to assess detector performance.

8 Results and Analysis

A preliminary look at the results reveals that—while the equal-error rate and the zero-miss false-alarm rate differ from one another—they show the same trends with respect to different detectors and clock resolutions. Consequently, we focus on the equal-error-rate results and acknowledge similar findings for the zero-miss false-alarm rate.

Table 1. The average equal-error rates for the three detectors are compared when using (1) the high-resolution clock, (2) the derived 15 ms resolution clock, and (3) the 15.625 ms Windows-event clock. The numbers in parentheses indicate the percent increase in the equal-error rate over that of the high-resolution timer. The results from the 15 ms derived clock very closely match the results with the actual 15.625 ms clock.

	Detectors		
Clock	Mean-based	Nearest Neighbor	Multilayer Perceptron
(1) High-resolution	0.1100	0.0996	0.1624
(2) Derived 15 ms resolution	0.1153 (+4.8%)	0.1071 (+7.5%)	0.1631 (+0.4%)
(3) 15.625 ms Windows-event	0.1152 (+4.7%)	0.1044 (+4.8%)	0.1634 (+0.6%)

The accuracy of our results depends on our derived low-resolution timestamps behaving like real low-resolution timestamps. Our first step is to establish the validity of the derived clock data by comparing a detector's performance on derived low-resolution data to its performance on data from a real clock operating at that resolution. Then we proceed to examine our primary results concerning the effect of clock resolution on detector performance.

8.1 Accuracy of the Derived Clock

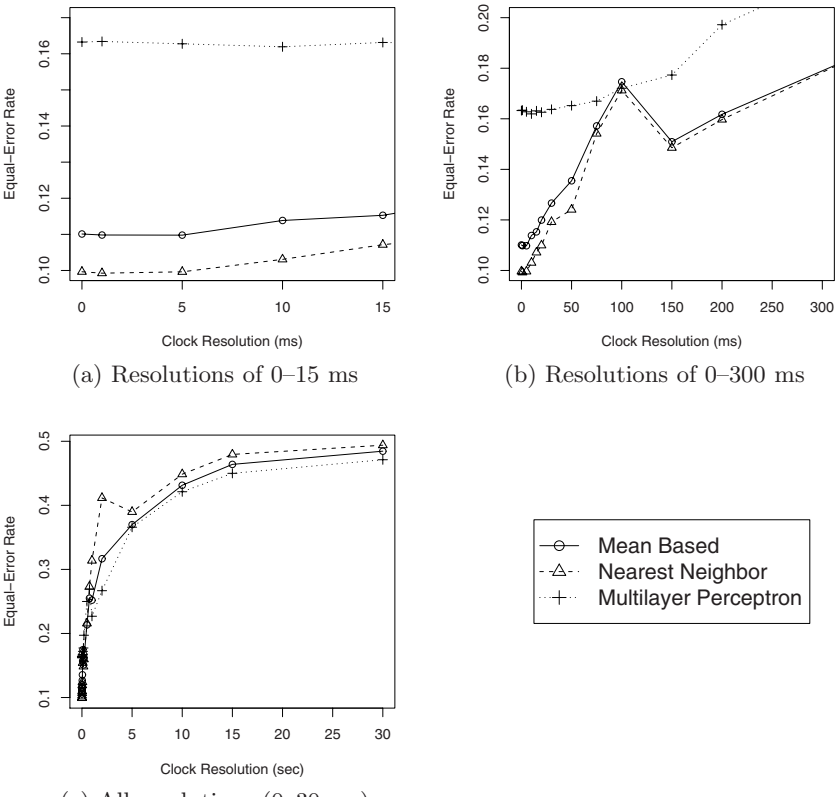
Table 1 shows the average equal-error rate for each of the three detectors, using the high-resolution clock, the derived 15 ms resolution clock, and the real 15.625 ms resolution Windows-event clock. In addition to the equal-error rates, the table includes a percentage in parentheses for the derived clock and the Windows-event clock. This percentage indicates the percent increase in the equal-error rate over that from the high-resolution clock.

To verify the correctness of the results using the derived low-resolution clocks, we compare the second and third rows of Table 1. The results are almost exactly the same except for the nearest-neighbor detector. Since the nearest-neighbor detector is not robust to small changes in the training data, it is not surprising to see a comparatively large difference between the derived 15 ms clock and the real 15.625 ms clock. The similarity in the results of the other two detectors indicate that the derived clock results are accurate.

Even if we had been able to directly derive a 15.625 ms clock (impossible because of the limitations of the derivation procedure described in Section 5), small differences between the derived and real timestamps would still cause small differences in detector performance (e.g., differences resulting from small delays in how quickly the real clock is queried).

8.2 Effects of Clock Resolution on Detector Performance

Figure 5 depicts the effect of clock resolution on the average equal-error rate of the three detectors. Each panel displays a curve for each of the three detectors, but at different scales, highlighting a different result.



(a) Resolutions of 0–15 ms

(b) Resolutions of 0–300 ms

(c) All resolutions (0–30 sec)

Fig. 5. The equal-error rates of the three detectors increase as clock-resolution goes from fine to coarse. Panel (a) depicts the minor but significant change in performance resulting from a transition from the high-resolution clock to typical 15 ms clocks. Panel (b) shows how the error jumps significantly when the clock resolution is between 50 ms and 300 ms. Panel (c) characterizes the variation in detector performance over the full range of derived clock resolutions from 1 ms to 30 seconds (where the detector does no better than randomly guessing).

Panel (a) shows the effect of clock resolutions in the range of 0–15 ms on the equal-error rate. These are resolutions that we see in practice (e.g., in Windows and X11 event timestamps). We observe some increase in the equal-error rate for the mean-based and nearest-neighbor detectors, even from the 1 ms clock to the 15 ms clock. The change from the 1 ms clock to the 15 ms clock does not seem to have much effect on the multilayer perceptron (which could be because that detector's performance is already comparatively poor, rather than because the multilayer perceptron is more robust to lower-resolution clocks). The parenthetical percentages in Table 1 quantify the change from high resolution to typical resolutions. When the detectors use the 15 ms clock, their equal-error rate

is an average of 4.2% higher than with the high-resolution clock. While this loss may not seem significant, keystroke dynamics needs near-perfect accuracy to be practical (1% false-alarm rate and 0.001% miss rate according to the European standard for access control [3]), so every possible boost in performance will help.

Panel (b) examines the effect of clock resolution beyond the 15 ms range. The graph reveals that the equal-error rates of the mean-based and nearest-neighbor detectors increase sharply after a resolution of 50 ms, and all three detectors' equal-error rates increase together after a resolution of 150 ms. While such low-resolution clocks are not used for keystroke dynamics, we can consider clock resolution to be one of many factors that might affect a detector. (Other factors include bus contention, system load, and even networking delays.) This panel suggests that these detectors are not particularly robust to noise in the form of low clock resolution. By extrapolation, it suggests that tens of milliseconds of noise from any of these sources (or any combination thereof) could be a problem.

Panel (b) also reveals a peak in the equal-error rate of the mean-based and nearest-neighbor detectors at a resolution of 100 ms. The cause of the peak is not obvious; it could be an artifact of our particular subjects' typing characteristics and would disappear with more or different subjects. More typing data and analysis would be necessary to determine whether such peaks appear consistently for a particular detector and clock resolution, but the existence of a peak does suggest that the effects of factors like clock resolution are not always easy to predict.

Panel (c) demonstrates the effect of very-low-resolution clocks on the equal-error rate of a detector. All three detectors' equal-error rates tend to 0.5, which is the theoretically worst possible equal-error rate (akin to random guessing). That the equal-error rate goes to 0.5 is not surprising, but it is surprising that the equal-error rate converges so slowly to 0.5. With a 1-second resolution, the three detectors all have equal-error rates of about 0.3. While not great, it is certainly better than randomly guessing. It is surprising that key-hold times and digram intervals retain some (weakly) discriminative information even when expressed as a whole number of seconds. It may be that the features being used to discriminate users from impostors are present only because our impostors are unpracticed; they type the password a few seconds more slowly than a practiced user would. It is possible that a curve for practiced impostors would be steeper, more quickly ascending to 0.5 (to be investigated in future work).

9 Discussion

Based on these findings, we take away two messages from this investigation, each of which suggests a trajectory for the future. First, we have demonstrated that clock resolution does have an effect on the performance of keystroke-dynamics detectors, and as a result, we should consider the potential deleterious effects of timing noise. Fortunately, the effect appears to be small for the typical clock resolutions we see in practice, but we do get a small boost in performance by using a high-resolution clock. However, clock-resolution granularity is not the only

factor that affects keystroke timestamps. Given these results, it seems almost certain that other forms of noise (e.g., system load) will cause similar problems. In the long term, we should try to eliminate noise from our timestamps, but in the short term we should at least acknowledge and account for its presence by carefully evaluating our timing mechanisms (e.g., by measuring and reporting clock resolution).

Second, even with the high-resolution timestamps, our detectors' performance is less than ideal. The best performance we obtained was a 9.96% equal-error rate for the nearest-neighbor detector, which is a long way from a 1% false-alarm rate and a 0.001% miss rate. We were surprised, since the detectors we used are similar to those that have performed well in the literature (e.g., by Joyce and Gupta [10], and by Cho et al. [4]). However, it would be improper to compare our results directly to those in the literature, because there are significant differences between our experimental method and theirs. The most obvious difference is our control of potential confounding factors (e.g., password selection and practice effect).

We speculate that experimental control is indeed responsible for the poorer performance of our detectors. Furthermore, we advocate the control of potential confounding factors in future experiments. Why? While realistic but uncontrolled experiments can demonstrate that a detector does well (or poorly), controlled experiments are necessary to reveal a causal connection between experimental factors (e.g., password choice or practice) and detector performance. If we are to use keystroke dynamics as a biometric, causal factors must be identified—*why it works* is as important as *whether it works*. For instance, it would be significant to discover that, regardless of other factors, every typist has an immutable, intrinsically identifiable quality to his or her typing. It would also be significant (but unfortunate) to find that a detector's performance depends primarily on the number of times an impostor practiced a password, and that with enough practice, any impostor could pass for a legitimate user.

We intend to conduct a survey of other detectors proposed in the literature to see whether performance remains poor on our data. We also observe that these detection algorithms tend to treat typing data as arbitrary points in a high-dimensional space, ignoring the fact that the data are observations about fingers typing. Perhaps better results can be obtained by building a detector that relies upon a model of user typing (such as those proposed by Card et al. [2] or John [9]).

10 Summary and Conclusion

The goal of this work is to investigate the effect that clock resolution has on the performance of keystroke-dynamics detectors, in part to determine if a high-resolution clock would boost performance. We collected data at a high resolution, and derived data at lower resolutions. We implemented three detectors and evaluated their performances over a range of clock resolutions. We found that a high-resolution clock does provide a slight performance boost, and conversely,

clocks with a typical 15 ms resolution increase the equal-error rate by an average of 4.2%. Based on results using very-low-resolution clocks, we found that detectors are not particularly robust to timing noise. Finally, we discovered that none of the detectors achieved a practically useful level of performance, and identified significant opportunities for progress through controlled experimentation.

Acknowledgements

The authors are grateful to Rachel Krishnaswami for her insightful comments and helpful advice, and to Patricia Loring for running the experiments that provided the data for this paper. Fahd Arshad and Rob Reeder were responsible for the instrumentation that presented stimuli to participants. Thanks also to several anonymous reviewers for their comments.

This work was supported by National Science Foundation grant numbers CNS-0430474 and CNS-0716677, and by the Army Research Office through grant number DAAD19-02-1-0389 (Perpetually Available and Secure Information Systems) to Carnegie Mellon University's CyLab. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government, or any other entity.

References

[1] Bentley, J.L.: Multidimensional binary search trees used for associative searching. Communications of the ACM 18(9), 509–517 (1975)
[2] Card, S.K., Moran, T.P., Newell, A.: The keystroke-level model for user performance time with interactive systems. Communications of the ACM 23(7), 396–410 (1980)
[3] CENELEC. European Standard EN 50133-1: Alarm systems. Access control systems for use in security applications. Part 1: System requirements, Standard Number EN 50133-1:1996/A1:2002, Technical Body CLC/TC 79, European Committee for Electrotechnical Standardization (CENELEC) (2002)
[4] Cho, S., Han, C., Han, D.H., Kim, H.-I.: Web-based keystroke dynamics identity verification using neural network. Journal of Organizational Computing and Electronic Commerce 10(4), 295–307 (2000)
[5] Dodge, Y.: Oxford Dictionary of Statistical Terms. Oxford University Press, New York (2003)
[6] Forsen, G., Nelson, M., Staron Jr., R.: Personal attributes authentication techniques. Technical Report RADC-TR-77-333, Rome Air Development Center (October 1977)
[7] Gaines, R.S., Lisowski, W., Press, S.J., Shapiro, N.: Authentication by keystroke timing: Some preliminary results. Technical Report R-2526-NSF, RAND Corporation (May 1980)
[8] Hwang, B., Cho, S.: Characteristics of auto-associative MLP as a novelty detector. In: Proceedings of the IEEE International Joint Conference on Neural Networks, Washington, DC, July 10–16, 1999, vol. 5, pp. 3086–3091 (1999)

[9] John, B.E.: TYPIST: A theory of performance in skilled typing. Human-Computer Interaction 11(4), 321–355 (1996)
[10] Joyce, R., Gupta, G.: Identity authentication based on keystroke latencies. Communications of the ACM 33(2), 168–176 (1990)
[11] Keeney, M., Kowalski, E., Cappelli, D., Moore, A., Shimeall, T., Rogers, S.: Insider threat study: Computer system sabotage in critical infrastructure sectors. Technical report, U.S. Secret Service and CERT Coordination Center/SEI (May 2005), http://www.cert.org/archive/pdf/insidercross051105.pdf
[12] Limas, M.C., Meré, J.O., Gonzáles, E.V., Martinez de Pisón Ascacibar, F.J., Espinoza, A.P., Elias, F.A.: AMORE: A MORE Flexible Neural Network Package (October 2007), http://cran.r-project.org/web/packages/AMORE/index.html
[13] Microsoft. Password checker (2008), http://www.microsoft.com/protect/yourself/password/checker.mspx
[14] Mount, D., Arya, S.: ANN: A Library for Approximate Nearest Neighbor Searching (2006), http://www.cs.umd.edu/~mount/ANN/
[15] Microsoft Developer Network. EVENTMSG structure (2008), http://msdn2.microsoft.com/en-us/library/ms644966(VS.85).aspx
[16] Microsoft Developer Network. QueryPerformanceCounter function (2008), http://msdn2.microsoft.com/en-us/library/ms644904(VS.85).aspx
[17] Peacock, A., Ke, X., Wilkerson, M.: Typing patterns: A key to user identification. IEEE Security and Privacy 2(5), 40–47 (2004)
[18] R Development Core Team. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2008)
[19] Sheng, Y., Phoha, V., Rovnyak, S.: A parallel decision tree-based method for user authentication based on keystroke patterns. IEEE Transactions on Systems, Man, and Cybernetics 35(4), 826–833 (2005)
[20] Swets, J.A., Pickett, R.M.: Evaluation of Diagnostic Systems: Methods from Signal Detection Theory. Academic Press, New York (1982)
[21] PC Tools. Security guide for windows—random password generator (2008), http://www.pctools.com/guides/password/