RELENTLESS AUTHENTICATION OF KEYSTROKES ON ALARMINGLY HIGH RATE ERROR ATTACKS

1 U.Revathy, 2 Dr. M. Karnan, 3 Dr. S. Meenakshi Sundaram
1 Assistant professor, Department of Computer Science, Government Arts College for women, Sivagangai, India
2 Professor- Department of CSE, Aringer Anna College of Engineering and Technology, Dindigul, India.
3 Associate Professor, Department of CSE, Aringer Anna College of Engineering and Technology, Dindigul, India
urevathymeena@gmail.com, drmkaran@gmail.com, bosemeena@gmail.com

Abstract: In some network applications use authentication method to avoid the joining of malicious nodes, nevertheless, some malicious nodes could still deceive the authentication node through Snoop Forge replay attack and other means. As a result, encryption and identity authentication could not completely guarantee the safety of network, and a certain intrusion detection system is still required. Recently researchers have put forward some intrusion detection mechanisms in network, but these mechanisms generally require one or more central node to run complicated testing procedures, which result in large energy consumption and hinders its wide application in network. The result from 2640 application shows that 1) the snoop-forge-replay attacks achieve alarmingly high error rates compared to zero-effort impostor attacks, which have been the de facto standard for evaluating keystroke-based continuous verification systems; 2) four state-of-the-art verification methods, three types of keystroke latencies, and 11 matching-pair settings (-a key parameter in continuous verification with keystrokes) that we examined in this paper were susceptible to the attack; 3) the attack is effective even when as low as 20 to 100 keystrokes were snooped to create forgeries. In light of our results, we question the security offered by current keystroke-based continuous verification systems. We point out that virtualization setup such as the one used in our experiments can also be exploited by an attacker to scale and speed up the attack.

Keywords: Biometrics, continuous verification, keystroke dynamics, snooping, high rate error attacks.

I. Introduction

Encryption system depends largely on the secure system of the key distribution used by the security protocol. Needham and Schroeder (NS) proposed the first important authentication and key distribution protocol in 1978, it happens to be the basis of many authentication protocols. Denning pointed out a flaw of NS protocol in literature in 1981, making people begin to pay attention to the research work in the field of formal security protocol. Replaying attack is a typical issue breach of secured communication between peers that threatens the very design of authentication and key distribution protocols. In this paper, we are studying replaying attack in security protocols, propose anti-replaying attack principles and also propose a new cryptographic protocol corresponding to the principles.

Analyzing the security of such cryptographic protocols is a widely developed research area. However, so far no model has captured the specifics of two-round message exchange protocols. Our work is the first to rigorously specify and analyse protocols for these scenarios. For the analysis, we build upon frameworks developed for analysing cryptographic protocols. We develop a model for analysing our secure authentication protocols, protocols that offer different security verification. This requires us to model features in both frameworks that are usually not included in protocol models, e. g., timestamps and long-term memory, and security verification at each stage.

Although cryptography has been studied for thousands of years, designing secure cryptographic protocols still seems to be stunningly error-prone. The most prominent example is the Needham–Schroeder authentication protocol presented in, which was years later found to be vulnerable to a seemingly obvious attack. During the last decades, the even formal analysis of the security of cryptographic protocols has been a vast research area along with physical network analysis, scalable abstracted reckoning infrastructure that is available on – demand. This model not only saves the IT teams from investing but also protects them from involutions in infrastructure setup and maintenance. Apart from providing the on-demand infrastructure, cloud service provides by and large interfaces for the other related IT management services.

Effective — through a series of experiments conducted using keystroke data from 350 users (150 genuine and
200 impostors), four state-of-the-art continuous verification methods, and templates built with three types of keystroke latencies, we show that the snoop-forgereplay attacks have alarmingly high error rates compared to the error rates of zero-effort1 impostor attacks typically used to evaluate keystroke based continuous verification systems.

Few words become deadly—the attack is surprisingly effective even when a small amount of snooped latencies are used to build forgeries. With 20 characters (few words of text) to 100 characters (less than two lines of texts typed in a typical e-mail textbox) of snooped information, we achieved high error rates against state-of-the-art verification systems. (See Fig. 4 for the error rates of snoop-forge-replay attacks launched with short snooped text).

Legacy keystroke samples remain a threat—because the snoop-forgereplay attack uses forgeries built with stolen latencies of a user, the high attack success rates can seem to be obvious and expected. However, we snoop legacy keystrokes, which are keystrokes of a user captured approximately six months before collecting his/her training (enrollment) samples. Given that behavioral traits such as keystroke latencies have high intrauser variabilities and can change over time, it is interesting to note that our attack achieves high success rates when forgeries are created using legacy keystrokes.

Speed and scalability—Using short stolen samples the attack can be launched quickly as the attacker does not have to wait long to collect victims’ keystrokes. By exploiting virtualization, we show that thousands of forgeries can be generated to simultaneously attack hundreds of users in a short time span. Using a virtualization setup, we created on an average 5594.98 to 299.38 attacks per user in 24 hours.

II. Background

In Fig. 1, we illustrate continuous user verification with keystrokes. Details follow.

Keystroke Latencies: Widely used latencies in the literature are: 1) key hold latency—is the time between press and release of the same key, 2) key press latency—is the time between press of a key and press of the next key, and 3) key interval latency—is the time between the release of a key and press of the next key. We experimented with key hold, key interval, and key press latencies. Template: A template stores the keystroke signatures of a user. We used a 26-by-26 matrix as the template. Each cell corresponds to an English alphabet pair: For example, with key press latencies, if cell — l has , it means that the user (during enrollment) typed thrice with 110 ms, 90 ms, and 100 ms delay between the press of and the press of and the mean delay is 100 ms with 10 ms standard deviation. Unlike key press and interval latencies, a key hold latency by definition is associated with a letter (and not letter pair). Because our template holds only letter pairs, when we used key hold latencies, each cell stored the key hold latencies of the first letter of its letter pair (e.g., cell — l stored key hold latencies of only when the next letter typed is ). Our template is homogeneous, meaning it stores only one type of latencies (i.e., either key hold, interval, or press).

Outlier Detection: Latency values that markedly deviate from majority of the latency values of a user can distort the typing profile of a user, especially if the profile contains statistics sensitive to outliers (e.g., mean). Several studies (e.g., [2], [10], and [11]) performed outlier detection and reported performance gains. In our experiments we used a distance based outlier detection method that worked well in an earlier work [11].

Matching Pairs: Because there are no constraints on what a user types during continuous verification, some keystrokes typed during the verification phase may not have reference signatures in the template. This can happen because the enrollment text used for building the template may not have all the letter pairs present in the 26-by-26 matrix. This problem can be resolved by performing verification using letter pairs that are common to the template and the verification text. Following [1], we refer to these common letter pairs as matching pairs. We use to denote the number of matching pairs. This model involves illustration of user essentiality and cloud service inhibitor. USER: User may include authorized individual or an organization that have data to be stored in the cloud and rely on the cloud for data computational operations.
III. Related Research

A. Nonzero Effort Attacks on Keystroke based User Authentication Systems

To the best of our knowledge, this is the first work to propose nonzero effort impostor attacks against keystroke based continuous user verification. This paper significantly expands our preliminary work in [12]. The key difference between the attacks in [12] and the attacks presented in this paper is that, it is an automated sample-level attack (i.e., a computer program continuously generates key press and release events as if they were being produced by a legitimate user). On the other hand, in [12] we reported a feature level attack. A feature level attack requires the attacker to know what features the verification system is using. Additionally, the attacker has to know how to input the synthesized features directly into the verification algorithm, bypassing keystroke data acquisition, feature extraction, and preprocessing modules. Because the attack in this paper directly submits (fake) samples to the verification system, the attacker does not have to know the internal details of the system. So the attack in this paper is more practical and easier to launch compared to the attack in [12]. There are other important differences between this paper and our work in [12]. They are: 1) experiments with 350 participants (150 genuine and 200 impostors) compared to only 50 in [12], 2) evaluation with four state-of-the-art continuous verification algorithms compared to two in [12], and 3) evaluation with three different types of templates compared to one in [12]. The results and analysis presented in this paper are based on 2640 new attack experiments that are not present in [12]. Some previous studies in fixed-text (i.e., password) based keystroke verification systems used nonzero effort impostor attacks generated by trained human subjects. For example, [13] reported higher impostor pass rates when the impostor subjects were allowed to observe how a genuine user typed his/her password. [14] Conducted experiments to examine how the amount of impostor practice, among other factors, affected the performance of password based keystroke verification systems. [14] Concluded that impostor practice can be a —minorl threat to password based keystroke verification systems.

B. Nonzero Effort Attacks on Other Behavioral Biometric Authentication Systems

Work in this paper was motivated by the findings of two studies: 1) [17] studied the effect of forgery quality on handwriting biometric security and showed that impostor pass rates of trained and generative (i.e., algorithmic) forgery attacks outperformed naive forgeries and 2) [18] evaluated spoofing attacks on gait authentication and showed that attackers with knowledge of their closest person in the database can significantly raise impostor pass rates. Below, we briefly discuss [17], which is closer to our work in this paper. [17] Reported the effect of six types of forgery attack models on handwritten signature based verification. One of them was the generative forgery model, which involved algorithmically generating forgeries of a target writer by collecting a small set of writing samples from: 1) the target writer (these samples were referred as —parallel corpus) and 2) a set of different writers. Results in [17] showed that, compared to trained human forgers, generative attacks had higher impostor pass rates for block and cursive writers but had lower rates for mixed writers. A notable similarity between the generative attacks in [17] and the snoop-forgery-replay attack is that both require a surprisingly low number of stolen samples to generate effective attacks. Ballard et al. [19] present a strategy to counter behavioral forgeries in the context of biometric biometric cryptographic key generation (BKG).

IV. Alarmingly High Rate Error Attacks

A. Snooping Keystroke Timing Information

In this step, the attacker secretly steals a victim’s keystroke timing information. An attacker can snoop a victim’s keystroke timing information using a hardware keylogger or a software keylogger. Software keyloggers have become the most popular forms of keyloggers because they can be easily developed, are easily available, and can be deployed from remote locations onto a victim’s machine (e.g., using trojans and spyware). We used keystroke data collected from 150 participants during the period 13–21 October 2009 as snooped keystrokes (see Table I and Section VI-A for details). This data was collected using a software keylogger developed in C#. The snooped keystrokes were used to attack templates that were built from keystrokes collected approximately six months after the snooped keystrokes.

B. Creating a Keystroke Forgery

In this step, we create a keystroke forgery of a victim user. A forgery has two parts: 1) —dummy text and 2) a series of latencies between the press and release of letters in the dummy text. For example, a forgery of can have the dummy text —this is dummy text. The key hold and interval values for this text come from the snooped keystroke latencies of U.
What if there are Letters in the Dummy Text for Which Snooped Latencies are not Available?: Because our primary goal is to demonstrate how forgeries based on snooped keystrokes can be used to evade detection, when preparing a forgery, we ignored those letters in the dummy text for which corresponding snooped latencies were not available. This sometimes could render the text generated by the forgery linguistically meaningless, especially when forgery is created from limited amounts of snooped text. However, note that current keystroke based continuous verification systems, to the best of our knowledge, do not check the language generated by the typist and therefore, our attack in its present form straightforwardly exploits this vulnerability. If the attacker wants to forge specific words to execute a series of commands, then the attacker can choose to fill the missing latencies with very large values, so that they are filtered by the outlier detection method and thus are disregarded by the continuous verifier. An alternate way is to fill the missing latency values using latencies computed from a population of users (as done in [17] for spoofing handwritten signatures).

C. Replaying a Forgery of Victim

Keystroke Emulator: We developed a keystroke emulator that injects synthetic key press and release events. We programmed the emulator in Visual C++ and used SendInput API. The goal of the emulator is to use the snooped latencies to inject key press and release events for the dummy text in a way that the verifier thinks that it is the victim who is typing the dummy text. The emulator algorithm, referred as —Algorithm 11, gives the steps to forge and replay a victim user’s typing pattern. For each character pair in the dummy text, Algorithm 1 and Procedure 1 generate key press and release events. The time delays between the press and release events are derived from snooped key hold and interval latencies. The variable is used when Algorithm 1 encounters a character pair for which snooped latencies are not available. (Detailed explanation of Algorithm 1 and Procedure 1 is given in the Supplement.) 4) We ran Algorithm 1 for at least 24 hours so that we could generate enough number of forgeries for each user.

VI. Keystroke Data Collection

We used keystroke data collected from 350 participants at Louisiana Tech University. Majority of the participants were students, but university faculty and staff also participated. We used six Pentium IV desktop PCs to collect keystroke data. The PCs were equipped with Windows XP OS, a QWERTY keyboard, and a mouse. On each PC, we installed an interactive keystroke data collection software developed in C#. We collected data during three different periods—October 13–October 21, 2009; April 4–April 30, 2010; and 25 October-9 November, 2010.

We asked the participant to type two types of free text:
1) Copy text—each participant typed several paragraphs of English text from a document provided by us; and
2) Self text—participant had to compose and type text. The participants were allowed to make spelling mistakes, typographical errors and if they chose, could correct them using Backspace or Delete keys.

The keystroke data collection software provided GUI (e.g., text boxes, buttons, and character counters) for typing copy and self texts. Each participant was required to type at least 1800 characters (1200 during October 2009) of copy text. For typing copy text, we provided paper copies of five well known sample texts to the participants. (The list of sample texts is given in the Supplement.) A participant received one of the five sample texts randomly. After entering copy text, the participant was required to type about 300 characters of self text. Self text was collected during April 2010 and October–November 2010 periods. Copy versus self text—Typing self text is a closer representation of a user’s
typing activity. However, we conducted pilot trials in our laboratory before undertaking full-scale data collection and observed that typing 1200–1800 characters of self text took considerably more time than typing copy text of the same length and in most cases fatigued participants. To achieve a trade-off between participation time and obtaining realistic typing samples, we choose to collect a mixture of copy and self texts.

VII. Baseline (Zero-Effort Impostor Attack) Experiments

We experimented with four verifiers: 1) Relative (R) verifier [1], 2) Absolute (A) verifier [1], 3) Similarity (S) verifier [4], and 4) Fusion (F) verifier. We used three types of templates: 1) TKH—template containing key hold latencies, 2) TKI—template containing key interval latencies, and 3) TKP—template containing key press latencies. This resulted in nine verifier template combinations i.e., (R, TKH), (R, TKI), (R, TKP), (A, TKH), (A, TKI), (A, TKP), (S, TKH), (S, TKI), and (S, TKP). The —F— verifier fuses the outputs from (R, TKH), (R, TKI), (S, TKH), (S, TKI), and (A, TKP) using weighted sum fusion rule [25].

Extracting Verification Attempts: From a user’s typing sample, we extracted verification attempts as follows: 1) read the text in the order it was typed and extract latencies until matching pairs are obtained; 2) present the matching pairs to the verifier to obtain a verification score (this constitutes one verification attempt); 3) read the text from the point where it was stopped in Step 2 until matching pairs are obtained; and 4) repeat steps 2 and 3 until the text ends. This procedure partitions the text into contiguous, non-overlapping, variable-length windows, each containing exactly matching pairs. Each window corresponds to one verification attempt. We experimented with values: 20, 40, 60, 80, 100, 120, 150, 300, 350, 500, and 750. Relative (R) and Absolute (A) Verifiers [1]: Given a verification attempt, —RI verifier outputs a score as follows. Two arrays are constructed and it contains the matching pairs ranked in ascending order of their corresponding mean latencies (in the template). It contains the matching pairs ranked in ascending order of their latencies in the verification attempt. The —RI measure between and is computed as the normalized array disorder between and using (1). The —RI measure lies between 0 and 1, 0 (or 1) indicates a perfect match (or mismatch) between the verification attempt and the template.

VIII. References


