Available online at www.sciencedirect.com

SciVerse ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Robustness of keystroke-dynamics based biometrics against synthetic forgeries☆

Deian Stefan [a], *Xiaokui Shu* [b], *Danfeng (Daphne) Yao* [b,*]

[a] *Department of Electrical Engineering, The Cooper Union, New York, NY 10003, United States*
[b] *Department of Computer Science, Virginia Tech, 2202 Kraft Dr, Blacksburg, VA 24060, United States*

ARTICLE INFO

Article history:
Received 21 May 2011
Received in revised form
22 August 2011
Accepted 4 October 2011

Keywords:
Keystroke dynamics
Authentication
Malware detection
Forgery
Bot
Attack
Classification

ABSTRACT

Biometric systems including keystroke-dynamics based authentication have been well studied in the literature. The attack model in biometrics typically considers impersonation attempts launched by human imposters. However, this attack model is not adequate, as advanced attackers may utilize programs to forge data. In this paper, we consider the effects of *synthetic forgery attacks* in the context of biometric authentication systems. Our study is performed in a concrete keystroke-dynamic authentication system.

The main focus of our work is evaluating the security of keystroke-dynamics authentication against synthetic forgery attacks. Our analysis is performed in a remote authentication framework called TUBA that we design and implement for monitoring a user's typing patterns. We evaluate the robustness of TUBA through experimental evaluation including two series of simulated bots. The keystroke sequences forged by the two bots are modeled using first-order Markov chains. 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 the two specific types of synthetic forgery attacks studied, where attacker draws statistical samples from a pool of available keystroke dataset other than the target.

We also describe TUBA's use for detecting anomalous activities on remote hosts, and present its use in a specific cognition-based anomaly detection system. The use of TUBA provides high assurance on the information collected from the hosts and enables remote security diagnosis and monitoring.

1. Introduction

Keystroke-dynamics based authentication is a cheap biometric mechanism that has been proven accurate in distinguishing individuals (Bleha et al., 1990; Ilonen, 2003; Killourhy and Maxion, 2008; Monrose and Rubin, 2000; Song et al., 2001; Yu and Cho, 2003). 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 been little effort on studying the robustness of this technique against synthetic and automatic attacks and forgeries.

We evaluate the robustness of keystroke-based biometric authentication systems against a new type of forgery attacks. In the context of biometrics, a synthetic forgery attack is carried out by submitting generated or synthesized credentials to an authentication module. For example, an attacker writes a program that performs statistic manipulation and synthesis to produce keystroke sequences in

☆ This work was supported in part by Rutgers University DIMACS REU programs, National Science Foundation grants CNS-0831186 and CAREER CNS-0953638. Stefan is currently a graduate student at Stanford University.

* *Corresponding author.*
E-mail addresses: stefan@cooper.edu (D. Stefan), danfeng@cs.vt.edu, subx@cs.vt.edu (D. (Daphne) Yao).

0167-4048/$ – see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cose.2011.10.001

order to spoof others. These types of forgery attacks pose a serious threat. However, the research community has not extensively investigated on possible anti-forgery techniques. It is unclear from the current literature how robust keystroke dynamics is against forgery attacks. Synthetic forgery attacks may also be possible in other types of biometric systems as well.

The technical enabler for our investigation is a remote authentication framework that we design and implement. The framework called TUBA (Telling hUman and Bot Apart) monitors a user's typing patterns in a client-and-server architecture. We systematically study the robustness of TUBA through comprehensive experimental evaluation including two simulated bots. We perform a user study with 20 users and use the collected data to simulate and evaluate the difficulty and impact of synthetic forgeries.

Another contribution of this paper is that we describe the use of TUBA and keystroke dynamics to 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 launches attacks. This model is motivated by the increasing number of infected hosts caused by organized malicious botnets. Our solution provides strong assurance of authentication results. We provide a practical solution that effectively allows a remote trusted server to monitor the integrity of a computer. The main application of TUBA is to detect stealthy malware residing on a user's computer such as application-level spyware.

Our study uniquely combines techniques from system and network security, biometrics, machine learning, and usability engineering. Our technical 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 (X11 for short).
2. We analyze the 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 classification accuracy 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. The bots are capable of launching forgery attacks drawn upon the statistical analysis of collected keystroke data. Experiments show that our classification is robust against these specific attacks, and is able to correctly classify the attacks by GaussianBot and NoiseBot with low false positive rates. The GaussianBot and NoiseBot forge keystroke sequences following simple first-order Markov models.

TUBA is particularly suitable for detecting extrusion in enterprises and organizations, and protecting the integrity of hosts. Our work gives the indication that certain human behaviors, namely user inputs, may be suitable for malware detection purposes. We also give examples that illustrate the prevention of 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. Our work also suggests the need for studying the robustness of other biometrics against synthetic forgery attacks beyond the studied keystroke-authentication problem. Because of the wide use biometrics in government, military, and enterprise environments, the better understanding of their security against sophisticated attacks is important.

1.1. *Organization of the paper*

We describe our design of 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. We implement two bots that are capable of injecting synthetic keystroke events, which are presented in Section 4. Our experimental evaluation results and user study are described in Section 5. A specific application of TUBA for liveliness detection as well as an open problem are presented in Section 6. Related work is described in Section 7. In Section 8, we conclude the paper and describe plans for future work.

2. Overview and security model

TUBA 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.

2.1. *Security assumptions and malware attack model*

We assume that the host operating system kernel, our client-side keystroke-collection modules, and cryptographic keys 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, 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 synthesized from the data other than that of the owner. Thus, under this malware-attack model we assume that keylogging by spyware or by human intruders (Zhang and Wang, 2009) on the owner's computer does not exist. 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 chip TPM (trusted platform module) enabled, fake key events can be detected and removed

with reasonable overhead (Gummadi et al., 2009; Stefan et al., 2010). 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 (Sailer et al., 2004) or virtualization based introspection (Payne and Lee, 2007), which are not considered in this paper.

2.2. Definitions and design overview

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 a training phase, where the remote authentication server collects keystroke data from a legitimate user. We assume that the user's computer is not infected during training. 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. Both M and n are parameters of the model. 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.

When a suspicious network event is observed, TUBA prompts the user with a TUBA challenge – a window requesting him/her to type in a server-chosen string. 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 BINDER (Cui et al., 2005), DeWare (Xu et al., 2011), or according to other pre-defined policies.

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 (easily identified using lsof or netstat), certain user-initiated network activities such as sending email without keyboard/mouse inputs or with an active screensaver, listening sockets on suspicious ports, sending high-volume traffic to a single target host, attempting to disable the bot detection program, etc. For example, if the computer is used to send email with spam-like characteristics, having an unusual chat application running (possible IRC-based C&C channel), or periodically visiting a server in a foreign country with no hostname or other records (possible HTTP-based C&C channel).

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

2.3. Prototype implementation

The architecture of TUBA in client-server model is illustrated in Fig. 1. We describe the use of X server for keystroke forwarding from the client to the trusted server next. In TUBA, a trusted remote server is responsible for the data collection and analysis in a remote fashion, e.g., using SSH (Secure Shell) the client remotely logins to the server with X11-forwarding enabled so that the keystroke events are monitored by the server. The connection and storage of the remote server are assumed to be secure. Various keylogging methods for the GNU/Linux operating system exist. Common implementations include user-space programs which monitor privileged I/O ports (Keylogger, 2001), kernel modules that intercept the `sys_read` and `sys_write` functions (Kernel, 2001), and kernel modules that intercept the keyboard driver's interrupt handler (linux kernel keylogger, 2001). 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 (X or X11 for short) is a powerful graphical user interface composed of the X server and X *clients*. The X server

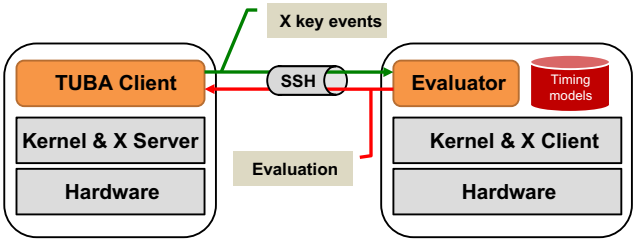

Fig. 1 – TUBA architecture in a client-server model. The keystroke events are collected on the client (left), and sent to the server (right) in a secure communication channel. The classification is performed on the server.

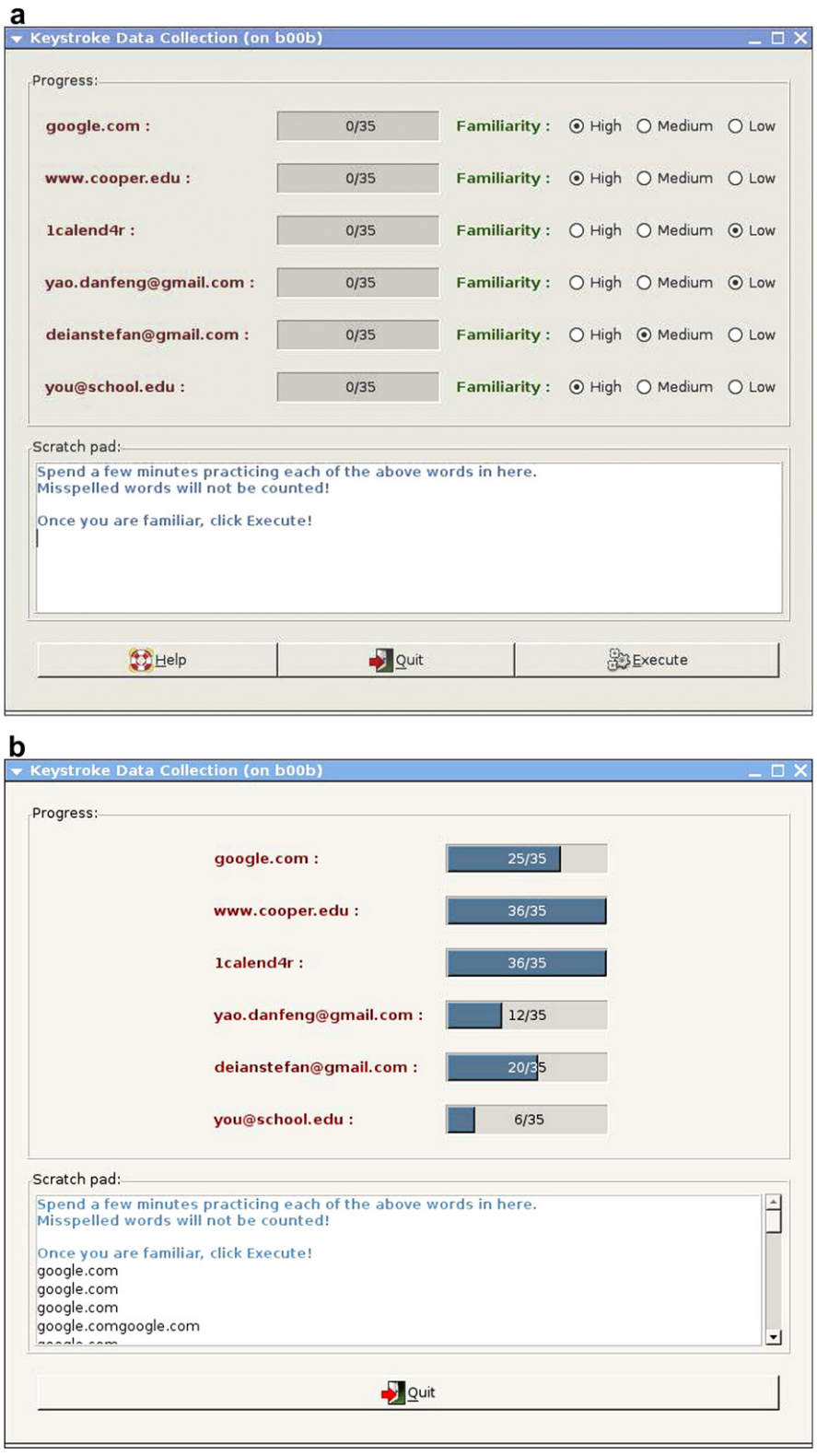

Fig. 2 – Screenshots of our data collection program at it starts in (a), and during the recording in (b). Participants report their familarity with the strings to be typed, and are given the opportunity to practice them. Each correct string is entered 35 times. Incorrect strings are ignored.

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 architecture design 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 intercept the core input events and forward them to XTrap client applications.

As such, our keylogger (client application) contains a callback function which 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. The communication channel between the client and server can be secured with SSL. Below is an example of the partial output of the keylogger when typing the word "bot".

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

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 $(s_i, i = 1, \ldots, M)$ and feature descriptions (i.e., keystroke durations, total time to type a word, press-to-press times, etc.), when the buffer contents of the keyboard input matches a database word, the features are extracted. The parameter C is adjusted to match the largest word in the database. An example of outputs from the feature extractor by typing `bot` 6 times is shown below, where PP, PR, RR, and duration respectively refer to the press-to-press time, press-to-release time, release-to-release time, and the duration of each character (see Table 1). The numbers are in milliseconds. Negative press-to-release time means that a character is pressed before the previous character is released.

```
1 @word=bot|PP=227,63|PR=100,-28|RR=191,92|
  duration=127,91,120|total=410
2 @word=bot|PP=190,56|PR=105,-10|RR=171,83|
  duration=85,66,93|total=339
3 @word=bot|PP=117,84|PR=32,9|RR=107,103|
  duration=85,75,94|total=295
4 @word=bot|PP=107,82|PR=6,-6|RR=94,83|
  duration=101,88,89|total=278
5 @word=bot|PP=123,130|PR=16,56|RR=90,141|
  duration=107,74,85|total=338
6 @word=bot|PP=125,125|PR=5,31|RR=99,115|
  duration=120,94,84|total=334
```

3. Feature extraction and classification

In this section, we describe the feature extraction and classification performed in TUBA on keystroke data, as well as our Markov chain model used for simulating keystroke-forgery attacks. We illustrate how the dimensionality affects the classification results and its security implications.

3.1. *Features and dimension reduction*

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 inter-key timings such as the interval between two adjacent press or release events. Even for a short string such as `www.amazon.com`, the dimensionality of all possible features is quite high. We give a concrete example to illustrate the high dimensionality of the extracted keystroke features. Considering the string `www.cooper.edu` that contains $N = 14$ characters, TUBA extracts the group of features shown in Table 1.

Utilizing a large number of features is desirable because it typically results in a better (more specific) prediction model. More importantly, a high-dimensional feature vector used in the classification makes it difficult for adversaries to successfully simulate keyboard events that pass our classification test. We illustrate an example with keystroke data from two real users (collected in our user study, as described in Section 5), in which classification results improve with the

Table 1 – An example of features extracted from keystroke data for the string `www.cooper.edu`.

Feature name	# Dimensions	Description
Total duration	1	Time for keying the entire string
Total duration	1	Time difference between $x_{N,r}$ and $x_{1,p}$
Duration of each character (D(a))	14	Duration of a key being pressed
Duration of a key being of character (D(x_i))	14	Time difference between $x_{i,r}$ and $x_{i,p}$
Press-to-press time (PP(a,b))	13	Time between the beginning of a being pressed and the beginning of b being pressed
Press-to-press time (PP(x_{i-1},x_i))	13	Time difference between $x_{i,p}$ and $x_{i-1,p}$
Press-to-release time (PR(a,b))	13	Time between the beginning of a being pressed and the beginning of b being released
Press-to-release time (PR(x_{i-1},x_i))	13	Time difference between $x_{i,p}$ and $x_{i-1,r}$
Release-to-release time (RR(a,b))	13	Time between the beginning of a being released and the beginning of b being released
Release-to-release time (RR(x_{i-1},x_i))	13	Time difference between $x_{i,r}$ and $x_{i-1,r}$
Release-to-press time (RP(a,b))	13	Time between the beginning of a being released and the beginning of b being pressed

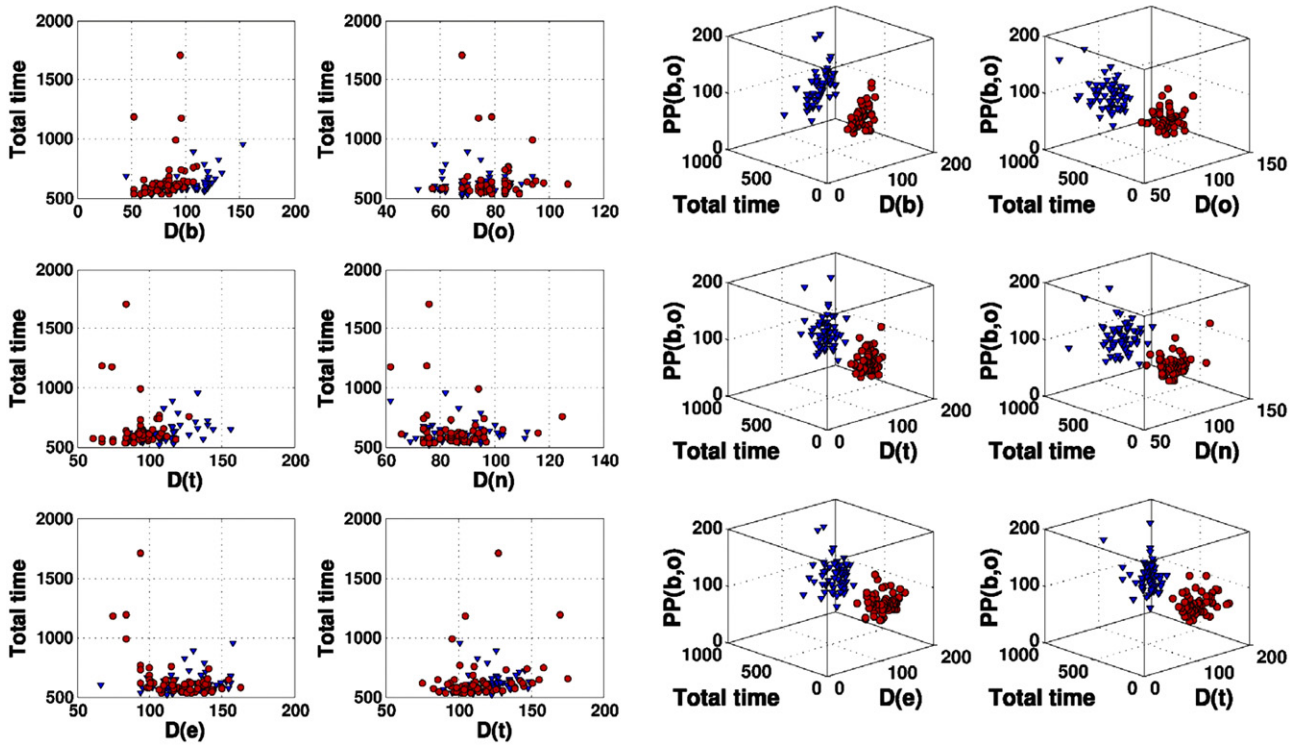

Fig. 3 – (Left) Distribution of two keystroke features between two users, i.e., (1) keystroke duration (in milliseconds) of individual characters from the string "botnet" in X-axis and (2) total time spent to type the word in Y-axis. (Right) Distribution of three keystroke features between two users, i.e., (1) and (2) as in (left) and (3) the press-to-press time in Z-axis. One user's data is shown with the red circles, and the other user's with blue triangles. D(b), D(o), D(t), … represent the keystroke duration of character b, o, t, …, respectively. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

increasing number of features used. The word "botnet" is typed by two individuals and as shown in Fig. 3.

Two features are considered in Fig. 3 (left) – key durations of individual characters and the total typing time of a word. With these two keystroke features, the two users' samples are somewhat differentiated from each other, but there still exists significant overlap between the data sets. In Fig. 3 (right), three keystroke features are used, including the press-to-press time of two adjacent characters, key durations of individual characters, and the total typing time of a word. The two users' samples are well-separated using a 3-dimensional feature vector in this example. This series of experiments show the importance of using multi-dimensional keystroke features for user classification, compared to previous studies that rely on a small number of features for identification (Bleha et al., 1990; Ilonen, 2003).

However, the high-dimensional feature space makes classification worse if not enough samples are collected, as the model tends to overfit the training data and therefore produce a classification model which incorrectly classified new data – overfitting problem. 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 dataset so that high variations within the data are retained, i.e., the principal components are retained. In our experiments, running PCA reduces the number of dimensions to, on average, one third of the original value. Using SVM with PCA-preprocessed features nearly doubles the classification accuracy; without PCA-preprocessing, our classification results were roughly 60%. We note that all the computations, including classification, are performed on the server-end. Hence, when a compromised machine is detected and the owner can be notified in a timely fashion.

3.2. *SVM-based classification*

Once keystroke features are collected and processed, we train and classify the data using support vector machine (SVM). The use of SVM 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 (Hastie and Tibshirani, 1997; Keerthi et al., 2001; Platt, 1998). 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,

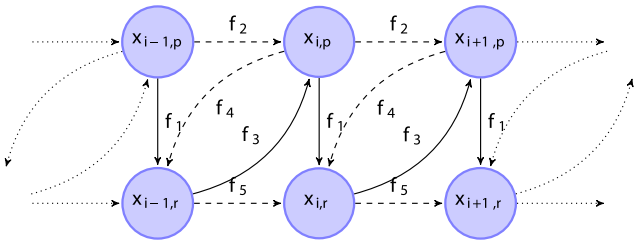

Fig. 4 – Comparisons between the typing abilities of a person and a bot modeled by using a first-order Markov chain of keystroke events. Nodes $x_{i,p}$ and $x_{i,r}$ denote the i -th letter press and release events, respectively. Linear combinations of the f_k elements represent the timing features used for the classification. Solid lines represent a bot's typing sequence in a first-order Markov model, and a user's sequence may be represented by both solid and dotted lines, allowing consecutive press events.

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 (Witten et al., 2005) SVM implementation with a Gaussian RBF kernel.

Our data analysis and classification tools used, including those for principle component analysis and support vector machines, are from WEKA (Witten et al., 2005). WEKA is a widely-used open-source data mining toolkit written in Java. The graphic user interface provided by WEKA, Knowledge Flow, makes the tool user-friendly to researchers. We refer readers to data mining and machine learning literature such as the book by Witten and Frank (Witten et al., 2005) or Bishop (Bishop, 2006) for detailed descriptions of SVM techniques. During our experiments, the SVM parameters were manually tuned to obtained the optimal classification results. For every user and every word we created a model – for a total of 120 models – we however did not find a general approach or intuition to finding the SVM and RBF parameters; instead, our tuning approach was more along the lines of brute-force, and we thus do not show the final chosen parameters.

3.3. *Assessing the difficulty of impersonation attacks*

Existing literature on keystroke-dynamics based authentication does not consider programs or bots as the potential type of adversaries when analyzing the security. We assess the difficulty for an attacker to program a bot to impersonate a target's typing patterns. To pass our classification, the bot needs to mimic the target's typing behaviors in a way such that the extracted features are similar as well.

Humans are imperfect typists and may create 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 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 $PR(x_{i-1}, x_i) = x_{i,p} - x_{i-1,r} < 0$. From our experimental data, we find that a large number of users have negative press-to-release timings in their dataset. Although an adversary can synthesize arbitrary keystroke events, we find that it is considerably more difficult to create an intelligent bot which can inject keystroke events that result in negative inter-key timings. Bot writers typically care more about infecting new machines than addressing all defensive issues; thus spending the time to build advanced models and write a complex bot is often not a worthy pursuit for them.

Fig. 4 illustrates the differences in the capabilities between human and bots in the first-order Markov model. Assuming that keystroke events can be modeled accurately by a first-order Markov chain, a human's key event path can 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 Fig. 4). First-order Markov chain is memoryless, and the transition to the next state is determined solely based on the current state. In higher-order Markov chains, the previous states also affect the transition and can support more sophisticated keystroke sequences.

When considering higher-order Markov chains, it is even more challenging for the attackers or bots to successfully mimic typing patterns with negative timing; a person may, for example, press 'c' before both 'a' and 'b' are released. Using high-dimensional data leads to higher authentication accuracy and stronger security guarantees. We note that if the complexity of the model is increased (e.g., to a second- or third-order Markov chain), it is important to collect additional training instances as to avoid overfitting the data.

Based on our analysis, negative inter-key timings and durations showed to be the most distinguishing features. The latter feature is apparent in long words, as users tend to take pauses in different positions. Moreover, this information is apparent to a bot only in the variance, effectively making it less useful. As the bots we consider do not support negative inter-key timings, this feature is particularly distinguishing. It is important to note, however, that a bot using negative timings in the Markov model will still have a key difference from humans: a person typing is physically constrained and will thus finish writing a word; conversely, a bot has no such

physical limitations and could, at least in principle, not terminate. A simple Markov model is not sufficient to model a human, constrained by a physical environment, as explained above. Although with sufficient effort a sophisticated, terminating bot can be implemented, we believe that, for the target problem, there is little incentive for bot herders to develop such complex bots as opposed to simply compromising other machines.

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 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 creates fake keystroke events expecting them to be classified as the owner's. Under our adversary model, we assume that bots possess keystroke data of any users except the owner's. As described in Section 2, replaying the owner's keystroke sequence is disallowed (see (Stefan et al., 2010) for an extension which considers an attack model that includes replay attacks).

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 a 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 Fig. 4. That is, bots consider only keystroke durations and positive inter-key timings (paths shown by the solid lines in Fig. 4).

In our simulations, the keystroke duration of the i -th character in a word is modeled as a random variable $X_i \geq 0$, where X_i is:

1. Gaussian with mean μ_i and variance $\sigma_i^2 : X_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ in GaussianBot, or
2. Constant with additive uniform noise $(\text{mean } 0) : X_i \sim \mu_i + \mathcal{U}(-\eta_i, \eta_i)$ in 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. Similarly, the press-to-release inter-key timing feature between the i -th and $(i-1)$ -th character is 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 bots, which inject n instances of the given string. 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.

Similar to Algorithm 1, the pseudocode for a bot which generates noisy instances (i.e., mean $\pm$ noise) is shown in Algorithm 2. The parameters for Experiment 3 were calculated as those for GaussianBot in Experiment 2, with the noise parameters $\eta_i = \sigma_i/2$ and $\eta'_i = \sigma'_i/2$.

Algorithm 1. *GaussianBot* simulation of a human typing.

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 typing.

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. Screenshots of the GUI are shown in Fig. 2. 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 used are also kept constant.

5.1. *Analysis on the user group*

The participants were mostly college students who interact with computers on a daily basis and proficient with typing in general. The age distribution of participants is shown in Fig. 5. We required the participants to categorize their familiarity levels with the six strings into *high*, *medium*, or *low*. Participants mostly have the high familiarity with website URLs such as `google.com` and `www.cooper.edu`. We quantify the means and standard deviations of their familiarity levels, after assigning 1 to *low*, 2 to *medium*, and 3 to *high* in Fig. 6. We expect that the classification results may differ for a group of less sophisticated users.

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 2.

5.2. *Experiment 1 (Human vs. Human)*

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 a high accuracy in classifying individual humans.

Among the 20 users, we set up a basic 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: $c_1 = \{$all male instances$\}$, $c_2 = \{$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 true positive (TP) and false positive (FP) rates.

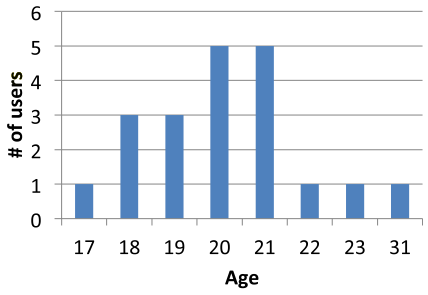

Fig. 5 – The age distribution of participants in our user study.

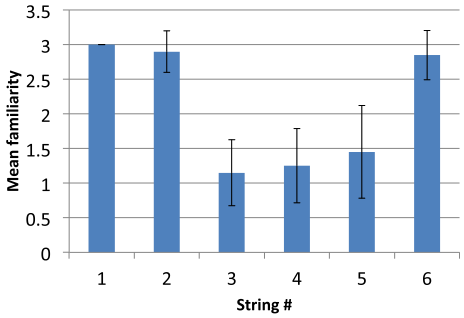

Fig. 6 – The means and standard deviations of users' familiarity levels for the six strings evaluated. 3 is for high familiarity, 2 for medium, and 1 for low. The order of strings is as shown in Fig. 2.

The classification analysis was repeated for all the user subjects, and words in the database and classification sets. Finally, the average TP and FP rates for every word and class (1. male, 2. female, and 3. both) were calculated. The results are summarized in Table 3 – 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. In our work, a true positive represents a correctly classified user (e.g., Alice is correctly being recognized), and a false positive means that a user is wrongfully classified (e.g., Bob or an adversary is classified as Alice). As shown in Table 3 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 user is 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 (such as `yao.danfeng@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 pause (unconsciously) mid-word, which is reflected by some of the inter-key timings.

5.3. *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 relatively sophisticated bot attacks. We evaluate the robustness of keystroke analysis against artificially 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 tuning 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 very slightly, 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 and false positive rates.

The procedure for Experiment 3 is the same for the NoiseBot. Table 4 shows the results of Experiments 2 and 3. In summary, the successes of the GaussianBot and NoiseBot in breaking the model are negligible, as 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 the two specific types of forgery attacks.

We note that replay attacks are only possible in the weak adversary model (defined in Section LABEL:definitions) when the TUBA integrity service is not deployed. Nevertheless, we carry out the bot simulation and event injection studies to investigate the robustness of classification algorithms.

6. Discussion and open question

In this section, we describe how TUBA can be integrated with an existing anomaly detection model that leverages the cognitive ability of users for security. We also describe an

Table 2 – 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`.

#	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 3 – Human vs. human true positive (TP) and false positive (FP) SVM classification results.

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%

open question in TUBA about how to further strengthen its security against a stronger adversary model.

6.1. *TUBA application: keystroke-based liveliness detection*

TUBA can be used as a practical tool for liveliness detection and integrated with host-based tools for security monitoring. The purpose of host-based liveliness detection is to ensure that the computer is being used by a human user. Such detection can be utilized to remotely diagnose and monitor a networked computer and identify anomalous activities. A simple liveliness detection by leveraging keystroke activities of users is for a server to observe whether there exists any keystroke events on the host. However, malware may inject fake keystroke events as we demonstrated in our experiments, and pass the test. In TUBA, the liveliness detection on the client is triggered by the remote server upon detecting certain types of suspicious network activities. The policies specifying suspicious patterns can be defined based on external network sensors or anomaly detection tools. The analysis on keystroke-dynamics provides high assurance to the test results.

Xiong et al. (2009) proposed a cognition-based anomaly detection mechanism that requires a user to answer cognitive questions in pop-up windows. Specifically, the questions require the user to recognize her own network activities upon detecting suspicious events. The user answers the questions by clicking on the Yes or No buttons in the dialog window. For example, if the user does not initiate the suspicious connection to the URL, then the user enters No. However, that system does not provide any authentication mechanism for the users' inputs, and may be vulnerable as malware may intercept the questions and spoof the user's answers.

TUBA can be integrated with this cognition-based anomaly detection system to improve its assurance. Specifically, the user' questions are accompanied with TUBA challenges. The user responses not only to the question regarding the suspicious network activities, but also enters a sequence of characters for keystroke-dynamic based authentication. The server first verifies that the keystroke events can be correctly classified and then processes the user's response to the cognitive questions. Such a framework is simple and can effectively allow the server to obtain trustworthy information from participating users at real time for anomaly detection. In comparison to CAPTCHA test, our keystroke authentication is more fine grained and personalized. Compared to password-based authentication, keystroke-dynamics biometric is more difficult to forge.

Table 4 – Human vs. bots SVM classification results.

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%

6.2. *Open problem*

Keyloggers are, in general, difficult to remove (Ortolani et al., 2010). There exists the demonstration of soft-timer based keyloggers that do not need to change any kernel code or data (Wei et al., 2008). Our model assumes that the attacker is unable to log and replay the target's keystroke sequences. One approach to relax this assumption is to strengthen the host security by utilizing specialized hardware namely trusted platform module. For example, in (Stefan et al., 2010) each keystroke event is cryptographically signed on the client and verified by a remote server. Due to hardware constraints and complexity of such solutions, the TPM-based approach may not be practical in some scenarios.

A simple mitigation is for the server to generate one-time challenges for the client that are never reused. The challenge is a string that the user needs to type, the timing patterns of which are evaluated by the server for authentication. Even if the client is infected with stealthy keylogger, the keylogger is unable to replay old sequences of logged keystroke events. However, such a scheme requires a large amount of training data – the server keeps a record of keystroke data from the user corresponding to the one-time challenges to be issued at the setup phase. Without the proper training data for specific challenges, the classification cannot be performed accurately. Gathering a large amount of training data *a priori* can be cumbersome.

An open problem is how or whether it is possible to classify keystroke-dynamic data for authentication with a small (e.g., sublinear) amount of training data. For example, is it possible to recognize and classify keystroke data of new n strings each with m characters with training dataset of size $O(m\log n)$ or $O(n\log m)$, which are much smaller than $O(nm)$?

7. Related work

Keystroke-dynamics based authentication has been extensively studied in the security and machine learning literature.

Existing work on this topic mainly focuses on the use of keystroke-dynamics analysis for biometric authentication. Recently, Killourhy and Maxion performed thorough comparative analysis on state-of-the-art keystroke dynamics solutions, and presented a methodology for predicting classification error rates (Killourhy and Maxion, 2010). What differs our work from existing keystroke-dynamics work (Bleha et al., 1990; Ilonen, 2003; Killourhy and Maxion, 2008; Monrose and Rubin, 2000; Song et al., 2001; Yu and Cho, 2003) or mouse-movement based continuous authentication such as in (Pusara and Brodley, 2004) is that we analyze the robustness of keystroke-dynamics authentication against automatic synthetic forgery attacks, as opposed to impersonation attacks launched by other persons. We also describe the practical challenges and solutions in applying keystroke-dynamic analysis for remote diagnosing the computer integrity and detecting stealthy malware. Our empirical investigation indicates the feasibility and security of keystroke authentication against two bot attacks, as opposed to just human impostors.

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 (von Ahn et al., 2004). *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 typically are based on computer vision techniques and can be quite successful, as demonstrated in (Mori and Malik, 2003) for example. 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.

Our work also belongs to the new line of research that utilizes behavior-based characteristics of human users for enforcing security properties of systems and networks. The element of human behavior has not been extensively studied in the context of malware detection, with a few notable exceptions including solutions such as Cui et al. (2005) and Gummadi et al. (2009). Gummadi et al. (2009) proposed a bot detection solution on a personal computer that used hardware-assisted certification mechanism to distinguish human-generated traffic from malware-generated activities. Their solution requires a trusted proxy server to certify keystroke events entered by the user. Shirley and Evans (2008) proposed to generate and enforce access-control policies for file systems based on user intentions that are inferred from the context of a transaction on a host. The BINDER work (Cui et al., 2005) describes the correlation of inputs and network traffic based on timestamps. Recently, mouse-click behaviors are leveraged to detect drive-by download exploits (Xu et al., 2011). The work on behavior-driven malware detection approaches presents new technical challenges, but also may hold promises for producing next generation cyber defenses.

8. Conclusions and future work

This paper addressed the important problem of biometric security, in particular the robustness of keystroke-based biometric authentication against automatically generated keystroke sequences from attackers. Our work recognizes the security gap that exists in the current biometric research, where adversaries are limited to human users. In order to evaluate the impact of synthetic forgery in the keystroke-dynamic authentication, 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. Our analysis is based on data collected from 20 users in a focused user study. We used support vector machine for classification in all our experiments. We performed experiments and found that given the first-order Markov chain model, our classification is robust against synthetic forgery attacks studied. The bot-generated keystroke sequences are detected with high true positive rates (>93%). We described how TUBA can be integrated with other anomaly detection systems to achieve remote monitoring and diagnosis of hosts with high assurance. The uniqueness of such security tools is the leveraging of human-behavior characteristics for enforcing system and network security properties.

Our work is a first step towards understanding the robustness of biometric techniques against synthetic forgeries. Because of the sophistication and adaptivity of modern malware, our future work requires more thorough and comprehensive evaluation of other advanced forgery patterns including higher-order Markov chains. We will also carry out more investigation on the continuous and liveliness authentication problem in our future work. The TUBA model can be adopted to be used for continuous and non-intrusive authentication in both, the stand-alone and client-server, architectures by monitoring frequently typed strings, such as usernames, passwords, email addresses, URLs, etc. A database of these strings 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 incorrectly classified, 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.

REFERENCES

Bishop C. Pattern recognition and machine learning. Springer; 2006.

Bleha S, Slivinsky C, Hussien B. Computer-access security systems using keystroke dynamics. IEEE Transactions on Pattern Analysis and Machine Intelligence 1990;12(12):1217–22.

Cui W, Katz RH, Tian Tan W. In: Design and implementation of an extrusion-based break-in detector for personal computers. IEEE Computer Society; 2005. p. 361–70. ACSAC.

Gummadi R, Balakrishnan H, Maniatis P, Ratnasamy S. Not-a-bot: improving service availability in the face of botnet attacks. In: NSDI'09: proceedings of the 6th USENIX symposium on networked systems design and implementation. Berkeley, CA, USA: USENIX Association; 2009. p. 307–20.

Hastie T, Tibshirani R. In: Jordan MI, Kearns MJ, Solla SA, editors. Classification by pairwise coupling. The MIT Press; 1997. NIPS.

Ilonen J. Keystroke dynamics, http://www2.it.lut.fi/kurssit/03-04/010970000/seminars/Ilonen.pdf; 2003.

Keerthi S, Shevade S, Bhattacharyya C, Murthy K. Improvements to platt's smo algorithm for SVM classifier design. Neural Computation 2001;13(3):637–49.

Lkl linux keylogger, http://sourceforge.net/projects/lkl/; 2001.

Kernel Based Keylogger. http://packetstormsecurity.org/UNIX/security/; 2001.

Killourhy KS, Maxion RA. The effect of clock resolution on keystroke dynamics. In: Lippmann R, Kirda E, Trachtenberg A, editors. RAID. Lecture notes in computer science, vol. 5230. Springer; 2008. p. 331–50.

Killourhy K, Maxion R. Why did my detector do that?! predicting keystroke-dynamics error rates. In: Proceedings of the Recent Advances in Intrusion detection (RAID). Lecture notes in computer science, vol. 6307; 2010. p. 256–76.

rd, writing linux kernel keylogger, Phrack Magazine 12(14), http://freeworld.thc.org/papers/writing-linux-kernel-keylogger.txt; 2001.

Monrose F, Rubin A. Keystroke dynamics as a biometric for authentication. Future Generation Computer Systems 2000; 16(4):351–9.

Mori G, Malik J. Recognizing objects in adversarial clutter: breaking a visual CAPTCHA, in: Proceedings of the IEEE computer society conference on computer vision and pattern recognition, 2003, pp. 134–141.

Ortolani S, Giuffrida C, Crispo B. Bait your hook: a novel detection technique for keyloggers. In: Jha S, Sommer R, Kreibich C, editors. RAID. Lecture notes in computer science, vol. 6307. Springer; 2010. p. 198–217.

Payne BD, Lee W. In: Secure and flexible monitoring of virtual machines. IEEE Computer Society; 2007. p. 385–97. ACSAC.

Platt J, Fast training of support vector machines using sequential minimal optimization, in: Advances in kernel methods - support vector learning, 1998, Ch. 12.

Pusara M, Brodley CE. In: Brodley CE, Chan P, Lippman R, Yurcik W, editors. User re-authentication via mouse movements. VizSEC, ACM; 2004. p. 1–8.

Sailer R, Zhang X, Jaeger T, van Doorn L. Design and implementation of a TCG-based integrity measurement architecture. In: USENIX security symposium. USENIX; 2004. p. 223–38.

Shirley J, Evans D. The user is not the enemy: fighting malware by tracking user intentions. In: NSPW '08: proceedings of the 2008 workshop on new security paradigms. New York, NY, USA: ACM. p. 33–45, http://doi.acm.org/10.1145/1595676.1595683; 2008.

Song D, Wagner D, Tian X. Timing analysis of keystrokes and SSH timing attacks, In: Proceedings of the 10th USENIX security symposium; 2001.

Stefan D, Wu C, Yao D, Xu G. 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), 2010.

von Ahn L, Blum M, Langford J. Telling humans and computers apart automatically. Communications of the ACM 2004;47(2):56–60.

Wei J, Payne BD, Giffin J, Pu C. Soft-timer driven transient kernel control flow attacks and defense. In: ACSAC '08: proceedings of the 2008 annual computer security applications conference. Washington, DC, USA: IEEE Computer Society. p. 97–107, http://dx.doi.org/10.1109/ACSAC.2008.40; 2008.

Witten IH, Frank E, Mining Data. Practical machine learning tools and techniques. 2nd ed. San Francisco: Morgan Kaufmann. Available at: http://www.cs.waikato.ac.nz/ml/weka/; 2005.

Xiong H, Malhotra P, Stefan D, Wu C, Yao D. User-assisted host-based detection of outbound malware traffic, in: Proceedings of International Conference on Information and Communications Security (ICICS), 2009.

Xu K, Yao D, Ma Q, Crowell A. Detecting infection onset with behavior-based policies, in: Proceedings of the fifth international conference on Network and System Security (NSS), 2011.

Yu E, Cho S. Novelty detection approach for keystroke dynamics identity verification. LNCS; 2003. 1016–1023.

Zhang K, Wang X. Peeping Tom in the neighborhood: keystroke eavesdropping on multi-user systems, in: proceedings of the USENIX security symposium, 2009.

Yao is an assistant professor in the Department of Computer Science at Virginia Tech, Blacksburg. She received her Computer Science Ph.D. degree from Brown University. Before joining VT, she was a assistant professor at Rutgers University CS Department for two years. Her research interests are in network and information security. She received the NSF CAREER Award in 2010 for her work on human-behavior driven malware detection. She won the Best Student Paper Award in ICICS 2006, and the Award for Technological Innovation from Brown in 2006, both for her privacy-preserving identity management work. Danfeng has one provisional patent filed for her recent bot detection techniques.

Stefan received his bachelor and master degrees from Cooper Union Electrical and Computer Engineering Department and is currently a graduate student at Stanford University.

Shu received his bachelor degree from the University of Science and Technology of China. He is currently a Ph.D. student at Virginia Tech.