Distributed One-Time Keyboard Systems

YoungLok PARK\textsuperscript{1}, Nonmember and MyungKeun YOON\textsuperscript{1(a)}, Member

SUMMARY When attackers compromise a client system, they can steal user input. We propose a distributed one-time keyboard system to prevent information leakage via keyboard typing. We define the problem of secure keyboard arrangement over distributed multi-devices and channels. An analytical model is proposed for the optimal keyboard layout.

key words: keyboard security, network security, multi-channel, Internet banking

1. Introduction

When a computer device at client-side is compromised, attackers can steal whatever users type in through the keyboard. As server-side security has been enhanced with advanced security mechanisms, firewalls, intrusion detection systems, intrusion prevention systems, to mention a few, attackers are targeting client devices, which are easier to compromise[1]. For example, attackers may spread malware to be installed at a client PC or smart-phone. When a client device is infected and completely occupied by attackers, it is almost impossible to prevent them from stealing credential input values typed by users. Although there are commercial solutions to mitigate key logging, expert attackers can evade or even stop the defensive mechanisms after acquiring higher-level system accounts.

We assume that a user types in a credential input value through a keyboard at a client device, for example, PC, smart-phone, tablet, etc., and the input value is delivered to a server over encrypted Internet channels. An Internet banking service is a good target application for our study, but the main idea can be applied to any client devices to which users enter credential information such as a password, account number, social security number, etc. The problem is that current technologies cannot stop attackers from stealing these input values from a user device when the device is completely compromised by them.

In this paper, we propose a novel distributed one-time keyboard system to stop attackers from obtaining input information keyboard-typed by users. We explain the main idea briefly; a user has $n$ client devices, $(n-1)$ for output and one for input. An one-time virtual keyboard is generated at server-side. The keyboard keys are randomly mixed and split into $(n-1)$ pieces, each of which is securely transferred to the corresponding output device. Looking up the $(n-1)$ screens of the output devices, the user can find the correct location of each key. Another user device should play the role of an input device. Even when the input device and one output device are compromised by attackers, they only know the keystrokes whose key values belong to the compromised output device. Unless the attackers compromise all input and output devices, they should not know the complete input values. The enhanced security comes from the fact that the probability of multiple different systems around a user being compromised by an attacker is negligibly small.

Recent studies show that client devices such as PCs and smart-phones are a good target of attackers[1], [2]. Expert hackers can eavesdrop any input value typed by users. When attackers have such special purposes as financial benefits, or political espionage missions, they persistently prepare for the attack for a long time, called an advanced persistent attack[3]. In this attack scenario, user devices can be the first target, and the attackers compromise more critical systems by first obtaining credential information from the users.

2. Distributed One-Time Keyboard System

We first explain the motivation of this research, and the distributed one-time keyboard system is described in detail.

Suppose that a computer, especially a client device, is completely compromised by an attacker. When a key is pressed through a keyboard, the attacker can know its value. Keyboard security solutions, a mitigating method for information leakage, can encrypt the value of a pressed key, or a virtual keyboard can be alternately displayed and the user may select the key by a mouse click[4]. However, expert attackers can turn off the security solutions after promoting their privilege to the level of a system administrator. They can even continuously take screenshots from the monitor. Therefore, it is almost impossible to prevent an attacker from stealing what the user has entered.

We tackle this problem by proposing a Distributed One-Time Keyboard system (DOK) where a random virtual keyboard is generated and the function of user input and output (I/O) is split over multiple user devices, $(n-1)$ output devices and 1 input device. For example, users may have a PC, smart-phone, and tablet PC ($n=3$ in this case). In this paper, we use a smart-phone as an input device, and others as output devices, for the sake of easy explanation. The goal is that attackers cannot steal the input values typed by users even when the input device and one of the output devices
are compromised together. We first show how the DOK is set up, and then explain the secure input process in detail.

We consider an Internet banking scenario for an easy explanation. For simplicity, we assume that the keyboard consists of 10 digits only, and \( n = 3 \). We denote the number of keys as \( m \), therefore in this case \( m = 10 \). In Fig. 1, the user has three devices, a PC, smart-phone, and tablet PC. The smart-phone is the input device, the PC is the first output device, and the tablet PC is the second output device.

We first explain the setup of the DOK. An Internet banking server generates a virtual keyboard at server-side. In Fig. 1, the key sequence in the virtual keyboard is \{1, 9, 0, 4, 6, 5, 8, 2, 7, 3\}, from the top-left to bottom-right. The virtual keyboard is split into 2 pieces because \( n \) is 3. The splitting should be non-overlapping and inclusive. Note that the first piece consists of \{1, null, 0, null, null, 5, 8, 2, null, null\}, and the second piece is \{null, 9, null, 4, 6, null, null, null, 7, 3\}, where null means an empty space. Each piece is transferred to output device 1 and 2, respectively, over secure channels. We assume that communication channels between the server and each user device is encrypted independently, which can be easily implemented with a secure socket layer. A new virtual keyboard should be randomly generated for each new session to prevent replay-attacks.

Now, we explain the secure input process. Suppose that the user input, say a password, is “1234”. The user hits each key in the input device that is placed in the same position as output devices 1 and 2, namely, the PC and tablet PC. The keystroke sequence is “ahlj” at the smart-phone, and this input value is transferred to the server. Because the server knows the correct position of each key, it can derive the correct input value, “1234”.

A general DOK includes \( n \) user devices and \( m \) keys. We denote the input device as \( d_0 \), and the \( i \)th output device as \( d_i \), \( 1 \leq i \leq n - 1 \). The length of an input value is \( k \). We denote the number of keys assigned to \( d_i \) as \( m_i \). The parameters of the DKS in Fig. 1 can be summarized as \( n = 3, m = 10, k = 4, m_1 = 5, \) and \( m_2 = 5 \).

We explain the preparation operations for the DOK, especially how to register output devices and how to link a session with the input and output devices. We recommend to register \( r \) output devices \((r \geq n - 1)\). A DOK user makes a session with a server from an input device, and selects \((n - 1)\) output devices out of \( r \) for the session. During the registration, device-specific information such as an equipment identification number is included for strong authentication. The devices can be replaced, but this is allowed only when DOK is used. Therefore, the attackers cannot register their arbitrary devices until they compromise \((n - 1)\) output devices of the user.

After the registration process, the DOK is ready to be used. The input device first makes a session with a server, which generates a virtual keyboard to be split into \((n - 1)\) pieces. An one-time key is also generated and sent to the input device. The client user enters the key into each of \((n - 1)\) output devices, each of which connects to the server with the one-time key. Now, the server can group \( n \) devices together for the session, and it sends \((n - 1)\) pieces of the virtual keyboard to the registered output devices.

3. Analysis Model and Optimal DOK

We make an analysis model for DOK, especially the probability that an attacker can successfully guess a user input with brute-force attacks. This modeling gives an idea for the optimal assignment of a virtual keyboard.

The input and output device is the same one when \( n = 1 \). For \( n = 2 \), the DOK consists of one input device and only one output device. Therefore, the virtual keyboard is wholly delivered to that output device. We are interested in the case when \( n \) is larger than two because the user has multiple output devices. The security is enhanced because attackers cannot know the whole input value even with the input and one output devices compromised. However, user convenience decreases as \( n \) increases because he/she should look up \((n - 1)\) devices at a time. In this paper, we focus on the case of \( n = 3 \) and \( n = 4 \) because it is a practical setting to pursue both user convenience and security.

We first consider \( n = 3 \). We assume that the probability of each output device being hacked is known a priori, or at least the relative probability among them can be assessed. For example, a private smart-phone with vaccine software would have a half chance of compromise, compared with a public PC in a library. Let \( p_i \) be the probability of output device \( d_i \) being compromised. We assume that the input device is already compromised. This is because attackers cannot steal any information without hacking the input device first. Let \( f(m_1, m_2) \) be the probability that an attacker successfully guesses a user input of length \( k \), then

\[
f(m_1, m_2) = p_1 \times p_2 + p_1 \times (1 - p_2) \times \left( \frac{1}{m_2} \right)^{k-m_2} + p_2 \times (1 - p_1) \times \left( \frac{1}{m_1} \right)^{k-m_1}
\]
where $m = m_1 + m_2$. The first term is when both output devices are compromised, the probability of which is negligibly small and ignored. The second term is when the first output device is compromised, but not output device 2; the attacker knows all the keystrokes displayed at $d_1$, but he/she should guess other keystrokes whose keys are displayed at $d_2$. The average length of user input displayed at $d_2$ is proportion to $m_2/k$ because the virtual keyboard was randomly generated. Therefore, the attackers should guess $m_2/k$ keys, and the probability of successful guessing of one key is $1/m_2$. Therefore, the probability that the attacker successfully guess what the user types in is equal to $(1/m_2)^{m_2/k}$. The third term is equal to the second one, but output device 2 is compromised.

We assume that $p_1$ is small, and therefore the probability of two or more output devices being compromised at the same time is negligibly small. Then, Eq. (1) can be simply presented as follows:

$$f(m_1, m_2) \approx p_1 \times \frac{1}{m_2^{m_2/k}} + p_2 \times \frac{1}{m_1^{m_2/k}}.$$  

(2)

For security enhancement, Eq. (2) should be minimized in terms of $m_1$ and $m_2$. We focus on $m = 10$ and 94 because many input systems still consist of 10 digits, and standard keyboards include 94 keys of upper and lower-case alphabets, numbers, and special characters.

Tables 1 and 2 show the optimal key arrangement for $m = 10$ and $m = 94$, respectively. The row indicates the relative probability of output device 2 being compromised to device 1. The column shows the change of $k$, the length of an user input. The tables show the optimal $(m_1, m_2)$ that minimizes Eq. (2).

Tables 1 and 2 imply that more keys should be assigned to more secure output device, $d_1$, when $p_2/p_1$ increases. For example, when the ratio is 512 with $m = 10$, 9 out of 10 keys should be assigned to $d_1$. However, when $m$ is 94, this bias is relaxed, and the optimal value is (64, 30). The length of an input value also has an impact on the optimal value. When $k$ increases, the bias is also relaxed. An interesting property is that splitting the virtual keyboard over $d_1$ and $d_2$ on a fifty-fifty basis results in quite good optimization when $p_2/p_1$ is small, say less than 10.

For $n = 4$, we can take the same approach as $n = 3$, and then Eq. (1) can be simply presented as follows:

$$f(m_1, m_2, m_3) \approx p_1 \times \frac{1}{m_2^{m_2/k}} + p_2 \times \frac{1}{m_1^{m_2/k}} + p_3 \times \frac{1}{m_1^{m_3/k}} + p_4 \times \frac{1}{m_1^{m_3/k} \times m_2^{m_3/k}}.$$  

(3)

There is a trade-off between security and feasibility in a DOK. When the number of devices increases, the security is enhanced, but the operation time for typing a secret input increases, which degrades user convenience. We implement DOKs of $n = 2$ and $n = 3$, and the time to type secrets of different sizes is measured. The result is shown in Table 4. When the input value is of 10 characters, $k = 10$, the DOK with $n = 2$ takes more than three times than a naive scheme ($k = 1$). We recommend the DOK system for the cases where a high-level secure input is required.

Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2010-0008708 & 2011-0016246).

References

[1] R. Oplinger, R. Rytz, and T. Holderegger, “Internet banking: Client-side attacks and protection mechanisms,” Computer, vol.42, no.6, pp.27–33, 2009.

[2] A. Hilgten, K. Kramp, and T. Weigold, “Secure Internet banking authentication,” IEEE Security & Privacy, vol.4, no.2, pp.21–28, 2006.

[3] A. Juels and T. Yen, “Sherlock Holmes and the case of the advanced persistent threat,” LEET’12, USENIX, 2012.

[4] K. Lee and K. Yim, “Keyboard security: A technological review,” IMIS’11, pp.9–15, 2011.