Keystroke-Dynamics Authentication Against Synthetic Forgeries

Deian Stefan, *Member, IEEE,* and Danfeng (Daphne) Yao, *Member, IEEE*

Abstract—We describe the use of keystroke-dynamics patterns for authentication and detecting infected hosts, and evaluate its robustness against forgery attacks. Specifically, we present a remote authentication framework called TUBA for monitoring a user's typing patterns. We evaluate the robustness of TUBA through comprehensive experimental evaluation including two series of simulated bots. Support vector machine is used for classification. Our results based on 20 users' keystroke data are reported. Our work shows that keystroke dynamics is robust against synthetic forgery attacks studied, where attacker draws statistical samples from a pool of available keystroke datasets other than the target. TUBA is particularly suitable for detecting extrusion in organizations and protecting the integrity of hosts in collaborative environments, as well as authentication.

Index Terms—Keystroke dynamics, authentication, malware detection, forgery.

1 Introduction

Keystroke-dynamics based authentication is a cheap biometric mechanism that has been proven accurate in distinguishing individuals [2], [7], [9], [10], [17], [21]. Most of the attack models considered in keystroke-dynamics literature assume the attackers are humans, e.g., a colleague of Alice trying to log in as Alice. However, there has not been much study on the robustness of this technique against synthetic and automatic attacks and forgeries. For example, an attacker may write a program that performs statistic manipulation and synthesis to produce keystroke sequences in order to spoof others. These types of forgery attacks pose a serious threat. It is unclear from the current literature how robust keystroke dynamics is against forgery attacks. We address this gap in our work.

We present a design and implementation of a remote authentication framework called TUBA for monitoring a user's typing patterns; and evaluate the robustness of TUBA through comprehensive experimental evaluation including two series of simulated bots.

- *D. Stefan is with Department of Electrical Engineering, The Cooper Union, New York, NY 10003. Email: stefan@cooper.edu.*
- *D. Yao is with Department of Computer Science, Virginia Tech, Blacksburg, VA 24060. Email: danfeng@cs.vt.edu.*

This work has been supported in part by Rutgers DIMACS REU programs, National Science Foundation grants CNS-0831186 and CAREER CNS-0953638.

We also describe the use of keystroke dynamics as a tool to help identify anomalous activities on a personal computer, e.g., activities that may be due to malware. We consider a model where a user's computer in an organization or enterprise may be infected with malicious software that may stealthily launch attacks. This model is motivated by the increasing number of infected hosts caused by organized malicious botnets.

Our contributions are summarized as follows.

1) We design and implement a simple and easy-to-adopt protocol for authenticating a computer owner that utilizes the user's keyboard activities as an authentication metric. We present our protocol in a lightweight client-server architecture using the X Windows System (X or X11 for short).
2) We analyze keystroke data from a group of users on a diverse set of inputs, including email addresses, a password, and Web addresses. We find that performance results vary according to the strings used for authentication. We find that different types of strings give different performance when used for authentication.
3) We evaluate the robustness of keystroke-dynamics based authentication against automated bot attacks. We implement two bot programs, called *GaussianBot* and *NoiseBot*, respectively, which are capable of injecting statistically-generated keystroke event sequences on a (victim) machine. The bot programs aim to pass our keystroke authentication tests by mimicking a particular user's keystroke dynamics. Our prototype and evaluation results on the accuracy and robustness of TUBA demonstrate the feasibility of utilizing human keystroke-dynamics as behavior features in host-based malware detection.
   The bots are capable of launching a series of intelligent attacks drawn upon the statistical analysis of collected keystroke data. Experiments show that our classification is robust against several types of attacks, and is able to correctly classify the attacks by GaussianBot and NoiseBot with low false positive rates.

TUBA is particularly suitable for detecting extrusion

in enterprises and organizations, protecting the integrity and security of hosts in collaborative environments, and as an authentication method. Our work also suggests that certain human behaviors, namely user inputs, can be leveraged for malware detection. We give concrete examples detailing how to prevent malware forgery in such human-behavior driven security systems. This study is the result of an on-going effort towards designing human-inspired security solutions. The techniques and results presented in this paper serve as fundamental building blocks for constructing advanced and sophisticated host-based malware detection tools.

Organization of the Paper: We describe a remote authentication framework and our security model in Section 2, where a use case of using TUBA to detect anomalous network activities is also described. Details of our implementation including data collection, keystroke logging, feature extraction, and classification can be found in Section 3. Two bots capable of injecting synthetic keystroke events are presented in Section 4. Our experimental evaluation results and user study are described in Section 5. Related work is described in Section 6. In Section 7, we conclude the paper and describe plans for future work.

2 Overview and Security Model

TUBA (Telling Human and Bot Apart) is a remote biometric authentication system based on keystroke-dynamics information. We use machine-learning techniques to detect intruders merely based on keystroke dynamics, i.e., timing information of keyboard events. We allow for certain types of key event injection by bots.

Security model and malware attack model We assume that the host operating system and kernel-level data—including our client-side keystroke-collection modules and cryptographic keys used—are secure and not compromised. The remote server for issuing keystroke challenge and data analysis is trusted and secure. Client-side malware may run as a user-level application – type I malware (malicious software) according to the stealthy malware taxonomy in [15], e.g., spyware implemented as Firefox extensions. Malware is active in making outside connections for command & control or attacks. We allow malware to inject arbitrary keystroke events and sequences, e.g., synthesized sequences, *except* the ones belonging to the owner of the computer. Thus, under this malware-attack model we assume that keylogging by spyware or by human attackers [22] on the owner's computer is infeasible. An attacker may also carry out conventional network attacks such as eavesdropping on the communication channel between client and server, or replaying network packets.

We note that with hardware TPM (Trusted Platform Module) enabled, fake key events can be detected and removed with reasonable overhead (e.g., [5], [18]). In comparison, we consider a relaxed environment where TPM is not enabled or available – referred to, by us, as a non-TPM environment. Our security assumption on the kernel integrity can be relaxed through the use of TPM attestation [16] or virtualization based introspection [12], which are not considered in this paper.

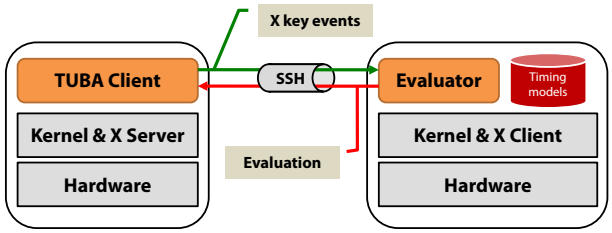


Fig. 1. TUBA architecture in a client-server model.

We introduce definitions used in our model. We refer to an individual who has legitimate access to the computer as the *owner*. Without loss of generality, we assume that a computer has one owner, as our solutions can be easily generalized to a multi-owner setting. Our TUBA framework can be realized with a *stand-alone program* on the client's local machine. The program is responsible for collecting training keystroke data, building learning models, analyzing, and classifying TUBA challenges. This type of stand-alone architecture is easy to deploy and implement. It is, however, required that the user ensure that the program is running and that proper measures are taken if TUBA issues warnings or alerts.

The use of keystroke-dynamic authentication requires initial training, after which TUBA challenges may be issued.

Training Phase: The remote authentication server collects keystroke data from a legitimate user. We assume that the user's computer is not infected during the training phase, but may be infected and recruited into a botnet after the training phase has ended. The training phase is as follows. The user and the remote server authenticate each other and set up a secure connection. The user then types M strings s_i, $i = 1, \ldots, M$, as specified by the server, n times each. The authentication server records the keystroke data from the user, which is possible using the X Window System. The user runs X server with a XTrap extension, which intercepts the user's keystroke events and sends the information to the application on the remote authentication server. Once a sufficient number of samples have been collected, the authentication server processes the user's keystroke data by training a support vector machine, the details of which are presented in Section 3.1.

TUBA challenge: When a suspicious network event is observed, TUBA prompts the user with a window requesting him/her to type in a server-chosen string, s_i. Based on this user's keystroke timing data and the classification model built during the training phrase, TUBA decides whether the user is the legitimate owner or not. The suspicious events mentioned above may be triggered by existing bot detection solutions, such as BotHunter [4], BINDER [3], or according to other

(simple) pre-defined policies.

Use cases A TUBA authentication test can be triggered periodically or when one or more suspicious events are observed. Our TUBA authentication model can also run in a non-intrusive mode where the user's keystroke timing is analyzed without explicitly prompting an authentication window for the user to type into. We define an *event* as a set of network and/or input activities (keyboard or mouse). Suspicious events are activities that are pre-defined and related to malicious bot activities, such as sending a large number of email messages (potential spam) or making a large number of HTTP requests to a single target host (potential DoS attacks). A suspicious event can be related to external inputs, such as the computer sending email (i.e., SMTP traffic) without any prior keyboard or mouse activities. Some additional examples of trigger events that can be used to start a TUBA challenge including: HTTP requests without a browser process which can be identified using `lsof` and `netstat`, certain user-initiated network activities such as sending email without keyboard/mouse input or with a screensaver active, listening sockets on suspicious ports, sending high-volume traffic to a single target host, attempting to disable the bot detection program, etc. Furthermore, trigger events may be issued based on network traffic patterns or content. For example, if the owner's computer is sending email with spam-like characteristics, is periodically visiting a server in a foreign country with no hostname or other records (possible HTTP-based C&C channel), or has an unusual chat application establishing connections (possible IRC-based C&C channel) a TUBA challenge is issued.

Next, we describe the technical details of our TUBA framework, including feature extraction, classification, and comprehensive evaluation.

3 Implementation Details

The architecture of TUBA in client-server model is illustrated in Figure 1. We describe the use of X server for key forwarding from the client to the trusted server next.

In TUBA, a trusted remote server is responsible for data collection and analysis in a remote fashion, e.g., using SSH (Secure Shell) the client would remotely login to the server with X11-forwarding enabled so that the keystroke events can be monitored by the server. The connection to and storage on the remote server is assumed to be secure. Various key-logging methods for the GNU/Linux operating system exist. However, most of the currently-available keyloggers were *not* designed with the intention to extract timing information from a user's typing pattern, and require superuser privileges to be installed or used. Addressing these issues and the need for a platform-independent utility, we implemented a keylogger for the *X Windows System* using the XTrap extension. The X Windows System is a powerful graphical user interface composed of the *X server* and *X clients*. The X server runs on the machine where the keyboard, mouse and screen are attached, while X clients are common applications (e.g., *Firefox*, *KPDF* or *XTerm*) that run on either the local machine or a remote machine, due to the inherent network capabilities of X11.

The X server can be extended with modules, such as the XTrap server extension used in our event collection. One of the capabilities of the XTrap extension is to intercepts the core input (keyboard and mouse) events and forward them to XTrap client applications. As such, our keylogger (client application) contains a callback function that is executed whenever a `KeyPress` or `KeyRelease` event occurs to record the event information. Some supplementary data, such as the current location of the mouse pointer and the name of the current window in focus, are obtained and formatted to be easily parsed by the feature extractor.

Figure 2 is an example of the partial output of the keylogger when typing the word "bot".

The communication channel between the client and server can be secured using SSL or other encryption protocols, which effectively prevent eavesdropping and packet tampering attacks.

The key events are parsed by the feature extractor, which contains a small buffer of the last C `KeyPress` and `KeyRelease` events. Given a database of words ($s_i,\ i = 1, \ldots, M$) to monitor and feature descriptions (i.e., keystroke durations, total time to type a word, press-to-press times, etc.) of how the strings were typed, when the buffer contents of the keyboard input matches a database word, the features are extracted and again formatted to be easily parsed by the classifier. Size C is adjusted to match the largest word in the database.

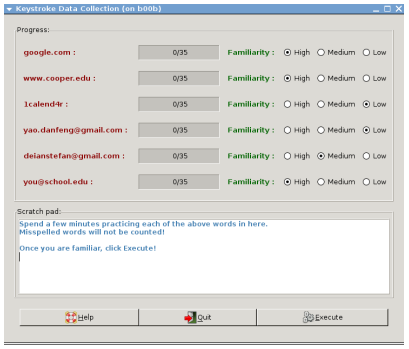
3.1 Feature Extraction and Classification

Given a sequence of key-press and key-release events, features represent various temporal aspects of the user's typing patterns. Features may include the total typing time of the word and various inter-key timings such as the interval between two adjacent press or release events. Even for a short string such as the URL `www.amazon.com`, the dimensionality of all possible features is quite high. The TUBA classification algorithm uses principle component analysis (PCA) to reduce the dimensions of the feature vectors as a preprocessing step. PCA is an existing data mining and statistical technique which is commonly used to condense high-dimensional data to lower dimensions in order to simplify analysis. The premise of PCA is to reduce the dimensions of and transform the original multi-dimensional datasets so that high variations within the data are retained, i.e., the *principal components* are retained.

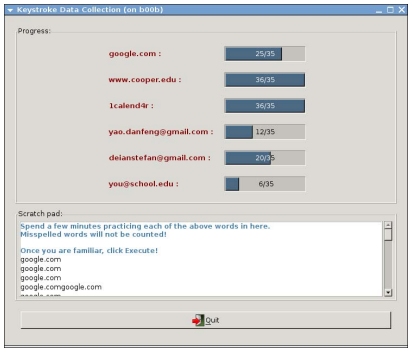
A key observation in feature extraction is that humans are imperfect typists and are keen to creating negative timing features in a sequence of keystroke events. For example, when typing the string "abc", a user may create negative press-to-release (PR) time by pressing 'c' before having released 'b'. More formally, if we denote the state

```
1  Window=xterm:XTerm|Event=KeyPress|...|char=b|screen=0|rootXY=(1236,370)|root=0|state=0|time=86474468
2  Window=xterm:XTerm|Event=KeyRelease|...|char=b|screen=0|rootXY=(1236,370)|root=0|state=0|time=86474562
3  Window=xterm:XTerm|Event=KeyPress|...|char=o|screen=0|rootXY=(1236,370)|root=0|state=0|time=86474626
4  Window=xterm:XTerm|Event=KeyPress|...|char=t|screen=0|rootXY=(1236,370)|root=0|state=0|time=86474683
5  Window=xterm:XTerm|Event=KeyRelease|...|char=o|screen=0|rootXY=(1236,370)|root=0|state=0|time=86474692
6  Window=xterm:XTerm|Event=KeyRelease|...|char=t|screen=0|rootXY=(1236,370)|root=0|state=0|time=86474785
```

Fig. 2. Examples of logged key events when typing “bot”.



(a)



(b)

Fig. 3. Screenshot of the data collection GUI (a) before and (b) during recording.

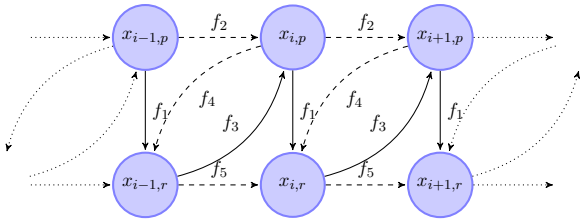


Fig. 4. Comparisons between the typing abilities of a person and a bot modeled by using a first-order Markov chain. $x_{i,p}$ and $x_{i,r}$ denote the i-th letter pressed and released, respectively. Linear combinations of the f_k elements represent timing features.

at $i-1$ as x_{i-1} = ‘b’, and that at i as x_i = ‘c’, given that ‘c’ is pressed before ‘b’ is released then $\mathrm{PR}(x_{i-1}, x_i) = x_{i,p} - x_{i-1,r} < 0$. From our experimental data, we find that a large number of users have negative press-to-release timings in their datasets. Although an adversary can synthesize arbitrary keystroke events, we find that it considerably more difficult to create an intelligent bot which can inject keystroke events that result in negative inter-key timings (See also Section 4).

Figure 4 illustrates the practical differences in the capabilities between human and bots. Assuming that keystroke events can be modeled accurately by a first-order Markov chain, a human’s key event path would be a combination of the dashed and solid lines shown in the figure. It is, however, difficult for a bot to simulate certain events, as is the case of negative timing features (paths including dashed lines in Figure 4). When considering higher-order Markov chains, it is even more challenging for the attackers to successfully mimic typing patterns with negative timing; a person may, for example, press ‘c’ before both ‘a’ and ‘b’ are released. Using such high-dimensional data leads to higher authentication accuracy and stronger security guarantees. However, increasing the complexity of the model (e.g., to a second- or third-order Markov chain) should be accompanied by an increase in training instances as to avoid overfitting (the model to) the data.

Once keystroke features are collected and processed, we train and classify the data using support vector machines (SVMs). The use of SVMs is appropriate as the technique can be used to classify both linearly-separable (i.e., classes which are separable into two or more groups using hyperplanes) and non-linearly separable data [6], [8], [13]. To classify a set of data points in a linear model, support vector machines select a small number of critical boundary points from each class, which are called the *support vectors* of the class. Then, a linear function is built based on the support vectors in order to separate the classes as much as possible; a *maximum margin* hyperplane, i.e., a high-dimensional generalization of a plane, is used to separate the different classes. An SVM model can classify non-linear data by transforming the feature vectors into a high-dimensional feature space using a kernel function (e.g., polynomial, sigmoid or radial basis function (RBF)) and then performing the maximum-margin separation. As a result, the separating function is able to produce more complex boundaries and therefore yield better classification performance. In our authentication system, we use the WEKA [20] SVM implementation with a Gaussian RBF kernel. We refer readers to data mining and machine learning literature such as the book by Witten and Frank [20] or Bishop [1] for detailed descriptions of SVM techniques.

4 Bot Simulation and Events Injection

We find that even if we allow for certain types of key event injection by bots under our security model,

classification based on keystroke dynamics is able to identify intruders with high accuracy. We play the devil's advocates and create two series of bots, the algorithms of which are described next. We assume that the goal of an adversary in our model is to create keystroke events that pass our classification tests. That is, the attacker attempts to create fake keystroke events expecting them to be falsely classified as the owner's. Under our adversary model, we assume that bots possess keystroke data of some users except the owner's, i.e., replaying the owner's keystroke sequence is prohibited as described at the beginning of Section 2.

We implement a program in C which injects keyboard events with specific timing information in order to simulate forgeries. Our attack simulator has two components: the *data synthesizer* and *typing event injection*. To simulate an (intelligent) bot's attack, we write a program to create fake keyboard events and inject them into the X server core-event-stream (using the XTrap extension) as if typed on the actual keyboard. From the application's (or X client's) perspective, the fake keyboard events cannot be distinguished from actual key events (even though the keyboard is not touched). To test the performance of a bot injecting fake events we implemented two bots which simulate human typing patterns according to the *first-order Markov model* shown in Figure 4. That is, bots consider only keystroke durations and positive inter-key timings (paths shown by the solid lines in Figure 4).

In our simulations, the keystroke duration of the ith character in a word is modeled as a random variable $X_i \geq 0$, where X_i is either

1) Gaussian with mean μ_i and variance σ_i^2: $X_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$, or
2) constant with additive uniform noise (mean 0): $X_i \sim \mu_i + \mathcal{U}(-\eta_i, \eta_i)$,

depending on the type of bot desired, GaussianBot or NoiseBot. The parameter μ_i is calculated as the mean key duration of the i-th character from selected instances of the user study. For example, to calculate μ_1 for the first character ('1') in the string `1calend4r`, we take the `1calend4r` instances from the user study and calculate the sample mean and variance of the keystroke durations for the character '1'. Similarly, the press-release inter-key timing feature between the i-th and $(i-1)$-th character was modeled as a random variable X_i', whose parameters are also calculated from the user study instances. Algorithm 1 below shows the pseudocode for the *GaussianBot*, which injects n instances of the given string. Similarly, the pseudocode for *NoiseBot* which generates noisy instances (i.e., mean $\pm$ noise) is shown in Algorithm 2. The classification performance of these bots against users are further presented later.

It is important to note that a more complex bot would additionally consider negative inter-key timing and therefore a high-order Markov Model may be implemented. This advanced bot would require considerably greater effort from the bot designer, as the order of events would have to be calculated a priori. For example, if the bot were to correctly simulate the word "botnet" typed by a person, the probability of injecting a `KeyPress` event for the character 'o' before injecting a `KeyRelease` event of 'b' would have to be considered and therefore our bot Algorithms would need to be modified significantly.

Algorithm 1: *GaussianBot* simulation of a human

input: string=$\{x_1, x_2, \ldots, x_N\}$,
durations=$\{(\mu_1, \sigma_1), (\mu_2, \sigma_2), \ldots, (\mu_N, \sigma_N)\}$,
inter-key
timing=$\{(\mu_2', \sigma_2'), (\mu_3', \sigma_3'), \ldots, (\mu_N', \sigma_N')\}$,
n=number of words to generate

for $n \leftarrow 1$ **to** n **do**
  for $i \leftarrow 1$ **to** N **do**
    SimulateXEvent(`KeyPress`, x_i);
    $X_i \leftarrow \mathcal{N}(\mu_i, \sigma_i^2)$; `/* key duration */`
    if $X_i < 0$ **then** $X_i \leftarrow 0$; `/* adjust for large variance */`
    Sleep(X_i);
    SimulateXEvent(`KeyRelease`, x_i);
    $X_i' \leftarrow \mathcal{N}(\mu_i', \sigma_i')$; `/* inter-key timing */`
    if $X_i' < 0$ **then** $X_i' \leftarrow 0$;
    Sleep(X_i');

Algorithm 2: *NoiseBot* simulation of a human

input: string=$\{x_1, x_2, \ldots, x_N\}$,
durations=$\{(\mu_1, \eta_1), (\mu_2, \eta_2), \ldots, (\mu_N, \eta_N)\}$,
inter-key
timing=$\{(\mu_2', \eta_2'), (\mu_3', \eta_3'), \ldots, (\mu_N', \eta_N')\}$,
n=number of words to generate

for $n \leftarrow 1$ **to** n **do**
  for $i \leftarrow 1$ **to** N **do**
    SimulateXEvent(`KeyPress`, x_i);
    $X_i \leftarrow \mu_i + \mathcal{U}(-\eta_i, \eta_i)$; `/* key duration */`
    if $X_i < 0$ **then** $X_i \leftarrow 0$; `/* adjust for large noise */`
    Sleep(X_i);
    SimulateXEvent(`KeyRelease`, x_i);
    $X_i' \leftarrow \mu_i' + \mathcal{U}(-\eta_i', \eta_i')$; `/* inter-key timing */`
    if $X_i' < 0$ **then** $X_i' \leftarrow 0$; `/* adjust negative timing */`
    Sleep(X_i');

5 Evaluation of Classification Accuracy

We collect keystroke timing data from 20 user subjects, 10 females and 10 males on $M = 5$ different strings. We

implement a program with a graphic user interface (GUI) that records the keystroke dynamics of the participants. Screen shots of the GUI are shown in Figure 3. The user is asked to type in the following strings, $n = 35$ times each: `google.com`, `www.amazon.com`, `1calend4r`, `yao.danfeng@gmail.com`, and `deianstefan@gmail.com`. The gender and age of each participant are recorded, as well as their familiarity ('high', 'medium', or 'low') with each string. This data is later used for analyzing the correlation between demographic data and keystroke dynamics. Before the recording begins, each user has a chance to practice typing each string up to five times each. The study is carried out one user at a time in a controlled environment where the user can concentrate and focus on what he or she is typing. Experimental variables, such as the keyboard, monitor and computer are also kept constant.

We perform three sets of experiments to test the feasibility and the performance of TUBA in classifying keystroke timing features. We illustrate the setup of the experiments in Table 1.

The goal of Experiment 1 is to confirm our ability to distinguish different individuals' keystroke patterns with good prediction results, as has been shown in the existing literature. We are able to achieve high accuracy in classifying individual humans.

5.1 Experiment 1 (Human vs. Human)

Among the 20 users, we set up a basic SVM test to see if our classification algorithm can distinguish each user from the others. Three different classification sets $c_i,\ i = 1, 2, 3$ for each word were created according to the users' gender: $c_1 = \{\text{all male instances}\}$, $c_2 = \{\text{all female instances}\}$, and $c_3 = c_1 \cup c_2$. The class i experimental setup of word s_l for user u_j was then performed as follows:

- Label each of the user's 35 instances as `owner`,
- Pick 5 random instances for every user $u_k \neq u_j$ whose instances are in the set $\{c_i\}$ and label them as `unknown`,
- Given the relabeled instances, perform a 10-fold cross-validation for SVM classification (manually adjusting the model parameters).
- Calculate the average TP and FP rates.

The classification analysis was repeated for all the user subjects, words in the database, and classification sets. Finally, the average TP and FP rates for every word and class were calculated, the results of which are summarized in Table 2 – the average false positive rate of 4.2% confirms the robustness of using keystroke dynamics for authentication.

In general, the performance across the different classes had little effect on the performance of the SVM classifier. We note, however, that the familiarity and length *do* affect the results. From Table 2 we observer that less familiar strings such as `1calend4r`, have a lower true positive rate than the more familiar strings, like `www.amazon.com`. This is because the users were still not very comfortable with the string and the variance (which in this case may effectively be considered *noise*) in the feature vectors is quite high.

On average, the true positive and false positive rates of the longer strings (`yao.danfeng@gmail.com` and `deianstefan@gmail.com`) perform better because the users have an additional "freedom" to demonstrate their unique typing style. Since the strings are very long, some users, for example, pause (unconsciously) mid-word and this is reflected in some of the inter-key timing measurements.

5.2 Experiments 2 & 3 (Human vs. Bots)

Existing literature on keystroke authentication does not provide any analysis of attacks that are based on statistical and synthetic keystroke timing; to our knowledge, there are currently no bots which are able to perform the attacks that we consider. Therefore, we design two sets of experiments to simulate some sophisticated bot attacks.

We evaluate the robustness of keystroke analysis against artificially and statistically created sequences of events. As auxiliary information for the attacker, we give the adversary access to the keystroke data of all 19 users excluding the owner's data. Results from Experiment 2 and 3 are presented below.

In the bot experiments, only 10 user cases and $M = 3$ strings are used, with extended focus on tweaking the model parameters. The chosen strings ($s_j,\ j = 1, \ldots, M$) included a URL (`www.amazon.com`), an email address (`deianstefan@gmail.com`) and a password (`1calend4r`). Similar to the results of Experiment 1, gender classes only affect the results minimally, and therefore only the class containing both genders was considered for Experiments 2 and 3. The detailed setup for Experiment 2, for word s_j of user u_j was performed as follows:

- Label each of the user's 35 instances as `owner`,
- For each character $x_i,\ i = 1, \ldots N$ in string s_j, calculate the parameters μ_i and σ_i, and similarly the average and standard deviation of the press-to-release times (μ_i' and σ_i') using the remaining users' ($u_k \neq u_j$) instances,
- Using the parameters as arguments for GaussianBot, Algorithm 1, generate $n = 35$ bot instances and label them `unknown`
- Perform a 10-fold cross-validation for SVM classification using the `owner` and `unknown` data sets,
- Calculate the average true positive (TP) and false positive (FP) rates.

The procedure for Experiment 3 is the same, using NoiseBot and parameters computed as $\eta_i = \sigma_i/2$ and $\eta_i' = \sigma_i'/2$. Table 3 shows the results of Experiments 2 and 3. In summary, the successes of the GaussianBot and NoiseBot in breaking the model are negligible, as

#	Experiment series	Purpose	Tests on Gender
1	Human vs. Human	To distinguish between two users	Yes
2	Human vs. GaussianBot	To distinguish between a user and a GaussianBot (Algorithm GaussianBot)	No
3	Human vs. NoiseBot	To distinguish between a user and and a NoiseBot (Algorithm *NoiseBot*)	No

TABLE 1
The setup of three series of experiments. We evaluate the following strings in all experiments: `www.amazon.com`, `1calend4r`, `deianstefan@gmail.com`. For human vs. human experiments, we also perform separate analysis on different gender groups and also evaluate additional strings: `google.com` and `yao.danfeng@gmail.com`.

String	Female		Male		Both	
	TP	FP	TP	FP	TP	FP
`google.com`	93.68%	5.56%	92.00%	5.50%	91.86%	4.53%
`www.amazon.com`	94.00%	4.46%	94.71%	4.62%	91.71%	2.89%
`1calend4r`	92.29%	5.69%	92.57%	7.51%	89.29%	4.48%
`yao.danfeng@gmail.com`	96.26%	2.90%	95.14%	3.17%	94.00%	2.26%
`deianstefan@gmail.com`	95.29%	3.68%	96.00%	2.90%	94.43%	2.79%

TABLE 2
Human vs. human true positive(TP) and false positive (FP) SVM classification results. Real email addresses are anonymized.

String	GaussianBot		NoiseBot	
	TP	FP	TP	FP
`www.amazon.com`	96.29%	2.00%	100.0%	0.00%
`1calend4r`	93.74%	3.43%	97.71%	1.43%
`deianstefan@gmail.com`	96.57%	1.71%	99.71%	0.29%

TABLE 3
Human vs. bots SVM classification results.

indicated by the extremely low (average 1.5%) FP rates. Furthermore, these experiments support the results of Experiment 1 and confirm the robustness of keystroke authentication to statistical attacks that are considered.

6 Related Work

What differs our work from existing keystroke-dynamics work [2], [7], [9], [10], [17], [21], and mouse-movement-based continuous authentication work, such as [14], is that we analyze the robustness of keystroke-dynamics authentication against synthetic forgery attacks. Our empirical investigation indicates the feasibility and security of keystroke authentication against potential statistical bot attacks, as opposed to just human impostors. Our evaluation on bot attacks involves more complex implementations and attack simulations.

It is worth mentioning that there exists a fundamental difference between TUBA and CAPTCHA, which is a technique that attempts to differentiate between humans and machines on visual ability [19]. *TUBA's challenges are personalized*, whereas CAPTCHA challenges are generic. TUBA is a fine-grained authentication and identification framework, where CAPTCHA is a coarse-grained classification mechanism. Attacks on CAPTCHA are typically based on computer vision techniques and can be quite successful, as demonstrated in [11]. However, a successful attack on TUBA requires forging a specific person's keystroke patterns, which represents a personalized type of attack as the attacker needs to learn about the typing patterns of the target.

TUBA is complementary to existing traffic-based malware detection solutions. The detection results produced by other means may serve as triggers to invoke a remote authentication session. For example, TUBA can start a verification test for the user whenever BotSniffer or BotHunter identify suspicious communication patterns. The element of human behavior has not been extensively studied in the context of malware detection, with a few notable exceptions including solutions by Cui, Katz, and Tan [3] and Gummadi *et al.* [5]. They investigated and enforced the temporal correlation between user inputs and observed traffic. The BINDER work [3] describes the correlation of inputs and network traffic based on timestamps. Note, however, that TUBA does *not* rely on existing botnet detection solutions to work because the verification tests may be launched periodically or according to the trigger events defined by TUBA.

7 Conclusions and Future Work

We presented our design and implementation of a remote authentication framework called TUBA for monitoring a user's keystroke-dynamics patterns and identifying intruders. We evaluated the robustness of TUBA

through comprehensive experimental evaluation including two series of simulated bots.

The TUBA model can be adopted to be used for continuous and non-intrusive authentication in both, the standalone and client-server, architectures by monitoring frequently typed strings, such as usernames, passwords, email addresses, URLs, etc. A database of these strings ($s_i,\ i = 1, \ldots, M$) and corresponding SVM models is created during an initial training phase. After the training phase we assume TUBA to be running in the background (non-intrusively) checking the stream of typed characters for matching strings in the database and only extracting features for evaluation against the trained models when a match occurs. When a match occurs the features of the typed string are classified as either `owner` or `unknown`. After a number of instances are classified as `unknown` the user is notified of the suspicious behavior and (depending on the chosen configuration) the computer may be automatically locked, under the assumption that it's under attack. Conversely, if the majority of the instances are classified as `owner` then no suspicion arises. We will carry out more investigation on the continuous authentication problem in our future work.

REFERENCES

[1] C. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

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

[3] W. Cui, R. H. Katz, and W. tian Tan. Design and implementation of an extrusion-based break-in detector for personal computers. In *ACSAC*, pages 361–370. IEEE Computer Society, 2005.

[4] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. Bothunter: Detecting malware infection through IDS-driven dialog correlation. In *Proceedings of the 16th USENIX Security Symposium (Security)*, 2007.

[5] R. Gummadi, H. Balakrishnan, P. Maniatis, and S. Ratnasamy. Not-a-Bot: Improving service availability in the face of botnet attacks. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NDSI)*, 2009.

[6] T. Hastie and R. Tibshirani. Classification by pairwise coupling. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *NIPS*. The MIT Press, 1997.

[7] J. Ilonen. Keystroke dynamics. *Advanced Topics in Information Processing–Lecture*, 2003.

[8] S. Keerthi, S. Shevade, C. Bhattacharyya, and K. Murthy. Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computation*, 13(3):637–649, 2001.

[9] K. S. Killourhy and R. A. Maxion. The effect of clock resolution on keystroke dynamics. In R. Lippmann, E. Kirda, and A. Trachtenberg, editors, *RAID*, volume 5230 of *Lecture Notes in Computer Science*, pages 331–350. Springer, 2008.

[10] F. Monrose and A. Rubin. Keystroke dynamics as a biometric for authentication. *FUTURE GENER COMPUT SYST*, 16(4):351–359, 2000.

[11] G. Mori and J. Malik. Recognizing objects in adversarial clutter: breaking a visual CAPTCHA. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 134–141, 2003.

[12] B. D. Payne and W. Lee. Secure and flexible monitoring of virtual machines. In *ACSAC*, pages 385–397. IEEE Computer Society, 2007.

[13] J. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods - Support Vector Learning*, chapter 12. 1998.

[14] M. Pusara and C. E. Brodley. User re-authentication via mouse movements. In C. E. Brodley, P. Chan, R. Lippman, and W. Yurcik, editors, *VizSEC*, pages 1–8. ACM, 2004.

[15] J. Rutkowska. Introducing stealth malware taxonomy, November 2006.

[16] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a tcg-based integrity measurement architecture. In *USENIX Security Symposium*, pages 223–238. USENIX, 2004.

[17] D. Song, D. Wagner, and X. Tian. Timing analysis of keystrokes and SSH timing attacks. *Proceedings of the 10th USENIX Security Symposium*, 2001.

[18] D. Stefan, C. Wu, D. Yao, and G. Xu. Cryptographic provenance verification for the integrity of keystrokes and outbound network traffic. In *Proceedings of the 8th International Conference on Applied Cryptography and Network Security (ACNS)*, June 2010.

[19] L. von Ahn, M. Blum, and J. Langford. Telling humans and computers apart automatically. *Commun. ACM*, 47(2):56–60, 2004.

[20] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, 2 edition, 2005. WEKA available at http://www.cs.waikato.ac.nz/ml/weka/.

[21] E. Yu and S. Cho. Novelty detection approach for keystroke dynamics identity verification. *LNCS*, pages 1016–1023, 2003.

[22] K. Zhang and X. Wang. Peeping tom in the neighborhood: Keystroke eavesdropping on multi-user systems. In *Proceedings of the USENIX Security Symposium*, 2009.