Comparing Anomaly-Detection Algorithms for Keystroke Dynamics

Kevin S. Killourhy
ksk@cs.cmu.edu

Roy A. Maxion
maxion@cs.cmu.edu

Dependable Systems Laboratory
Computer Science Department
Carnegie Mellon University
5000 Forbes Ave,
Pittsburgh, PA 15213

Abstract

Keystroke dynamics—the analysis of typing rhythms to discriminate among users—has been proposed for detecting impostors (i.e., both insiders and external attackers). Since many anomaly-detection algorithms have been proposed for this task, it is natural to ask which are the top performers (e.g., to identify promising research directions). Unfortunately, we cannot conduct a sound comparison of detectors using the results in the literature because evaluation conditions are inconsistent across studies.

Our objective is to collect a keystroke-dynamics data set, to develop a repeatable evaluation procedure, and to measure the performance of a range of detectors so that the results can be compared soundly. We collected data from 51 subjects typing 400 passwords each, and we implemented and evaluated 14 detectors from the keystroke-dynamics and pattern-recognition literature. The three top-performing detectors achieve equal-error rates between 9.6% and 10.2%. The results—along with the shared data and evaluation methodology—constitute a benchmark for comparing detectors and measuring progress.

1. Introduction

Compromised passwords and shared accounts are frequently exploited by both external attackers and insiders. External attackers test whether accounts use default or common passwords. Insiders compile lists of shared or compromised passwords for later use, e.g., to damage the system in the event that they are fired or demoted. If we had some means, other than knowledge of a password, with which to identify exactly who is logging into an account, and to discriminate between the *genuine user* of an account and an *impostor*, we could significantly curb these security threats. One proposed approach is the use of keystroke dynamics—the analysis of typing rhythms to discriminate among users—as a biometric identifier. With keystroke dynamics, impostor attempts to authenticate using a compromised password could be detected and rejected because their typing rhythms differ significantly from those of the genuine user.

Many anomaly-detection algorithms have been proposed for detecting impostors (to be reviewed in Section 2). It is natural to ask how well each of the detectors performs, and how different detectors compare to each other. The principle reason to evaluate and compare detectors is to assess whether a detector is sufficiently dependable to be put into practice. 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].

At present, no anomaly detector has achieved these error rates in repeated evaluation, and so keystroke dynamics could not be deployed as a sole access-control technology. Consequently, a second reason to try to assess and compare detector performance is to drive progress toward better results. By establishing which anomaly-detection strategies outperform others, the research community gains insight into what detector characteristics reduce the error rate, identifying promising directions for further study.

Unfortunately, although researchers have conducted experiments to measure the performance of various anomaly detectors, these experimental results are impossible to compare. Too many factors vary from one evaluation to another. If we are to assess the state of the art in keystroke dynamics, and to measure future progress, we need a shared benchmark data set and a repeatable procedure for conducting evaluations. Only then can the error rates of anomaly detectors be properly measured and compared against one another.

2. Background and related work

In this section, we briefly review various uses for keystroke dynamics (e.g., password vs. whole-paragraph analysis), and also various classes of analysis technique (e.g., one-class anomaly detection vs. multi-class classification). Then, we focus on a particular technique: using anomaly detectors to analyze password-typing times. Even though it should be possible to compare the performance of different anomaly detectors, we explain why a comparison based on evaluation results reported in the literature would be unsound.

978-1-4244-4421-2/09/$25.00

2.1. Review of keystroke dynamics

Since Forsen et al. [6] first investigated in 1977 whether users could be distinguished by the way they type their names, many different techniques and uses for keystroke dynamics have been proposed. Peacock et al. [17] conducted an extensive survey of the keystroke-dynamics literature. Not all of that research is relevant to the use considered in this work, namely, analyzing password-typing times. For instance, Gaines et al. [7] considered whether typists could be identified by analyzing keystroke times as they transcribed long passages of text. Techniques for analyzing passages are different from those for analyzing passwords, and the same evaluation data cannot always be used to assess both.

Further, even among studies of password-typing times, not all of the extant techniques can be evaluated using the same procedure. One class of techniques is anomaly detection. The typing samples of a single, genuine user are used to build (or *train*) a model of the user's typing behavior. When a new typing sample is presented, the detector *tests* the sample's similarity to the model, and outputs an *anomaly score*. In contrast, another class of techniques is multi-class classification. The typing samples of multiple users are used to find decision boundaries that can be used to distinguish each user from the others. Since anomaly detectors train on a single user's data, while multi-class classifiers train on multiple users' data, these two techniques require different evaluation procedures.

2.2. Anomaly detectors for password timing

Table 1 presents a concise summary of seven studies from the literature that use anomaly detection to analyze password-timing data. Each study described one or more anomaly detectors, gathered password-typing data, conducted an evaluation, and reported the results. The key observation from this table is that, despite the surface similarities, the studies contain substantial differences beyond just the anomaly detectors. These differences make it impossible to compare anomaly-detector performance from one study to another.

The table has been split into two sections for readability. The first column in each section provides a reference to the source study. The remaining columns provide the following information:

Detector: a descriptive name for the anomaly-detection algorithm used in the study (sometimes using different terminology than the authors to clarify the type of data analysis that underlies the detection strategy).

Feature Sets: features used to train and test the detectors (Enter: the Enter key is considered to be part of the password; Keydown-Keydown: time between the key presses of consecutive keys is used as a feature; Keyup-Keydown: time between the release of one key and the press of the next is used; Hold: time between the press and release of each key is used).

Password Length: number of characters used in the passwords (N/A indicates the length was not available in the source study).

Password Reps: number of password-typing repetitions used to train the detector.

Filtering Users: users whose typing times were highly variable or inconsistent were identified during data collection and excluded from the study.

Filtering Times: the collected timing data were processed with an outlier-handling procedure to remove extreme values.

Testing #Attempts: number of attempts that users were given to try to authenticate successfully (e.g., a 2 means a user who was rejected the first time would be given a second chance to type the password).

Testing Updating: the typing-model was updated during testing to accommodate typing behavior that changes over time.

Results Threshold: name of the procedure for choosing a threshold on the detectors' anomaly scores to obtain miss and false-alarm rates (heuristic: threshold score was chosen with a heuristic described in the study; zero-miss: threshold was chosen to obtain a miss rate of zero; equal-error: threshold was chosen to obtain equal miss and false-alarm rates).

Results Miss/False Alarm: reported miss rates (percentage of impostor passwords that are not detected) and false-alarm rates (percentage of genuine user passwords that are mistakenly detected as impostors): (a) denotes that the detectors were combined into an aggregate detector, and only the results for the aggregate were reported; (b) indicates that only results for two-attempt authentication were reported.

To illustrate the problem we encounter when we try to use the literature to determine which detector has the best performance, suppose we tried to compare two detectors: the Neural Network (auto-assoc) detector developed by Cho et al. [4], and the Outlier-Count (z-score) detector designed by Haider et al. [8]. The neural net has a reported miss rate of 0.0% and a false-alarm rate of 1.0%. The outlier-counting detector has a reported miss rate of 13% and a false-alarm rate of 2%. Since the neural net has a better miss rate and false-alarm rate, one might be inclined to conclude that it is better than the outlier-counting detector.

However, that conclusion is unsound because the table reveals many differences between the studies used to evaluate the two detectors. The neural net (1) trained on a different set of timing features; (2) had more repetitions in the training data; (3) did not have to deal with users whose typing was inconsistent, or with timing outliers in the training

	Source Study	Detector	Feature Sets: Enter Key	Feature Sets: Keydown-Keydown	Feature Sets: Keyup-Keydown	Feature Sets: Hold	Password: Length	Password: Reps
1	Joyce & Gupta (1990) [10]	Manhattan (filtered)	✓	✓			N/A	8
2	Bleha et al. (1990) [2]	Euclidean (normed)		✓			11–17	30
		Mahalanobis (normed)		✓			11–17	30
3	Cho et al. (2000) [4]	Nearest Neighbor (Mahalanobis)	✓		✓	✓	7	75–325
		Neural Network (auto-assoc)	✓		✓	✓	7	75–325
4	Haider et al. (2000) [8]	Fuzzy Logic		✓			7	15
		Neural Network (standard)		✓			7	15
		Outlier Count (z-score)		✓			7	15
5	Yu & Cho (2003) [21]	SVM (one-class)	✓		✓	✓	6–10	75–325
6	Araujo et al. (2004) [1]	Manhattan (scaled)		✓	✓	✓	10+	10
7	Kang et al. (2007) [11]	k-Means			✓	✓	7–10	10

	Source Study	Filtering: Users	Filtering: Times	Testing: #Attempts	Testing: Updating	Results (%): Threshold	Results (%): Miss	Results (%): False Alarm
1	Joyce & Gupta (1990) [10]		✓	1		heuristic	0.25	16.36
2	Bleha et al. (1990) [2]			1	✓	heuristic	2.8	8.1[(a)]
				1	✓	heuristic	2.8	8.1
3	Cho et al. (2000) [4]	✓	✓	1		zero-miss	0.0	19.5
		✓	✓	1		zero-miss	0.0	1.0
4	Haider et al. (2000) [8]			2		heuristic	19.	11.[(b)]
				2		heuristic	22.	20.
				2		heuristic	13.	2.
5	Yu & Cho (2003) [21]	N/A	N/A	1		zero-miss	0.0	15.78
6	Araujo et al. (2004) [1]			1	✓	heuristic	1.89	1.45
7	Kang et al. (2007) [11]	N/A	N/A	1	✓	equal-error	3.8	3.8

Table 1. Seven different studies investigate 11 different anomaly detectors and report evaluation results. The diversity of evaluation conditions makes a direct comparison of the anomaly-detector performance results impossible. Section 2.2 details the contents of each column of the table.

data; (4) was given only one attempt (not two) to correctly verify the user; and (5) was assessed using a different type of threshold on the anomaly score. Any of these five factors might explain why the neural net has lower error rates than the outlier-counting detector. If these factors were all controlled, we might discover that the outlier-counting detector outperforms the neural net (as reported later in this work; see Table 2).

3. Problem and approach

As illustrated in the previous section, it is unsound to compare anomaly detectors using the evaluation results reported in the literature. Too many factors differ from one evaluation to another to make sensible conclusions about the relative performance of two detectors.

Our objective in this paper is to collect a keystroke-dynamics data set, to develop an evaluation procedure, and to measure the performance of a range of anomaly-detection algorithms so that the results can be compared on an equal basis. In the process, we establish which detectors have the lowest error rates on these data, and we provide a data set and evaluation methods that can be shared throughout the research community to evaluate new detectors and assess progress. Our approach is as follows:

1. Password-data collection: We collected typing data from 51 subjects, each typing 400 repetitions of a password. The various timing features used by researchers (e.g., the keydown-keydown times and hold times) were extracted from the raw data.

2. Detector implementation: Fourteen anomaly detectors from the literature were reimplemented: eleven had been proposed by keystroke-dynamics researchers, and three were "classic" anomaly detectors from the pattern-recognition literature.

3. Evaluation methodology: We developed a procedure to evaluate each detector, on the same data, under the same conditions, so that performances can be compared. The error rates of the detectors were calculated and tabulated.

In Sections 4–6, we describe the three steps of our methodology in more detail.

4. Password-data collection

The first step in our evaluation was to collect a sample of keystroke-timing data. In this section, we explain how

we chose a password to use as a typing sample, designed a data-collection apparatus, recruited subjects to type the password, and extracted a set of password-timing features.

4.1. Choosing a password

Choosing passwords for a keystroke-dynamics evaluation is tricky. On one hand, it is often more realistic to let users choose their own passwords. On the other hand, data collection becomes more difficult since different impostor samples would be needed for each password. Some researchers have suggested that letting users choose their own passwords makes it easier to discriminate them [1]. If true, then letting users choose their own passwords would bias the results of an experiment designed to evaluate performance on an arbitrary password. We decided that the same password would be typed by all of our subjects.

To make a password that is representative of typical, strong passwords, we employed a publicly available password generator [16] 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 more than 7 characters, a capital letter, a number, and punctuation. The top rating is reserved for passwords longer than 13 characters, but according to the studies that were presented in Table 1, 10 characters is typical. Those studies that used longer strings often used names and English phrases that are easier to type.

4.2. Data-collection apparatus

We set up a laptop with an external keyboard to collect data, and developed a Windows application that prompts a subject to type the password. The application displays the password in a screen 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 press Enter. If any errors in the sequence are detected, the subject is prompted to retype the password. The subject must type the password correctly 50 times to complete a data-collection session. Whenever the subject presses or releases a key, the application records the event (i.e., keydown or keyup), the name of the key involved, and what time the event occurred. An external reference clock was used to generate highly accurate timestamps. The reference clock was demonstrated to have an accuracy of ± 200 microseconds (by using a function generator to simulate key presses at fixed intervals).

Of course, subjects would not naturally type their password 50 times in a row, and they would type it on their own computers, not our keyboard. We chose to sacrifice some amount of realism so we could use this carefully-controlled data-collection apparatus. We had two reasons for this decision. First, we wanted to ensure the accuracy of the timestamps (as described above). Second, we wanted to make the environment as consistent as possible for all subjects. If some subjects typed the password more frequently than others, or if different subjects used different keyboards, these differences could make it artificially easier for an anomaly detector to distinguish between typists.

4.3. Running subjects

We recruited 51 subjects from within the university. Subjects completed 8 data-collection sessions (of 50 passwords each), for a total of 400 password-typing samples. They waited at least one day between sessions, to capture some of the day-to-day variation of each subject's typing.

Our set of subjects consisted of 30 males and 21 females. We had 8 left-handed and 43 right-handed subjects. The median age group was 31–40, the youngest was 18–20 and the oldest was 61–70. The subjects' sessions took between 1.25 and 11 minutes, with the median session taking about 3 minutes.

4.4. Extracting timing vectors

The raw typing data (e.g., key events and timestamps) cannot be used directly by an anomaly detector. Instead, sets of timing features are extracted from the raw data. These features are typically organized into a vector of times called a *timing vector*. Different researchers extract different combinations of features (as shown in the Feature Sets columns of Table 1).

We decided to use all the features used across all of the studies shown in Table 1. Specifically, we considered the Enter key to be part of the password (effectively making the 10-character password 11 keystrokes long), and we extracted keydown-keydown times, keyup-keydown times, and hold times for all keys in the password. For each password, 31 timing features were extracted and organized into a vector. The times are stored in seconds (as floating-point numbers).

Many of the timing features are correlated and some are linearly dependent (i.e., each keydown-keydown time can be decomposed into the sum of a hold time and a keyup-keydown time). We did not transform the data to remove these correlations despite their adverse effect on some detectors because they are typical of keystroke timing data. Other researchers have investigated strategies to compensate through feature selection [21]. By retaining all the features, we expect our timing data could be useful in a future evaluation of such work.

5. Detector implementation

The second step in our evaluation was to implement 14 anomaly-detection algorithms that analyze password-

timing data. We tried to faithfully reimplement the eleven algorithms listed in the Detector column of Table 1. We also implemented three "classic" anomaly-detection methods (i.e., the Euclidean, Manhattan, and Mahalanobis distance metrics) from the pattern-recognition literature.

All of these detectors have been described in comprehensive detail elsewhere, and due to space constraints we cannot include the same detail. However, we provide a concise explanation (and a citation to a more detailed description) for each detector. We note ambiguities which could inadvertently cause our reimplementation to differ from the original. Occasionally, we must resort to terminology that is specific to a statistical or machine-learning technique (e.g., the vocabulary of neural networks); references are provided.

The 14 detectors were implemented using the R statistical programming environment (version 2.6.2) [18]. Each detector has a training phase where a set of timing vectors from the genuine user is used to build a model of the user's typing behavior, and a test phase where each new test vector is assigned an anomaly score.

Because some detectors have parameters that affect their performance, the question of parameter tuning arises. Since there is no commonly accepted method for tuning the parameters of anomaly detectors for a data set without introducing bias in the evaluation results [12], we used the parameters specified by the source study when possible, and explained what parameters we used.

5.1. Euclidean

This classic anomaly-detection algorithm [5] models each password as a point in p-dimensional space, where p is the number of features in the timing vectors. It treats the training data as a cloud of points, and computes the anomaly score of the test vector based on its proximity to the center of this cloud. Specifically, in the training phase, the mean vector of the set of timing vectors is calculated. In the test phase, the anomaly score is calculated as the squared Euclidean distance between the test vector and the mean vector.

5.2. Euclidean (normed)

This detector was described by Bleha et al. [2] who called it the "normalized minimum distance classifier." In the training phase, the mean vector is calculated as in the standard Euclidean detector. In the test phase, the squared Euclidean distance between the test vector and the mean vector is calculated, but the anomaly score is calculated by "normalizing" this distance, dividing it by the product of the norms of the two vectors (i.e., if $\mathbf{x}$ is the mean vector, $\mathbf{y}$ is the test vector, and d is the squared Euclidean distance, then the anomaly score is $d/(\|\mathbf{x}\|\,\|\mathbf{y}\|)$).

5.3. Manhattan

This classic anomaly-detection algorithm [5] resembles the Euclidean detector except that the distance measure is not Euclidean distance, but Manhattan (or city-block) distance. In the training phase, the mean vector of the timing vectors is calculated. In the test phase, the anomaly score is calculated as the Manhattan distance between the mean vector and the test vector.

5.4. Manhattan (filtered)

This detector was described by Joyce and Gupta [10]. It is similar to the Manhattan detector except outliers are filtered from the training data. In the training phase, the mean vector of the timing vectors is calculated, and the standard deviation for each feature is calculated also. Any timing-vector element that is more than three standard-deviations above the mean is removed, and a more robust mean vector is computed without these extreme values. In the test phase, the anomaly score is calculated as the Manhattan distance between this robust mean vector and the test vector.

5.5. Manhattan (scaled)

This detector was described by Araújo et al. [1]. In the training phase, the mean vector of the timing vectors is calculated, and the mean absolute deviation of each feature is calculated as well. In the test phase, the calculation is similar to the Manhattan distance, but with a small change. The anomaly score is calculated as $\sum_{i=1}^{p} |x_i - y_i| / a_i$ where x_i and y_i are the i-th features of the test and mean vectors respectively, and a_i is the average absolute deviation from the training phase. The score resembles a Manhattan-distance calculation, except each dimension is scaled by a_i.

5.6. Mahalanobis

This classic anomaly-detection algorithm [5] resembles the Euclidean and Manhattan detectors but the distance measure is more complex. Mahalanobis distance can be viewed as an extension of Euclidean distance to account for correlations between features. In the training phase, both the mean vector and the covariance matrix of the timing vectors are calculated. In the test phase, the anomaly score is calculated as the Mahalanobis distance between the mean vector and the test vector (i.e., if $\mathbf{x}$ is the mean vector, $\mathbf{y}$ is the test vector, and $\mathbf{S}$ is the covariance matrix, the Mahalanobis distance is $(\mathbf{x}-\mathbf{y})^{\mathsf{T}}\mathbf{S}^{-1}(\mathbf{x}-\mathbf{y})$).

5.7. Mahalanobis (normed)

This detector was described by Bleha et al. [2] who called it the "normalized Bayes classifier." In the training phase, the mean vector and covariance matrix of the training vectors are calculated. In the test phase, the Mahalanobis distance between the mean vector and test vector is calculated. The anomaly score is calculated by "normalizing" the Mahalanobis distance using the same divisor as the Euclidean (normed) detector.

5.8. Nearest-neighbor (Mahalanobis)

This detector was described by Cho et al. [4]. In the training phase, the detector saves the list of training vectors, and calculates the covariance matrix. In the test phase, the detector calculates the Mahalanobis distance between each of the training vectors and the test vector. The anomaly score is calculated as the distance from the test vector to the nearest training vector.

5.9. Neural-network (standard)

This detector was described by Haider et al. [8]. It incorporates a feed-forward neural-network trained with the back-propagation algorithm [5]. In the training phase, a network is built with p input nodes, one output node, and $\lceil 2p/3 \rceil$ hidden nodes. The original detector was designed specifically with 6 input nodes and 4 hidden nodes because the researchers only considered 6-feature timing vectors. Our vectors are 31-features long, so we extended their structure to include 31 input nodes while maintaining a ratio of 2 hidden nodes for every 3 input nodes.

The network weights were all initialized to 0.1, and the bias term of each node was initialized randomly, as described in the source study. The detector was trained to produce a 1.0 on the output node for every training vector. We trained for 500 epochs using a learning-rate parameter of 0.0001. During testing, the test vector was run through the network, and the output of the network was recorded. If s denotes the output of the network, the anomaly score was calculated as $1 - s$ since, intuitively, if s is close to 1.0, the test vector is similar to the training vectors, and with s close to 0.0, it is dissimilar.

5.10. Neural-network (auto-assoc)

This detector was described by Cho et al. [4], who called it an "auto-associative, multilayer perceptron." It also incorporates a feed-forward neural-network trained with the back-propagation algorithm, but unlike a typical neural network, the structure of the network is designed for use in an anomaly detector [9]. Intuitively, the training phase teaches the network to produce output vectors close to the inputs for the training vectors (hence the "auto-associative" descriptor). Then, during the test phase, input vectors that produce dissimilar outputs are assigned high anomaly scores.

In the training phase, the neural network is constructed with p input nodes and p output nodes (where p is the number of timing-vector features). We used p hidden nodes as well (as described by one of the sources [4]). The network is trained to reproduce each input timing vector as the output. We trained for 500 epochs using a learning-rate of 0.0001 and a momentum parameter of 0.0003 (equivalent to those reported by Cho and his colleagues since our times are in seconds not milliseconds). In the test phase, the p-feature test vector is run through the network, producing a p-feature output vector. The Euclidean distance between the test vector and the output vector is calculated and used as the anomaly score.

5.11. Fuzzy-logic

This detector was described by Haider et al. [8]. It incorporates a fuzzy-logic inference procedure. The key idea is that ranges of typing times are assigned to fuzzy sets (e.g., the times in the range of 210–290 milliseconds are part of a set named "very fast"). The sets are called fuzzy because elements can partially belong to a set (e.g., the time 255 is strongly in the "very fast" set while 290 is only weakly a member of the set).

In the training phase, the detector determines how strongly each feature belongs to each set, and each feature is matched with the set in which its membership is strongest (e.g. the t-hold-time feature will be matched with the "very fast" set if most t hold times are around 255 milliseconds). In the test phase, each timing feature is checked to see if it belongs to the same set as the training data (e.g., the test vector's t hold time is checked for membership in the "very fast" set). The anomaly score is calculated as the average lack of membership across all test vector timing features. Note that we added sets (e.g., "very very fast") to accommodate faster times than seen in the source study.

5.12. Outlier-counting (z-score)

This detector was described by Haider et al. [8], who called it the "statistical technique." In the training phase, the detector calculates the mean and standard deviation of each timing feature. In the test phase, the detector computes the absolute z-score of each feature of the test vector. The z-score for the i-th feature is calculated as $|x_i - y_i| / s_i$, where x_i and y_i are the i-th features of the test and mean vectors respectively and s_i is the standard deviation from the training phase. The anomaly score is a count of how many z-scores exceed a threshold (which was unspecified, so we used 1.96, typical for outlier detection in this setting).

5.13. SVM (one-class)

This detector was described by Yu and Cho [21]. It incorporates an algorithm called a support-vector machine (SVM), that projects two classes of data into a high-dimensional space and finds a linear separator between the two classes. A "one-class" SVM variant was developed for anomaly detection. It projects the data from a single class and finds a separator between the projection and the origin.

In the training phase, the detector builds a one-class SVM using the training vectors. In the test phase, the test vector is projected into the same high-dimensional space and the (signed) distance from the linear separator is calculated. The anomaly score is calculated as this distance, with the sign inverted, so that positive scores are separated from the data. The SVM parameter ν was set to 0.5; the source study set ν with a "heuristic search" but did not elaborate.

5.14. k-means

This detector was described by Kang et al. [11]. It uses the k-means clustering algorithm to identify clusters in the training vectors, and then calculates whether the test vector is close to any of the clusters. In the training phase, the detector simply runs the k-means algorithm on the training data (with $k = 3$). The algorithm produces three centroids such that each training vector should be close to at least one of the three centroids. In the test phase, the anomaly score is calculated as the Euclidean distance between the test vector and the nearest of these centroids.

6. Evaluation methodology

The final step was to evaluate each of the 14 detectors using the password-timing data. Each detector was trained and tested using the same procedure, and the anomaly scores output by each detector were converted into standard measures of error.

6.1. 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 their password, while the impostor is unfamiliar with it (e.g., typing it for the first time). We measure how well each of our detectors is able to discriminate between the impostor's typing and the genuine user's typing in this scenario.

We start by designating one of our 51 subjects as the genuine user, and the rest as impostors. We train an anomaly detector and test its ability to recognize the genuine user and impostors as follows:

1. We run the training phase of the detector on the timing vectors from the first 200 password repetitions typed by the the genuine user. The detector builds a model of the user's typing behavior.
2. Then, we run the test phase of the detector on the timing vectors from the remaining 200 repetitions typed by the genuine user. We record the anomaly scores assigned to each timing vector as *user scores*.
3. Finally, we run the test phase of the detector on the timing vectors from the first five repetitions typed by each of the 50 impostors. We record the anomaly scores assigned to each timing vector as *impostor scores*.

This process is then repeated, designating each of the other subjects as the genuine user in turn. After training and testing each of the 14 detectors, we have a total of 741 sets of user and impostor scores (51 subjects $\times$ 14 detectors).

It may seem that 200 repetitions is an unrealistically large amount of training data. We were concerned that fewer passwords might unfairly cause one or more detectors to under-perform (e.g., Table 1 shows detectors trained with up to 325 passwords). Likewise, an unpracticed impostor might seem unrealistic, since impostors might practice if they knew timing mattered. We believe that our choices are fair for a first evaluation; experiments involving fewer training repetitions and more practiced impostors are planned.

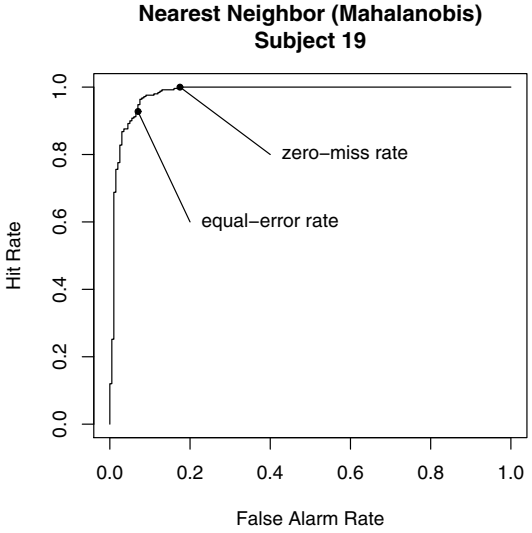

Figure 1. An example ROC curve depicts the performance of the Nearest Neighbor (Mahalanobis) detector with subject 19 as the genuine user. The curve shows the trade-off between the hit rate and the false-alarm rate. Proximity to the top-left corner of the graph is a visual measure of performance.

6.2. Calculating detector performance

To measure detector performance, we used the scores to generate a graphical summary of performance called an ROC curve [20]. An example ROC curve is shown in Figure 1. The hit rate is the frequency with which impostors are detected (i.e., $1 -$ miss rate), and the false-alarm rate is the frequency with which genuine users are mistakenly detected as impostors. Whether or not a password generates an alarm depends on how the threshold on the anomaly scores is chosen. The choice of threshold establishes the operating point of the detector on the ROC curve. Over the continuum of possible thresholds, the ROC curve illustrates the hit and false-alarm rates that would be attained at each possible detector operating point.

The ROC curve is a common visualization of a detector's accuracy, and on the basis of the ROC curve, various measures of error can be derived. Table 1 lists several studies that have chosen a threshold using detector-specific heuristics. As the ROC curve shows, the nature of these heuristics could have a major effect on the reported miss and false-alarm rates. Further, if different heuristics are used for different detectors, comparing detector performance becomes difficult.

Two measures are commonly used for determining a

	Detector	equal-error rate	
1	**Manhattan (scaled)**	**0.096**	**(0.069)**
2	**Nearest Neighbor (Mahalanobis)**	**0.100**	**(0.064)**
3	**Outlier Count (z-score)**	**0.102**	**(0.077)**
4	SVM (one-class)	0.102	(0.065)
5	Mahalanobis	0.110	(0.065)
6	Mahalanobis (normed)	0.110	(0.065)
7	Manhattan (filter)	0.136	(0.083)
8	Manhattan	0.153	(0.092)
9	Neural Network (auto-assoc)	0.161	(0.080)
10	Euclidean	0.171	(0.095)
11	Euclidean (normed)	0.215	(0.119)
12	Fuzzy Logic	0.221	(0.105)
13	k Means	0.372	(0.139)
14	Neural Network (standard)	0.828	(0.148)

	Detector	zero-miss false-alarm rate	
1	**Nearest Neighbor (Mahalanobis)**	**0.468**	**(0.272)**
2	**Mahalanobis**	**0.482**	**(0.273)**
3	**Mahalanobis (normed)**	**0.482**	**(0.273)**
4	**SVM (one-class)**	**0.504**	**(0.316)**
5	Manhattan (scaled)	0.601	(0.337)
6	Manhattan (filter)	0.757	(0.282)
7	Outlier Count (z-score)	0.782	(0.306)
8	Manhattan	0.843	(0.242)
9	Neural Network (auto-assoc)	0.859	(0.220)
10	Euclidean	0.875	(0.200)
11	Euclidean (normed)	0.911	(0.148)
12	Fuzzy Logic	0.935	(0.108)
13	k Means	0.989	(0.040)
14	Neural Network (standard)	1.000	(0.000)

Table 2. The average equal-error rates (left side) and average zero-miss false-alarm rates (right side) from the evaluation of the 14 detectors are ranked from best to worst (with standard deviations in parentheses). The set of top-performing detectors is indicated in bold-face (i.e., those that are not significantly worse than the best-performing detector).

threshold in a detector-independent way and summarizing performance: *equal-error rate* and *zero-miss false-alarm rate*. To calculate the equal-error rate, the threshold is chosen so that the detector's miss and false-alarm rates are equal. This measure was used by Kang et al. [11]. To calculate the zero-miss false-alarm rate, the threshold is chosen so that the false-alarm rate is minimized under the constraint that the miss rate be zero. This measure was used in two earlier studies [4, 21]. The equal-error rate and the zero-miss false-alarm rate are different performance measures, but both are error rates (i.e., lower values imply fewer errors and better performance). For each of the 714 combinations of detector and subject, we generated an ROC curve, and calculated these two measures.

7. Results and analysis

Table 2 shows the average equal-error and zero-miss false-alarm rates for each of the 14 detectors over all subjects. The detectors have been rank-ordered from best performance to worst (with separate rankings for each of the two types of performance measure).

Since the anomaly detectors were evaluated using the same data, under the same conditions, using the same procedures, it is possible to attribute differences in performance to the anomaly detector and not to different experimental conditions. We will establish which detectors are the top performers on this data set using a statistical analysis, and then proceed with a detailed comparison of the detectors.

7.1. Finding the top-performing detectors

When comparing the error rates (equal-error or zero-miss false-alarm) of the 14 anomaly detectors, we naturally ask which detector performs best. One detector will have the minimum error rate on the test data, and the other 13 detectors' error rates will be higher (assuming no ties). However, a test of *statistical significance* is required to establish that the difference between the best-performing detector and another detector is real, and not explained by random effects.

We conduct 13 hypothesis tests, comparing the performance of the best-performing detector to each of the other 13 detectors. Formally, each test's null hypothesis is that the best-performing detector is no better than the other detector (i.e., a one-sided test). If we cannot reject the null hypothesis for a detector, we accept that the detector's performance is competitive with the best-performing detector, and consider it to be a *top-performing* detector.

When using hypothesis testing to compare anomaly-detector performance, Cho et al. [4] used a paired t-test. The t-test assumes that the difference between the two error rates is normally distributed. We plotted the distributions of the error rates of each detector in our evaluation, and found that they were far from normal (e.g., some looked multimodal). Consequently, we opted to use the non-parametric Wilcoxon signed-rank test instead [19], which has been shown to be more robust when testing distributions that are non-normal. We used a significance level of $\alpha = 0.05$, and since we conducted 13 hypothesis tests, we applied a Bonferroni correction for multiple testing [14].

7.2. Detector performance comparison

The best equal-error rate was 0.096, obtained by the Manhattan (scaled) detector described by Araújo et al. [1]. The Nearest Neighbor (Mahalanobis) detector, and the Outlier Count (z-score) detector were the other top-performing detectors using the equal-error performance measure.

The best zero-miss false-alarm rate was 0.468, obtained by the Nearest Neighbor (Mahalanobis) detector described

by Cho et al. [4]. The classical Mahalanobis detector, the Mahalanobis (normed) detector, and the SVM (one-class) detector were the other top-performing detectors using the zero-miss false-alarm performance measure.

Our initial observation is that none of the performance measures comes close to what is needed to achieve the 0.001% miss rate and 1% false-alarm rate required by the European standard for access-control systems [3]. Since our first reason for wanting to evaluate and compare anomaly detectors was to determine if a detector was sufficiently reliable to be put into practice, we can conclude that further progress is needed before we could rely solely on keystroke dynamics for access control.

Since our second reason for comparing detectors is to discover promising anomaly-detection strategies that might drive progress, we look for shared strategies among our top-performing detectors (according to either performance measure). These detectors all employ some form of scaling of the timing features (e.g., by employing the Mahalanobis rather than Euclidean distance). It has been observed that different timing features have different variability (e.g., hold times tend to be faster and more consistent than keydown-keydown times which are slower and more variable). We find that detectors which account for these differences in scale demonstrate better performance.

The Nearest Neighbor (Mahalanobis) detector is the only top-performing detector according to both performance measures. Since the equal-error rate and the zero-miss false-alarm rate measure the performance of the detector at different operating points on an ROC curve (see Figure 1), this result suggests that the Nearest Neighbor (Mahalanobis) detector is comparatively robust to different threshold-selection procedures.

Our final observation is that the seven worst-ranked detectors (ranks 8–14) are the same for both performance measures, and equivalently, the seven detectors with the best rank are the same. Even though the ordering of the top seven is not the same for both measures, this observation suggests a clear division between seven detectors that are competitive in this evaluation and seven that are not.

8. Discussion and future work

This study is the first time a diverse group of 14 of the anomaly detectors for keystroke dynamics have been evaluated on an equal basis. While our results are only for a single data set and evaluation procedure, they demonstrate the value of shared data and a consistent evaluation methodology. Additionally, our work highlights the need for more shared data and resources.

8.1. Shared data and methods

The comparison of these particular 14 anomaly detectors is interesting in that it reveals which detection strategies seem to work, and which strategies fare poorly. However, in terms of usefulness to other keystroke-dynamics researchers, the comparison itself may be less important than that the comparison was made possible. We invite researchers to use our data and methods to evaluate other detectors and other implementations of these detectors. The benefit of using these data and methods is a valid comparison with the results in this work.[1]

To our knowledge, the only other researchers to make keystroke data publicly available through the Web are Montalvão et al. [15]. From discussion with colleagues, we understand that other data sets are forthcoming, and we welcome this trend since the availability of a diverse set of public data will boost progress in this field.

8.2. Extensions to the evaluation method

We foresee uses for the data beyond comparing anomaly detectors using our evaluation procedure. As explained throughout this report, we had to make various tradeoffs (e.g., about what features to include in our timing vectors, and how many password vectors to use in training) that undoubtedly had an effect on detector performance. Consequently, the data could also be used to evaluate what effect these decisions have.

For instance, we chose to use 200 samples to train the detector, which may seem excessive to some; and we used unpracticed impostors, which may boost detector performance. As explained in Section 6.1, we made these decisions for the sake of a fair evaluation. If researchers are concerned about high performance with less training data, a different evaluation procedure could be developed to train detectors using fewer passwords. We simply ask that extensions be explicitly and carefully described, so that different evaluation methodologies are not conflated and confused.

8.3. Detector variability across data sets

A key issue is that minor differences—in the detectors, the data, and the evaluation—can obviously cause major swings in detector performance. Keystroke-dynamics is a sensitive instrument in a noisy domain. Since assessment and comparison depends on controlling these small differences in performance, shared data and common evaluation procedures are critical.

For example, as shown in Table 1, Cho et al. [4] compared a Nearest Neighbor (Mahalanobis) detector to a Neural Network (auto-assoc) detector. They reported that the zero-miss false-alarm rate was 19.5% for the nearest-neighbor detector and 1.0%, and for the neural net. In our evaluation, the zero-miss false-alarm rates were 10.0% and

[1] The keystroke data set has been linked from each of the authors' individual webpages and is available directly from
`http://www.cs.cmu.edu/~keystroke`
The data are accompanied by a comprehensive description of the collection procedure and detector-evaluation methodology we used.

46.8% respectively. The error rates for both detectors are higher in our evaluation. Further, the neural net outperforms the nearest-neighbor detector in their evaluation while the opposite is true in ours. Additional shared data and further evaluations are needed to identify and tease apart the factors that boost and hinder each detector's performance.

9. Summary and conclusion

Previously, it was not possible to compare the performance of different anomaly detectors across studies in the keystroke dynamics literature. Our objective in this work has been to collect a data set, develop an evaluation procedure, and measure the performance of many anomaly-detection algorithms on an equal basis. In the process, we established which detectors have the lowest error rates on our data (e.g., the Nearest Neighbor (Mahalanobis) detector), and we provide a data set and evaluation methodology that can be used by the community to assess new detectors and report comparative results.

Acknowledgments

The authors are grateful to Patricia Loring for running the experiments that provided the data for this paper, and to Rachel Krishnaswami for her insightful comments and helpful advice. Thanks also to several anonymous reviewers for their helpful 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.

References

[1] L. C. F. Araújo, L. H. R. Sucupira, M. G. Lizárraga, L. L. Ling, and J. B. T. Yabu-uti. User authentication through typing biometrics features. In *Proceedings of the 1st International Conference on Biometric Authentication (ICBA)*, volume 3071 of *Lecture Notes in Computer Science*, pages 694–700. Springer-Verlag, Berlin, 2004.

[2] S. Bleha, C. Slivinsky, and B. Hussien. Computer-access security systems using keystroke dynamics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(12):1217–1222, 1990.

[3] CENELEC. European Standard EN 50133-1: Alarm systems. Access control systems for use in security applications. Part 1: System requirements, 2002. Standard Number EN 50133-1:1996/A1:2002, Technical Body CLC/TC 79, European Committee for Electrotechnical Standardization (CENELEC).

[4] S. Cho, C. Han, D. H. Han, and H. Kim. Web-based keystroke dynamics identity verification using neural network. *Journal of Organizational Computing and Electronic Commerce*, 10(4):295–307, 2000.

[5] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley & Sons, Inc., second edition, 2001.

[6] G. Forsen, M. Nelson, and R. Staron, Jr. Personal attributes authentication techniques. Technical Report RADC-TR-77-333, Rome Air Development Center, October 1977.

[7] R. S. Gaines, W. Lisowski, S. J. Press, and N. Shapiro. Authentication by keystroke timing: Some preliminary results. Technical Report R-2526-NSF, RAND Corporation, May 1980.

[8] S. Haider, A. Abbas, and A. K. Zaidi. A multi-technique approach for user identification through keystroke dynamics. *IEEE International Conference on Systems, Man and Cybernetics*, pages 1336–1341, 2000.

[9] B. Hwang and S. Cho. Characteristics of auto-associative MLP as a novelty detector. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, volume 5, pages 3086–3091, 10–16 July, 1999, Washington, DC, 1999.

[10] R. Joyce and G. Gupta. Identity authentication based on keystroke latencies. *Communications of the ACM*, 33(2):168–176, 1990.

[11] P. Kang, S. Hwang, and S. Cho. Continual retraining of keystroke dynamics based authenticator. In *Proceedings of the 2nd International Conference on Biometrics (ICB'07)*, pages 1203–1211. Springer-Verlag Berlin Heidelberg, 2007.

[12] H.-j. Lee and S. Cho. Retraining a keystroke dynamics-based authenticator with impostor patterns. *Computers & Security*, 26(4):300–310, 2007.

[13] Microsoft. Password checker, 2008. `http://www.microsoft.com/protect/yourself/password/checker.mspx`.

[14] R. G. Miller, Jr. *Simultaneous Statistical Inference*. Springer-Verlag, New York, second edition, 1981.

[15] J. Montalvão, C. A. S. Almeida, and E. O. Freire. Equalization of keystroke timing histograms for improved identification performance. In *2006 International Telecommunications Symposium*, pages 560–565, September 3–6, 2006, Fortaleza, Brazil, 2006.

[16] PC Tools. Security guide for windows—random password generator, 2008. `http://www.pctools.com/guides/password/`.

[17] A. Peacock, X. Ke, and M. Wilkerson. 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] D. J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & Hall/CRC, fourth edition, 2007.

[20] J. A. Swets and R. M. Pickett. *Evaluation of Diagnostic Systems: Methods from Signal Detection Theory*. Academic Press, New York, 1982.

[21] E. Yu and S. Cho. GA-SVM wrapper approach for feature subset selection in keystroke dynamics identity verification. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 2253–2257. IEEE Press, 2003.