This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2413410, IEEE Transactions on Dependable and Secure Computing

Password Extraction via Reconstructed Wireless Mouse Trajectory

Xian Pan, Zhen Ling, Aniket Pingley, Wei Yu, *Member,* Nan Zhang, *Member,* Kui Ren, *Senior Member,* and Xinwen Fu, *Member*

Abstract—Logitech made the following statement in 2009: "Since the displacements of a mouse would not give any useful information to a hacker, the mouse reports are not encrypted." In this paper, we prove the exact opposite is true - i.e., it is indeed possible to leak sensitive information such as passwords through the displacements of a Bluetooth mouse. Our results can be easily extended to other wireless mice using different radio links. We begin by presenting multiple ways to sniff unencrypted Bluetooth packets containing raw mouse movement data. We then show that such data may reveal text-based passwords entered by clicking on software keyboards. We propose two attacks, the prediction attack and replay attack, which can reconstruct the on-screen cursor trajectories from sniffed mouse movement data. Two inference strategies are used to discover passwords from cursor trajectories. We conducted a holistic study over all popular operating systems and analyzed how mouse acceleration algorithms and packet losses may affect the reconstruction results. Our real-world experiments demonstrate the severity of privacy leakage from unencrypted Bluetooth mice. We also discuss countermeasures to prevent privacy leakage from wireless mice. To the best of our knowledge, our work is the first to demonstrate privacy leakage from raw mouse data.

Index Terms—Mouse, Trajectory, Password, Sniffing, Privacy, Security

1 Introduction

Logitech made the following statement in a white paper published on March 2, 2009 [1]: "Since the displacements of a mouse would not give any useful information to a hacker, the mouse reports are not encrypted." Wireless mice may use a 27 MHz, a Proprietary 2.4 GHz, or a Bluetooth 2.4 GHz radio link. From our interview with major brand-name manufacturers including Logitech, Microsoft, Apple, and Lenovo and our literature study, no wireless mouse encrypts its communications [35, 36]. This practice is also reflected in the design of mouse communication protocols. The Bluetooth Human Interface Device (HID) profile [8] requires authentication and encryption support for keyboards and other HIDs such as fingerprint scanners, which transmit identification or biometric information [8, 26, 37]. Nonetheless, it does not mandate these security mechanisms for mice.

In this paper, we show mouse movement data leaks extremely sensitive information. The timings and positions of mouse movements are often used as an entropy source for random number and secret generation. Leaked mouse movement data could reduce the entropy of seeding for such random number generation. From a reconstructed mouse trajectory on screen, an attacker may build a user's computer usage profile, identify applications, or even obtain user passwords. This problem is particularly serious given the conventional belief that mouse traffic can be unencrypted, lending users a false sense of security.

In this paper, we focus on privacy leakage from Bluetooth mice and summarize the results of our research for over a year on addressing various challenges related to the problem. Note that our results can be easily extended to mice using other radio links [35, 36] as well. One question often raised when attacking Bluetooth devices is the attack distance. Although Bluetooth is designed as a short-range radio technology, it has been shown that long distance attacks against Bluetooth can be performed from over one mile away [11, 25].

Various off-the-shelf tools are available to sniff Bluetooth mouse communications. In particular, a USRP2 (Universal Software Radio Peripheral 2) device [17], a software-defined radio device, can be tuned to any Bluetooth channel with a 2.48GHz daughterboard. To sniff all Bluetooth channels, four USRP2s are needed. Tools such as Ubertooths [32, 38, 39] can be used to determine the MAC address of undiscoverable devices. This in turn can be fed into an FTS4BT [3], a commercial product, which is able to synchronize with victim Bluetooth devices. An FTS4BT is able to follow Bluetooth frequency hopping sequences, thereby sniffing an entire communication session.

Our major contributions in the paper can be summarized as follows. First, we examine mouse data semantics, investigate how mouse events are processed in an operating system, and propose a *prediction attack*

- *Xian Pan is with SimpliVity Corporation, Westborough, MA 01581. E-mail: panxian1212@gmail.com.*
- *Zhen Ling is with Southeast University, Nanjing, Jiangsu, China, 210018. E-mail: zhenling@seu.edu.cn.*
- *Aniket Pingley is with Intel Corporation, Hillsboro, OR 97124. E-mail: aspingley@gmail.com.*
- *Wei Yu is with Towson University, Towson, MD 21252. E-mail: wyu@towson.edu.*
- *Nan Zhang is with George Washington University, Washington, DC 20052. E-mail: nzhang10@gwu.edu.*
- *Kui Ren is with University at Buffalo (UB), State University of New York (SUNY), Buffalo, New York 14260, USA. E-mail: kuiren@buffalo.edu.*
- *Xinwen Fu is with University of Massachusetts Lowell, Lowell, MA 01854, USA. E-mail: xinwenfu@cs.uml.edu.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2413410, IEEE Transactions on Dependable and Secure Computing

to reconstruct the mouse on-screen cursor trajectory. The main challenge for designing the prediction attack is on the proper understanding of the impact of (i) mouse acceleration algorithms, (ii) packet timing, and (iii) impact of packet losses on the prediction accuracy of the cursor trajectory. To address these challenges, we derive the upper and lower bounds of the complex mouse acceleration for studying reconstruction errors.

Second, we are able to infer critical information from the reconstructed cursor trajectory. Various systems, including Windows, Linux, Mac, and applications [4, 9, 22] provide software keyboards as an alternate input method. Users may *click* a software keyboard and input their credentials. We use the attack against the software-keyboard-based authentication scheme to demonstrate the severity of wireless mouse privacy leakage. We develop two approaches to map a clicking topology to a password sequence. With the *basic inferring* approach, all candidate passwords are enumerated from a clicking topology. With the *enhanced inferring* approach, the statistical information of a human clicking on the region of a key is used to reduce the number of candidate passwords. Our experiments on Fedora 13 and OpenSUSE 11.1 show that the basic inferring approach has a success rate of more than 98% when recovering passwords, while the (much more efficient) enhanced inferring approach has a success rate of more than 95%.

Third, given that mouse acceleration algorithms are often proprietary and cannot always be easily reverse engineered, we propose a *replay attack* for reconstructing the on-screen cursor trajectory if the acceleration algorithm is unknown. In the replay attack, sniffed raw data is replayed on a computer installed with the same operating system as the one on the victim computer. With this approach, we can derive the cursor trajectory and apply the basic and enhanced inferring approaches to derive the password. Our real-world experiments show that the success rate of the replay attack on Fedora 13, Windows 7, and Mac OSX 10.6.5 achieves 69%, **100%**, and 44%, respectively. Please refer to Sections 5.5 for demo videos. The experiment results show that the prediction attack outperforms the replay attack if the acceleration algorithm is known.

The rest of this paper is organized as follows. In Section 2, we discuss how to reconstruct the mouse cursor trajectory. We analyze various factors that affect the accuracy of trajectory reconstruction in Section 3. In Section 4, we evaluate the accuracy of inferring passwords from the sniffed Bluetooth mouse movements using the software keyboard attack as an example. In Section 6, we briefly introduce the most relevant related work. Finally, we conclude this paper in Section 7. Potential countermeasures to the proposed attacks are given in Appendix D in the supplementary document.

2 Reconstruction of Mouse Cursor Trajectory

In this section, we first investigate raw Bluetooth mouse data semantics and then review various mouse cursor acceleration algorithms used in modern operating systems. Finally, we introduce our prediction and replay attacks for reconstructing an on-screen cursor trajectory. For an overview of Bluetooth and a discussion on how to sniff Bluetooth traffic, please refer to Appendix A. Please refer to the supplementary document for all appendices.

2.1 Raw Bluetooth Mouse Data

Although we use a Logitech MX 5500 Bluetooth Mouse as an example in most cases, we actually investigated many other Bluetooth mice and found mice under the same brand share the same semantics. These semantics have been understood through reverse engineering, HCI profile specifications, and related work [16].

For comparison, we briefly discuss the Microsoft Bluetooth Mouse 5000, which has a simple raw packet payload format. The following is an example of its payload: {A1 11 00 **01 FE** 00 00}. The fields in bold provide the X and Y movement, respectively. This data is expressed in two's complement form. Hence, the corresponding movement in this example will be 1 and -2, i.e., a unit movement to the right and two units in the upward direction.

An example of a Logitech MX 5500 mouse raw packet payload is listed as follows: {A1 02 00 **F3 FF FF** 00 00 00}. The three fields in bold are used to compute mouse movement. The following rules are applied to obtain the movement: Let the three fields be X_O (*F3* in the example above), $Y_{O,1}$ (*FF*) and $Y_{O,2}$ (*FF*), respectively. In this case, the reconstruction of mouse movements is more complicated than for the Microsoft Bluetooth Mouse 5000. Specifically, the hexadecimal values $A, \cdots, F$ do not necessarily refer to the decimal $10, \cdots, 15$. Whenever A - F does not represent *10* - *15*, we refer to the hash table in Algorithm 1, which calculates the raw mouse movement for the Logitech mouse. From Algorithm 1, we can see that *F3* on X equals to -(16 - 3) = -13 and *FF* on Y equals to -(16 - 15) = -1.

The raw mouse movement in the raw packet does not actually represent the on-screen cursor movement because the operation system handles such mapping using an acceleration algorithm. Figure 1 shows the Linux input driver stack where *Xserver* conducts the mapping from the raw mouse movement to the on-screen cursor coordinate. In Linux, each hardware component is treated as a special file (i.e., device file). The device file allows user-space applications to interact with the device driver through standard input/output system calls. In the kernel space, the *mousedev* (PS2-emulator) driver creates these device

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2413410, IEEE Transactions on Dependable and Secure Computing

Algorithm 1 Raw Mouse Movement Mapping Algorithm for Logitech Mouse

Require: HASH = (F → 16, E → 32, D → 48, C → 64, B → 80, A → 96);
if (X_O >= 127 in decimal) **then** #Left movement
  X = HASH[first digit of X_O] - second digit of X_O;
else #right movement
  X = X_O;
end if
if (first digit of $Y_{O,2}$ == F) **then** #Up movement
  Y = HASH[second digit of $Y_{O,2}$] - first digit of $Y_{O,1}$;
else #Down movement
  if ($Y_{O,2}$ == 00) **then**
    Y = first digit of $Y_{O,1}$;
  else
    Y = result of concatenating second digit of $Y_{O,2}$ with first digit of $Y_{O,1}$;
  end if
end if

files whereas the *evdev* generic input event driver provides APIs for user-space applications. In the user space, *Xserver* enforces mouse-cursor acceleration by artificially increasing the cursor speed based on how fast a user moves the mouse. For example, consider a raw mouse movement of Δx and Δy pixels on X and Y, respectively. An extremely simple acceleration algorithm may increase the amount of cursor movement by twice the amount (i.e., $(2\Delta x, 2\Delta y)$).

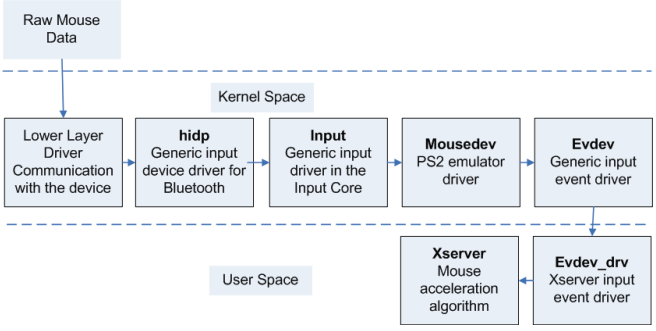


Fig. 1. Linux Input Device Driver Stack

To predict cursor trajectory from sniffed Bluetooth mouse packets, we need to have a precise understanding of mouse acceleration implementation. Mouse acceleration is a feature available in most operating systems today. This feature defines the mapping between the on-screen cursor motion and the physical movement of a mouse. It provides users with the ability to effectively navigate screens with a high resolution and a minimal physical movement of a mouse. Listed below, we derive the Linux mouse acceleration from its source code and examine it in detail as an example. Because we cannot obtain the source code of the Windows and Mac mouse acceleration algorithms, we propose the replay attack to reconstruct the on-screen cursor trajectory with no need of knowing which mouse acceleration algorithm is used.

2.2 Linux Mouse Acceleration

An OS may use an acceleration algorithm to calculate cursor position based on raw mouse movement data. Based on whether packet arrival time is considered in computing the cursor movements on screen, we classify mouse acceleration algorithms into two categories: (i) lightweight acceleration algorithm and (ii) complex acceleration algorithm. The *Lightweight acceleration algorithm* does not consider the packet arrival time and is used in Linux-based OS with Xserver versions before 1.5. The *Complex Acceleration Algorithm* takes the packet arrival time into account and is adopted in Linux-based OS with Xserver versions after 1.5 [6], current Windows OSs, and Mac OS X. We now explain these two types of algorithms in detail.

2.2.1 Lightweight Acceleration Algorithm

Algorithm 2 illustrates the Linux lightweight acceleration algorithm: If a mouse is physically moved more than T units, the algorithm amplifies the movement by M times the current amount along the X and Y axes, respectively, where T and M are pre-determined parameters. It is important to note that T is computed as the *Manhattan* distance (instead of the *Euclidean* distance) of the reported mouse movements. For example, if a mouse reports a movement of $(3, 4)$, the corresponding cursor movement will be $(6, 8)$ when $T = 6$ and $M = 2$ on the X and Y axes, respectively.

Algorithm 2 Lightweight Acceleration Algorithm

Require: Raw mouse movement $(\Delta x, \Delta y)$; Threshold T; Acceleration Factor M
if $(|\Delta x| + |\Delta y| \leq T)$ **then**
  cursor movement = $(\Delta x, \Delta y)$;
else
  cursor movement = $(M \times \Delta x, M \times \Delta y)$;
end if

2.2.2 Complex Acceleration Algorithm

We explain the complex acceleration algorithm based on Linux OSs with Xserver versions after 1.5. When a new mouse event arrives, a mouse event is created for the mouse packet. Next, the system first computes the velocity of the mouse movement and then computes the acceleration based on the derived velocity. Based on the raw movement information in the mouse packet and the derived acceleration, the system determines the cursor movement on screen.

To determine the mouse velocity, we first compute the distance between two mouse events. Denote the sequence of raw mouse events as $Z_1, Z_2, \ldots, Z_n$. A mouse event Z_i includes three elements relative to mouse motion: Δx_i, Δy_i, and timestamp t_i. Denote $D(k, n)$ as the distance between mouse events Z_k and Z_n, where $1 \leq k < n$.

$$D(k,n) = \sqrt{\left(\sum_{i=k}^{n} \Delta x_i\right)^2 + \left(\sum_{i=k}^{n} \Delta y_i\right)^2}. \quad (1)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2413410, IEEE Transactions on Dependable and Secure Computing

Based on the distance $D(k,n)$, we can derive the mouse velocity $V(k,n)$ between Z_k and Z_n as

$$V(k,n) = \frac{D(k,n)}{t_n - t_k} \times \alpha \times \beta, \tag{2}$$

where α and β are *velocity scaling* and *velocity softening* parameters with default values of 10 and 1, respectively. The Linux command *xinput* can be used to return the value of these parameters.

To compute the current mouse velocity V_n (note that V_n is not the same as the velocity $V(k,n)$ between Z_k and Z_n), the system uses a mouse event queue to buffer l mouse events and calculates V_n based on the past mouse events in the queue. Figure 2 shows a mouse event queue with a length of l, having a default value of 16. Denote Z_n as a new mouse event arriving at the queue. We now calculate $V(p,n)$, $V(p+1,n)$, ..., $V(n-1,n)$, and the mouse velocity between mouse event Z_n and those in the queue based on Equation (2), where $n-l+1 \leq p \leq n-1$, $t_n - t_{p-1} > 300ms$, and $t_n - t_p < 300ms$. *It can be observed that mouse events that have occurred 300ms before the current event Z_n do not participate in the calculation of mouse velocity V_n for Z_n.* V_n is derived by the following process: If there is only one mouse event before the mouse event Z_n, the current mouse velocity $V_n = V(n-1,n)$. If there are two mouse events before Z_n, then $V_n = V(n-2,n)$. If there are more than two past mouse events, $V(j,n)$ can be selected as the current mouse velocity V_n by solving the following problem:

$$\begin{aligned} &Maximize: Distance\ D(j,n), \\ &Subject\ to: |V(n-2,n) - V(j,n)| \leq 1 \text{ or}, \\ &\qquad \frac{|V(n-2,n) - V(j,n)|}{V(n-2,n) + V(j,n)} < 0.2, \end{aligned} \tag{3}$$

where $p < j \leq n-1$.

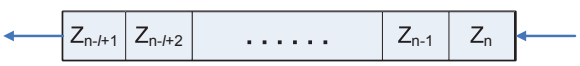


Fig. 2. Mouse Event Queue

When velocities are derived, the acceleration A can be derived as follows:

$$A = \frac{\mathcal{S}(V_n) + \mathcal{S}(V_{n-1}) + 4 * \mathcal{S}(\frac{V_n + V_{n-1}}{2})}{6}, \tag{4}$$

where $S(.)$ is a velocity smoothing function. Because $\mathcal{S}(V_n) \geq 1$, we have $A \geq 1$. Please refer to Appendix B for an explanation of $S(.)$.

Once A is derived, the cursor coordinate (X,Y) on screen can be derived as follows:

$$\begin{aligned} X &= X + A \times \Delta x_n, \\ Y &= Y + A \times \Delta y_n, \end{aligned} \tag{5}$$

where $(\Delta x_n, \Delta y_n)$ is the raw mouse movement. If $A = 1$, the system will not accelerate the mouse speed; otherwise, acceleration is in effect. Note that A can be a decimal number in which Equation (5) will produce a cursor position which is not an integer. The Linux complex acceleration algorithm takes effort in rounding the coordinate and maintaining the residues. Please refer to Appendix C for details of how this approximation is accomplished.

2.3 Reconstructing Cursor Trajectory

Given raw Bluetooth mouse movement data, if an attacker knows the mouse acceleration algorithm used in an operating system, an attacker can predict the cursor trajectory on a victim system. This is denoted as prediction attack and our prediction algorithm comes straight from the Linux driver code. However, the attacker may not know the mouse acceleration algorithm beforehand, particularly if the operating system is proprietary. It is not always trivial to reverse engineer proprietary operating systems and derive the hidden mouse acceleration algorithms.

We propose the *replay attack* if the mouse acceleration algorithm is unknown. A replay attack has two phases. In the first phase, an attacker sniffs raw Bluetooth mouse data between a Bluetooth mouse and a victim computer using the sniffer FTS4BT. In the second phase, to derive the on-screen cursor trajectory, the attacker uses a computer as the attack computer, as shown in Figure 3, and replays the sniffed mouse data to an impersonating computer, which is installed with the same operating system as the victim computer. The cursor trajectory on the impersonating computer is the approximate on-screen cursor trajectory on the victim computer.

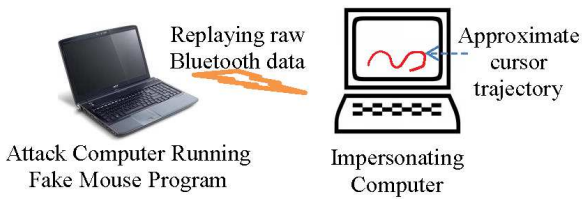


Fig. 3. Replay Attack Illustration

The benefit of our replay attack is that we do not need to understand the complex acceleration algorithm on the victim computer as long as we can discover the operating system running on the victim computer. We can know the type of operating system on the victim computer by using various scanning tools such as *nmap* and *Nessus*.

3 Analysis

In this section, we discuss two main factors that affect the accuracy of reconstructing the mouse cursor trajectory from sniffed raw mouse data: (i) Bluetooth packet loss during sniffing and (ii) the randomness of packet arrival time.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2413410, IEEE Transactions on Dependable and Secure Computing

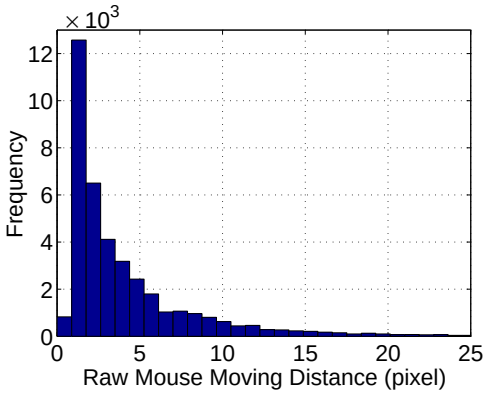


Fig. 4. Histogram of Raw Mouse Moving Distance

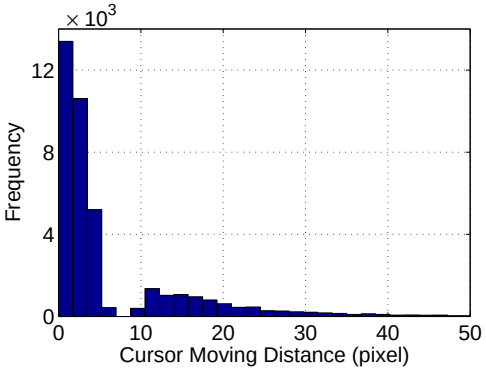


Fig. 5. Histogram of Cursor Moving Distance

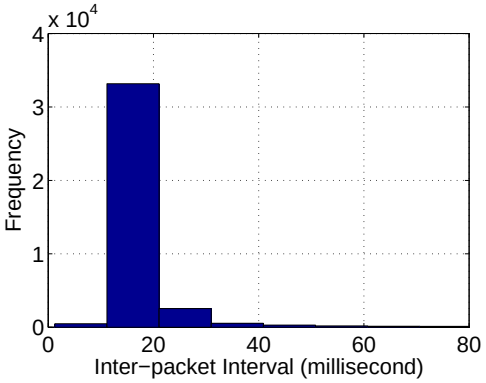


Fig. 6. Histogram of Bluetooth Mouse Inter-packet Interval

3.1 Impact of Bluetooth Packet Loss

A Bluetooth sniffer may miss packets due to various factors of fading or interference such as that from wireless LANs. We designed the following experiments with an FTS4BT device to measure how many pixels may be missing from the reconstructed onscreen cursor trajectory if a Bluetooth packet is lost. Suppose that a user uses a computer with a Bluetooth mouse (Logitech MX 5500) to surf the Internet and play games. At the same time, we use the FTS4BT device to sniff the communications between the mouse and computer for several 40 minute time periods. Each experiment run generates tens of thousands of packets. For example, there are more than 39000 raw mouse packets in one experiment run.

For the lightweight acceleration algorithm, our empirical results in Figure 4 illustrate that the mean value of absolute raw mouse movement distance incurred by a Bluetooth mouse packet is 4.21 pixels. This corresponds to a confidence interval of $[4.16, 4.26]$ with 95% confidence. From Figure 5, which is derived from Figure 4 using Algorithm 2, the mean value of absolute on-screen cursor movement distance is 6.76 pixels. This corresponds to a confidence interval of $[6.64, 6.86]$ with 95% confidence. Hence, under the lightweight acceleration algorithm, we expect an error of around six pixels in the predicted cursor trajectory when we are missing one Bluetooth packet. Lost packets also negatively impact the predicted mouse cursor trajectory for the complex acceleration algorithm. The impact is more complicated as the complex acceleration algorithm considers the timing of arriving packets to compute the mouse acceleration. The loss of a packet affects the computation of mouse movement speed and acceleration. We discuss the impact of timing in the following subsection.

3.2 Impact of Packet Arriving Time

The Bluetooth packet inter-arrival interval as shown in Figure 6 has no effect on an operating system that uses the lightweight acceleration algorithm (Algorithm 2) whereas it affects an operating system using the complex acceleration algorithm. According to the analysis in Section 2.2.2, the estimated current velocity depends on the inter-packet interval in Equation (2) and the historic mouse events in the mouse event queue. The current and previous estimated mouse velocities could affect the acceleration in terms of Equation (4). Eventually, the acceleration determines the ultimate on-screen mouse movement based on Equation (5). Hence, the Bluetooth packet timing and inter-packet interval play important roles in estimating the ultimate mouse movement.

In the prediction attack, packet timestamps recorded during sniffing are not those seen by the victim computer as an event scheduling algorithm adds randomness to timestamps when packets get into the operating system. In the replay attack, we use a high resolution timer to relay the sniffed packets. Similarly, the event scheduling algorithm of the impersonating computer adds randomness to replayed packet timings. However, in the replay attack, since a packet needs to travel through the network stack of the attack computer and the air to arrive at the impersonating computer, more randomness to packet timings may be introduced. This is why the performance of the replay attack is inferior when compared to the prediction attack in our experiments. Hence, in both attacks, we cannot obtain the same packet timestamps seen by the victim computer. The Bluetooth packet arrival

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2413410, IEEE Transactions on Dependable and Secure Computing

time is a factor which could affect the accuracy of reconstructing the mouse cursor trajectory from *sniffed* raw mouse data.

3.2.1 Bound of Complex Acceleration Algorithm

We now derive bounds for acceleration using the Linux complex acceleration algorithm in terms of the mouse velocity. The goal is to understand how the error of predicted mouse velocity, which is caused by packet timing, affects the acceleration and reconstructed cursor trajectory. Consider the system default mouse settings with the simple smooth profile, as discussed in Section 2.2.2 (i.e., the acceleration threshold $h = 4$ and the acceleration factor $a = 2$). Let the current and previous estimated velocity be V_n and V_{n-1}, respectively. The bound of the smoothed mouse velocity $S(V_n)$ is described in the following. A detailed proof can be found in Appendix C.

$$\begin{cases} S(V_n) = 1 \quad , \quad 0 < V_n \leq 4, \\ 1.5 < S(V_n) < 2 \quad , \quad 4 < V_n < 8, \\ S(V_n) = 2 \quad , \quad V_n \geq 8. \end{cases} \tag{6}$$

Based on the bound of $S(V_n)$, we derive the bound of the mouse acceleration A as follows:

$$\begin{cases} A = 1, \ 0 < V_n \leq 4, 0 < V_{n-1} \leq 4, \\ 1.083 < A < 1.167, \ (0 < V_n \leq 4, 4 < V_{n-1} < 8, \text{ or} \\ \qquad 4 < V_n < 8, 0 < V_{n-1} \leq 4), 2 < \frac{V_n + V_{n-1}}{2} < 4, \\ 1.417 < A < 1.703, \ (0 < V_n \leq 4, 4 < V_{n-1} < 8, \text{ or} \\ \qquad 4 < V_n < 8, 0 < V_{n-1} \leq 4), 4 < \frac{V_n + V_{n-1}}{2} < 6, \\ 1.5 < A < 2, \ 4 < V_n < 8, 4 < V_{n-1} < 8, \\ 1.583 < A < 2, (V_n \geq 8, 4 < V_{n-1} < 8), \text{ or} \\ \qquad (4 < V_n < 8, V_{n-1} \geq 8), \\ A = 2, \ V_n \geq 8, V_{n-1} \geq 8, \end{cases} \tag{7}$$

where A has non-continuous subdomains.

The acceleration bound shown in Equation (7) implies that the packet arrival time may affect the acceleration and the cursor trajectory, according to the cursor coordinate derived by Equation (5). Recall that V_n is $V(k, n)$ when Equation (3) is satisfied:

$$V(k, n) = \frac{D(k, n)}{t_n - t_k} \times \alpha \times \beta.$$

When the packet arrival time has a change Δt, the velocity changes to $V'(k, n)$:

$$V'(k, n) = \frac{D(k, n)}{t_n - t_k + \Delta t} \times \alpha \times \beta. \tag{8}$$

Hence, V_n will also change with packet arrival times. Specifically, a small change of timing may switch V_n and V_{n-1} in Equation (7) from one subdomain such as $(0, 4]$ to another subdomain such as $(4, 8]$. For example, if Δt shifts $0 < V_n \leq 4$ and $0 < V_{n-1} \leq 4$ to $4 < V_n < 8$ and $4 < V_{n-1} < 8$, respectively, the acceleration will be changed from $A = 1$ to $1.5 < A < 2$ according to Equations (7). When the coordinates are updated based on Equation (5), the cursor trajectory will be changed.

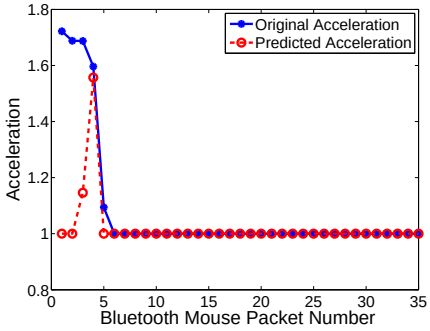


Fig. 7. Predicted Acceleration

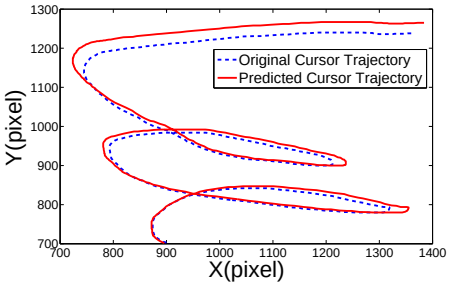


Fig. 8. Predicted Cursor Trajectory

3.2.2 Impact from Packet Arriving Time

Our experiments show the error of cursor trajectory reconstruction caused by the difference of arrival times of Bluetooth packets as seen by the target operation system and the sniffer. We use the sniffer FTS4BT to capture Bluetooth traffic between a Bluetooth mouse (Logitech MX 5500) and a Fedora core 13 computer, which uses the complex acceleration algorithm. Astute readers may question: Since the impact of packet arrival time is being evaluated, what if there is a packet loss during the sniffing phase by the FTS4BT device? Actually, to ensure there is no packet loss, we use a FTS4BT and HCI sniffing software called "hcidump" to sniff packets simultaneously. FTS4BT and hcidump capture the same Bluetooth traffic between the Computer A and the Bluetooth mouse. Note that hcidump runs on Computer A and is able to sniff all packets without loss. We compare the data set from FTS4BT with the data set from hcidump to make sure there is no packet loss in the data set from FTS4BT.

Figures 7 and 8 use the sniffed data set from FTS4BT and show that in the prediction attack, because the predicted acceleration deviates from the original one, the predicted cursor trajectory does not exactly overlap with the original trajectory. In our experiments, the original acceleration values and cursor trajectory are obtained from logs from a revised Linux kernel.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2413410, IEEE Transactions on Dependable and Secure Computing

4 Inferring Passwords from Reconstructed Cursor Topology

In this section, we investigate how the reconstructed cursor trajectory enables an attacker to compromise the sensitive information of a user. In particular, to quantify the results, we consider the scenario of inferring character sequences from a reconstructed cursor clicking topology when a user is clicking an on-screen soft keyboard.

4.1 Inferring Character Sequence

A cursor clicking topology is formed by connecting all points clicked in the reconstructed trajectory. Recall that the reconstruction can be conducted by either the prediction or the replay attack from raw mouse movement data.

We now introduce the *basic approach* to infer a character sequence from a cursor clicking topology. The basic approach directly maps the clicking topology to an on-screen keyboard. Assuming that we have derived the raw mouse data that contains a set of clicks on a soft keyboard, we can derive the clicking topology. However, we do not know the exact starting point of the trajectory and therefore cannot determine which keys have been clicked. To derive all candidates (i.e., all possible character sequences corresponding to the trajectory), we move the cursor clicking topology from the top left to the bottom right in the area of the on-screen keyboard. When the topology moves, the clicking points may produce a character sequence. We record all different character sequences. Consequently, a set of character sequences based on a cursor clicking topology can be derived. We consider the set of character sequences as *candidate character sequences*. The true character sequence must be one of candidates if there is no packet loss and the packet timing is correct. The challenge of this approach is that it may generate a large number of candidates.

To reduce the number of candidate character sequences, we propose an *enhanced inferring approach* which uses the statistical information of the area where the user clicks the on-screen keyboard. Intuitively, when hitting a key, the user tends to click in the center rather than the edge of the area belonging to the key. We define this area as the *hot area* for the key. Because the size of the keys on a soft keyboard is different, in order to derive a normalized hot area, we recruit a group of people and obtain clicking positions of random characters on the same on-screen keyboard and then normalize the rectangular area of a key to a 1×1 square area. The hot area is the area which contains 99% of the clicked positions. After obtaining the hot area, we map a cursor clicking topology to an on-screen keyboard from the top left to the bottom right. A character sequence will be considered as a candidate sequence only if all the characters' clicking positions are in the hot area. With the hot area, the number of candidate character sequences will sharply decrease. The benefit of the enhanced inferring approach is that the uncertainty of the clicked character sequences is significantly reduced.

4.2 Inferring Passwords

To evaluate our method of inferring a character sequence from the reconstructed cursor topology, we conducted extensive experiments. Unless explicitly noted, all of our analysis and figures in the following are derived from the sniffed data by an FTS4BT.

4.2.1 Why the Password Attack is Dangerous

Various systems and applications provide soft keyboards as an alternative input method. Users may "click" these soft keyboards and input sensitive information, which is under the threat of attacks investigated in this paper. We classify these soft keyboards into two categories: (i) the classical soft keyboard and (ii) the randomized soft keyboard. The classical soft keyboard emulates the physical QWERTY keyboard and the randomized soft keyboard has a randomized key layout. The randomization is for defending against other attacks such as the keystroke logging attack, which is different from the attacks investigated in this paper. A randomized keyboard could resist our proposed attack to some extent depending on how the keys are randomized. Our investigation suggests that a purely randomized key layout should be necessary for inputting sensitive information.

To demonstrate that many systems are under the threat of attacks investigated in this paper, we now give a brief summary of systems and applications and the class of soft keyboards. The classic soft keyboard has been widely used by operating systems, including Linux, Windows, Mac., and others. In particular, the known anti-virus software Kaspersky [4] believes that entering confidential data on a virtual keyboard is secure and makes the following statement: "When you enter your confidential data (for example, your login and password in an E-Store) using your keyboard, there is a risk that this personal information is intercepted using the hardware keyboard interceptors or keyloggers, which are programs that register keystrokes. Then, this information will be transferred to hackers/cyber criminals through the Internet. Kaspersky Anti-Virus includes a Virtual keyboard that allows users to avoid the interception of sensitive data." Online banking login systems including HSBC [22] and Westpac [9] use the classical soft keyboard. In contrast, the randomized soft keyboard is used to a very limited extent. Here are two examples: the online login system for State Bank of Travancore in India [5] and an online chat system QQ [7].

Hence, the attack of reconstructing a password clicked on a soft keyboard is truly realistic in various scenarios. The fact that Bluetooth mice leak passwords

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2413410, IEEE Transactions on Dependable and Secure Computing

is significant. To the best of our knowledge, we believe that the aforesaid hidden vulnerability of Bluetooth mice was largely ignored. Hence, we intend to sound a warning bell to the industry that unencrypted communications over Bluetooth mice may be detrimental to user online privacy and security. In Appendix D of the supplementary document, we discuss the encryption of Bluetooth mice and a purely randomized soft keyboard as countermeasures to the investigated attacks.

4.2.2 Performance Metrics

We consider two metrics for evaluating how well we can infer passwords based on the reconstructed clicking topology. One is the success rate, which is defined as the percentage of correctly detected passwords out of all evaluated passwords. Therefore, the success rate is per-password. A password is deemed "correctly detected" if it is in the set of candidate passwords. Recall that one topology may generate a number of candidate passwords. The second metric, *obscurity degree*, measures the average number of passwords corresponding to a single clicking topology. Obviously, an attacker prefers a small number of passwords from a given clicking topology. Assume that each candidate password has an equal probability to be the real password. Hence, if the cardinality of a set is m_i, its entropy is $\log_2 m_i$. The average entropy for all the clicking topologies is then defined as the obscurity degree and is derived by:

$$\text{Obscurity degree} = \frac{\sum_{i=1}^{n} \log_2 m_i}{n}, \tag{9}$$

where n is the number of clicking topologies. Note that the obscurity degree is an information-theoretic metric and a lower obscurity degree means fewer candidate passwords per clicking topology that the attacker has to guess.

In the rest of this paper, we will present the experiment results. For each derived success rate and obscurity degree, we generated 100 random passwords each with a length of 8 characters (including uppercase letters, lowercase letters, and numbers). Then, we used a Bluetooth mouse (Logitech MX 5500) to click on a soft keyboard, xvkbd of size 449×149 pixels (small-size soft keyboard), to input those passwords on a computer installed with different operating systems that use different mouse acceleration algorithms. At the same time, the FTS4BT sniffer was used to sniff all the Bluetooth traffic. To check whether our approach works on soft keyboards with different sizes, we conducted a similar set of experiments on a large size soft keyboard, xvkbd of size 896×254 pixels.

As we know, operating systems usually allow users to configure the mouse acceleration parameters. Our experiments are based on the default setting, which is used by most users. Even if a user changes the default setting, the attack can be feasible. For example, the Windows 7 mouse speed setting has only 11 possible levels. By applying our attack over each possible level individually, we will still be able to reconstruct the correct trajectory. Indeed, if the mouse configuration changes, the attacker has to make more attempts in order to successfully derive the password.

5 Evaluation of Inferring Passwords

To evaluate how well we can infer passwords based on the reconstructed clicking topology, we conducted extensive experiments and attacks were successful on Linux, Windows, and Mac OS X. Both prediction and the replay attacks were deployed on Linux. Because we could not obtain the mouse acceleration source code for Windows and Mac OS X, the replay attack was mainly deployed on these two operating systems.

5.1 Hot Area

To derive the normalized hot area, we generated 120 random passwords each with a length of 8 characters, including uppercase letters, lowercase letters, and numbers. Note that an uppercase letter corresponds to two clicks, *Shift + the letter*. Three persons were recruited and each of them input 40 passwords by clicking on the same software keyboard. We obtained more than 1000 clicking positions. Figure 9 shows the clicking positions on an on-screen keyboard after applying the normalization method. Although the sample size is small, our experimental results clearly demonstrate the severity of the attack presented in this paper.

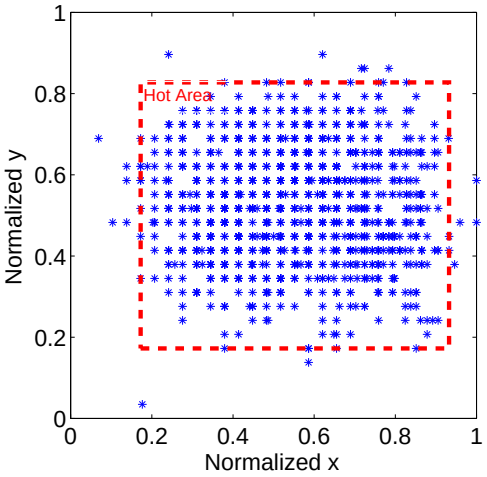


Fig. 9. Normalized Clicking Positions with Hot Area

5.2 Success Rate without Packet Loss in the Prediction Attack

This set of experiments was performed against one person on OpenSUSE 11.1 using the lightweight mouse acceleration algorithm. We evaluate both the basic and enhanced inference approaches for inferring a password on different-sized soft keyboards. For both

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2413410, IEEE Transactions on Dependable and Secure Computing

the small-sized and large-sized soft keyboards, we achieve a success rate of 100% for the basic inferring approach and 99% for the enhanced inferring approach.

We now show that the enhanced inferring approach can significantly reduce the number of candidate passwords for both the small-sized and large-sized soft keyboards. Figures 10 and 11 use the basic inferring approach and show a histogram detailing the number of password candidates for the small and large size soft keyboards, respectively. Figures 12 and 13 show a histogram detailing the number of password candidates from mouse clicking topologies for each of the two sized keyboards when the enhanced inferring approach is used on the hot area. In each of these four figures, the x-axis is the possible quantity of candidate passwords generated by a reconstructed trajectory. The y-axis is the frequency of such a quantity (i.e., how many reconstructed trajectories generate such a quantity of candidate passwords). From these figures, we can observe that the enhanced inferring approach reduces the number of candidate passwords for both small and large size keyboards sharply. In particular, for the small keyboard, the enhanced inferring method reduces the number of candidate passwords from the range of $(0, 425)$ to $(0, 22)$. For the large size keyboard, the enhanced inferring approach reduces the number of candidate passwords from the range of $(0, 400)$ to $(0, 15)$.

From Figures 10 and 11, we can derive the obscurity degree. Table 1 compares the obscurity degree for the basic and enhanced inferring approaches in the scenario where the lightweight acceleration algorithm is used. We can see that the enhanced inferring approach sharply reduces the obscurity of guessing a password. The basic inferring approach has an obscurity degree of around 6 bits whereas the enhanced inferring approach has an obscurity degree of around 1 bit, corresponding to two passwords per clicking topology, which the attacker has to guess.

We performed the prediction attack against another 10 persons on OpenSUSE 11.1 using the lightweight mouse acceleration algorithm. Each person input 10 random passwords of 8 characters long. The hot area in Figure 9, which is derived from clicking positions from three persons, is used for the enhanced inferring approach in this set of experiments. The observations are similar to those from Figures 9, 10, 11 and 12 when the attacks were performed against one person. The attack success rate for both basic inferring and enhanced inferring approaches is still 100%. The enhanced inferring approach can also dramatically reduce the number of candidate passwords. Therefore, the attack strategies presented in this paper are generic. In the rest of the paper, we use attack experiments against one person to demonstrate the severity of the attack.

TABLE 1
Obscurity Degree for Basic and Enhanced Inferring for Lightweight Acceleration

	Small keyboard	Large keyboard
Basic inferring	6.19	5.88
Enhanced inferring	1.70	1.11

5.3 Success Rate with Packet Loss in the Prediction Attack

Recall that during sniffing, Bluetooth packets may be dropped due to fading and interference. To reduce the packet loss rate, we use two FTS4BT dongles placed in redundant mode to sniff the same Piconet. Table 2 lists the packet loss rate in terms of distance between the sniffer and the target. The experiments were conducted in a corridor of a campus building. We can see that the sniffer has a loss rate of only 1.4% at a distance of 10 meters. This shows that the attack can be deployed stealthily from a reasonably long distance. When the distance is more than 10 meters, the loss rate dramatically increases. For a comprehensive evaluation of Bluetooth packet loss caused by various factors, please refer to related bibliography including [19, 28].

TABLE 2
Packet Loss Rate v.s. Distance (meter)

Distance	1	3	5	10	15	30
Loss rate	0	0	0.2%	1.4%	27.1%	97.8%

We now use emulation to show the impact of packet loss on the success rate of inferring passwords because it is not easy to control the loss rate in real-world experiments. The data we will use is from the large size on-screen keyboard on OpenSUSE 11.1. For each loss rate, we first randomly discard raw mouse packets from the original loss-less data set of the FTS4BT device at the specified loss rate. For those packets retained, we will form a new set of raw mouse packets. We then apply either the basic inferring approach or the enhanced inferring approach to the new set of raw mouse packets. In this way, we can derive the success rate given a specific packet loss rate. Figure 14 shows the success rate for the basic inferring approach versus different packet loss rates. We observe that when the packet loss rate is less than 2%, i.e., when the distance is 10 meters or less, the basic inferring approach achieves a very high success rate of around 80%.

Figure 15 shows the success rate for the enhanced inferring approach versus the packet loss rate. The confidence interval for both figures is derived over 10 emulations. When the packet loss rate is less than 1%, the enhanced inferring approach achieves a success rate of near 80%. Comparing Figure 14 with Figure 15, we can observe that when the packet loss rate is less than 1%, the success rate will not sharply decrease for the basic nor the enhanced inferring approaches.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2413410, IEEE Transactions on Dependable and Secure Computing

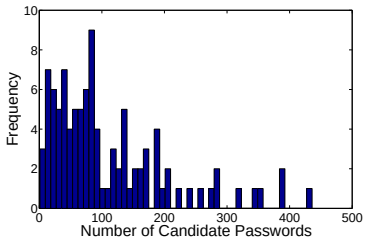


Fig. 10. Histogram of Password Candidates on Small On-screen Keyboard by Basic Inferring

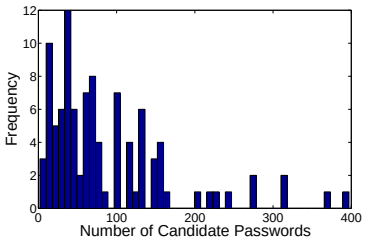


Fig. 11. Histogram of Password Candidates on Large On-screen Keyboard by Basic Inferring

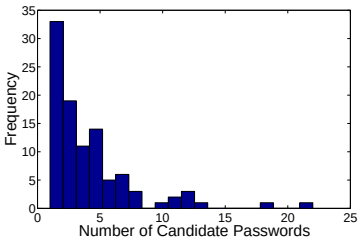


Fig. 12. Histogram of Password Candidates on Small On-screen Keyboard by Enhanced Inferring

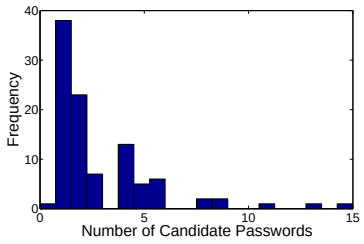


Fig. 13. Histogram of Password Candidates on Large On-screen Keyboard by Enhanced Inferring

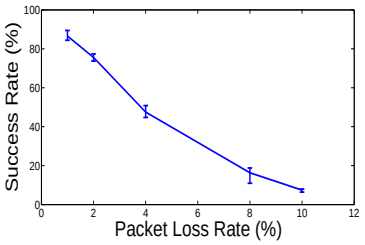


Fig. 14. Success Rate versus Packet Loss Rate by the Basic Approach

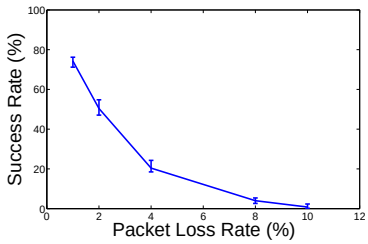


Fig. 15. Success Rate versus Packet Loss Rate by the Enhanced Approach

When the packet loss rate is more than 1%, the basic inferring approach achieves a much higher success rate than the enhanced inferring approach. Hence, the basic inferring approach is adopted when the packet loss rate is more than 1%. Nonetheless, recall that the basic inferring approach has a larger candidate set and therefore a higher uncertainty of guessing the correct password exists. Hence, if the packet loss rate is less than 1%, the enhanced inferring approach can be adopted for lower uncertainty.

5.4 Success Rate with Complex Acceleration in Prediction Attack

As we discussed in Section 3, the packet arrival timing affects the attack accuracy when reconstructing the on screen mouse cursor trajectory for operating systems using the complex acceleration algorithm. We conducted extensive real-world experiments on Fedora Core 13, which uses the complex acceleration algorithm, to investigate how packet timing affects inferring passwords. Note that the data for our investigation is from the FTS4BT sniffer. To reduce the impact from timing, we should use the data starting at the time when the first click for a password occurs as this reduces the prediction error according to the discussion in Section 3.

Table 3 compares the results of inferring passwords for both the lightweight and complex acceleration algorithms. We can see that passwords can be derived with a success rate of more than 95% for the complex acceleration algorithm. One reason for the high success rate is that the mouse movement during password entries (clicking an on-screen keyboard) is different from the mouse movement in other conditions. Each character on the on-screen keyboard corresponds to a small area. Users always take caution when inputting passwords and will not move the mouse too fast in an effort not to miss a key. This slow movement reduces the impact of packet timing on mouse acceleration and favors reconstructing a correct clicking topology. We observed in the experiments for the large size keyboard with the basic inferring approach that 98% of password clicking processes have a topology with a deviation in the range [0, 25] pixels for both the X and Y axes. In only one case, the deviation was 52 pixels in the X direction and 9 pixels in the Y direction. However, a large deviation does not always lead to a failure of password inference as the predicted clicking topology may fall within the characters' areas on the soft keyboard. We have observed similar results in our experiments on the small-sized keyboard.

5.5 Replay Attack

To evaluate the replay attack, we carried out experiments on Fedora Core 13, Windows 7, and Mac OS X 10.6.5. We implemented the raw mouse data replay program, i.e., fake mouse, on a Linux computer installed with Ubuntu 8.04, which could emulate various mouse brands. To guarantee that the replayed packet timing is accurate, we used a high resolution timer (*nanosleep* and real-time clock).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2413410, IEEE Transactions on Dependable and Secure Computing

TABLE 3
Password Reconstruction Success Rate for Lightweight and Complex Acceleration Algorithms

	Basic Inferring		Enhanced Inferring	
	Small keyboard	Large keyboard	Small keyboard	Large keyboard
Lightweight acceleration	100%	100%	99%	99%
Complex acceleration	99%	98%	98%	95%

TABLE 4
Performance of Replay Attack

	Fedora 13		Windows 7		Mac OSX 10.6.5	
	Basic inferring	Enhanced inferring	Basic inferring	Enhanced inferring	Basic inferring	Enhanced inferring
Success rate	69%	31%	100%	92%	44%	16%
Obscurity degree	4.81	0.60	7.41	2.14	6.26	1.13

We now show the results of our replay attack and examine the impact of packet timing changes caused by the replay attack. We first provide results for a victim computer installed with Fedora Core 13. Figures 16 and 17 show that the acceleration and cursor trajectory are changed during reconstruction in the replay attack. Because the acceleration value in the replay attack deviates from the original one, the cursor trajectory derived by the replay attack does not overlap with the original trajectory.

Table 4 shows the success rate and obscurity degree for the replay attack on a large-sized keyboard for 100 passwords. On Fedora 13, we can see that because of a greater impact from replayed Bluetooth packet timing, the performance of the replay attack is inferior when compared to the prediction attack. Even though Bluetooth packet timing is seriously distorted during the replay attack, a detection rate of 69% is still achieved when the basic inferring approach is used. The detection rate for the enhanced inferring approach is 31%. Hence, the basic inferring approach is recommended for the replay attack on a Linux OS with an Xserver version after 1.5.

On Windows 7, we conducted the replay attack using the default soft keyboard. To log the cursor clicking topology, we installed RUI, a tool *Recording User Input* from interfaces for use under Windows and Mac OS X [23], on the impersonating computer. Once a clicking topology is logged, either the basic inferring approach or the enhanced inferring approach can be used to map the clicking topology to the soft keyboard. As we can see from Table 4, the success rate of the basic inferring approach achieves 100% whereas the success rate of the enhanced inferring approach achieves 92% with an obscurity degree of around only 2, corresponding to 4 passwords on average that the attacker must choose to be successful in recovering the password. This demonstrates that the replay attack on Windows 7 is feasible and effective.

On Mac OSX 10.6.5, we conducted the replay attack using the default soft keyboard. RUI is again used to log the cursor clicking topology on the impersonating computer. As we can see from Table 4, the success rate of the basic inferring approach is 44% whereas the success rate of the enhanced inferring approach is 14%. It seems that Mac OSX adopts a more sensitive mouse acceleration algorithm and that the randomness introduced into the packet timing by the replay attack brings more trajectory deviation, leading to a low success rate. Based on our experiments, Mac OSX seems less vulnerable to the replay attack.

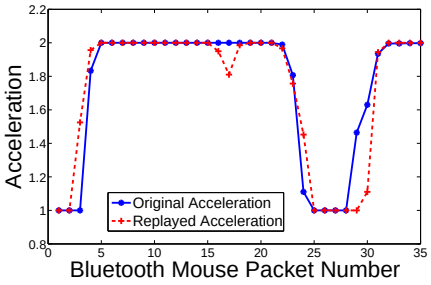


Fig. 16. Acceleration in Replay Attack

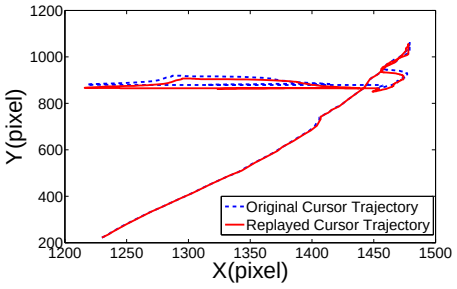


Fig. 17. Cursor Trajectory in Replay Attack

Please see the footnotes for videos of successful replay attacks on different target OSs: Fedora Core 13[1], Windows 7 default installation[2], and Mac OSX 10.6.5[3]. These videos show the replay attack process and do not include the sniffing process. In each demonstration, two computers are used. One emulates the Bluetooth mouse, denoted as "fake mouse," whereas the other computer is the "impersonating computer," installed with the same OS as the victim computer. In the video, the fake mouse is a laptop installed with

[1]Attack Fedora 13: http://youtu.be/qnjqgCCTVTk
[2]Attack Windows 7: http://youtu.be/FVJK_m3UPj0
[3]Attack Mac OSX: http://youtu.be/iFJoHBiYDWg

Ubuntu 8.04 and the impersonating computer is either a laptop or computer. The fake mouse replays sniffed data to the impersonating computer. The sniffed data is derived by the FTS4BT device.

At the beginning of each video, we begin with the mouse device registration and replay programs on the fake mouse. The impersonating computer then connects to the fake mouse. After the Bluetooth connection is set up, the fake mouse will replay the sniffed data packets according to their original timestamps to the impersonating computer. For greater clarity of demonstrating the attack impact, at the beginning of each replay attack, we move the cursor to the first character of the password and show that the replay attack correctly derives the positions of the rest of the password characters. In the video, we can see that the cursor on the target computer moves and clicks passwords automatically. Here, the word "automatically" means the cursor on the target computer is controlled by the fake mouse, rather than by a hand. As we can see, the mouse movement trajectory of the victim and clicking topology can be reconstructed from the cursor movement observed on the impersonating computer.

6 Related Work

Although there are various attacks against Bluetooth, our work is the first on reconstructing the Bluetooth mouse trajectory and deriving sensitive information from it such as passwords. Bluetooth sniffing has been investigated in [2, 32, 39, 40]. Existing attacks include [26, 37] on the pairing procedure for deriving link keys and [14, 31] against Bluetooth keyboards. For a comprehensive study of Bluetooth security and related attacks, please refer to [21, 30]. Please refer to [13] for the study of password strength.

Mouse movement can also be used as a behavioral biometric for the purpose of authenticating a user. For example, Pusara and Brodley [34] used mouse dynamics for conducting re-authentication. Aimed and Traore [10] proposed an approach that aggregates low-level mouse events as higher-level actions, including point-and-clicks or drag-and-drops action. Aimed *et al.*'s work in [29] achieved a very high authentication accuracy from the analysis of 2000 mouse actions. To deploy real-time authentication (such as online re-authentication) based on mouse biometrics, Zheng *et al.* [44] proposed fine-grained angle-based metrics to analyze mouse movement. Based on these metrics, they used the Support Vector Machines (SVM) to classify users. Their results showed that a high accuracy based on few mouse actions could be achieved. Zhao *et al*. studied the process of people choosing gestures from a picture and were able to hack a considerable number of picture passwords in their experiments [43]. Xu *et al*. proposed a cryptographic verification approach to ensure keystroke integrity [41]. Stefan *et al*. investigated the security of keystroke-dynamics authentication against synthetic forgery attacks [42].

7 Conclusion

In this paper, we conducted a holistic investigation of privacy leakage from unencrypted Bluetooth mouse traffic. We examined the Bluetooth mouse packet semantics to develop two attacks, the prediction attack and the replay attack. The two attacks reconstruct on-screen cursor trajectories based on sniffed raw mouse movement data when a lightweight or complex mouse acceleration algorithm is used. We also presented a careful analysis of how packet losses and variations of packet arrival timing may affect the accuracy of reconstructed cursor trajectories. Finally, we performed an extensive evaluation of Bluetooth mouse sniffing on the inference of passwords that a user enters through an on-screen software keyboard. We proposed two approaches for password inference: a basic inferring approach to enumerate all candidate passwords from a clicking topology and an enhanced inferring approach that uses the statistical distribution of human clicking patterns to reduce the number of candidate passwords corresponding to a clicking topology. Our real-world experiments showed the severity of privacy leakage from unencrypted Bluetooth mice. We also discussed potential countermeasures to the proposed attacks in the supplementary document. For future work, we plan to extend our attack to *graphical passwords* such as the picture password in Windows 8.

Acknowledgments

This work was supported in part by the US NSF under Grants 1116644, 1117175, 1350145, 0852674, 0915834, 1117297, 1343976, 1262275 and 1318948, and National Natural Science Foundation of China under grants 61272054, 61402104, and 61320106007, China National High Technology Research and Development Program under Grants No. 2013AA013503, Jiangsu Provincial Key Laboratory of Network and Information Security under grants BM2003201, and Key Laboratory of Computer Network and Information Integration of Ministry of Education of China under grants 93K-9. Any opinions, findings, conclusions, and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] Logitech advanced 2.4 GHz technology, revision 1.1h. http://www.logitech.com/images/pdf/roem/Logitech_Adv_24_Ghz_Whitepaper_BPG2009.pdf, March 2009.

[2] Frontline test system FTS4BT user manual. http://fte.com/docs/FTS4BT%20User%20Guide.pdf, 2010.

[3] FTS4BT Bluetooth protocol analyzer and packet sniffer. http://www.fte.com/products/fts4bt.aspx, 2012.

[4] Kaspersky Internet Security. http://support.kaspersky.com/kis2012/service?page=2&qid=208286483, 2012.

[5] State Bank of Travancore - Virtual Keyboard. https://www.sbtonline.in/sbijava/sbt/virtualkeyboard.html#, 2012.

[6] Pointer acceleration. http://www.x.org/wiki/Development/Documentation/PointerAcceleration, 2013.

[7] QQ International. http://www.imqq.com/, 2014.

[8] Bluetooth human interface device profile. http://www.bluetooth.com, 2015.

[9] Westpac online banking. http://www.westpac.com.au/personal-banking/westpac-online/, 2015.

[10] A. A. E. Ahmed and L. Traore. A new biometric technology based on mouse dynamics. *IEEE Transactions on Dependable and Secure Computing*, 4(3):165–179, 2007.

[11] A. Becker. Bluetooth security & hacks. http://gsyc.es/~anto/ubicuos2/bluetooth_security_and_hacks.pdf, August 2007.

[12] Bluetooth SIG. Specification adopted documents. https://www.bluetooth.org/en-us/specification/adopted-specifications, 2015.

[13] J. Bonneau. Statistical metrics for individual password strength. In *Proceedings of the 20th International Workshop on Security Protocols*, April 2012.

[14] T. Cuthbert, A. Gontarek, E. Jensen, and P. Robbins. A bluetooth keyboard attack. Technical report, University of Minnesota, 2011.

[15] M. Duflot, M. Kwiatkowska, G. Norman, and D. Parker. A formal analysis of bluetooth device discovery. *Journal International Journal on Software Tools for Technology Transfer (STTT)*, 8(6):621 – 632, October 2006.

[16] T. Engdahl. PC mouse information. http://www.epanorama.net/documents/pc/mouse.html, 2014.

[17] M. Ettus. USRP products. http://www.ettus.com/, 2015.

[18] Y. Gelzayd. An alternate connection establishment scheme in the Bluetooth system. Master's thesis, Polytechnic University, 2002.

[19] N. Golmie, R. E. V. Dyck, A. Soltanian, A. Tonnerre, and O. Rébala. Interference evaluation of bluetooth and IEEE 802.11b systems. *Wireless Networks*, 9(3):201–211, 2003.

[20] J. C. Haartsen. The Bluetooth radio system. *IEEE Personal Communications*, 7:28–36, February 2000.

[21] K. Haataja. *Security Threats and Countermeasures in Bluetooth-Enabled Systems*. PhD thesis, University of Kuopio, 2009.

[22] HSBC. Security key demo. http://www.banking.us.hsbc.com/personal/demo/cam/cam_demo.htm, 2012.

[23] U. Kukreja, W. E. Stevenson, and F. E. Ritter. RUI - recording user input from interfaces under Windows and Mac OS X. *Behavior Research Methods*, 38(4), November 2006.

[24] J. P. Lang. Gnu radio. http://gnuradio.org/redmine/projects/gnuradio/wiki, 2015.

[25] A. Laurie, M. Holtmann, and M. Herfurt. Hacking Bluetooth enabled mobile phones and beyond full disclosure. http://trifinite.org/Downloads/21c3_Bluetooth_Hacking.pdf, December 2004.

[26] A. Y. Lindell. Attacks on the pairing protocol of Bluetooth v2.1. In *Proceedings of Black Hat US*, August 2008.

[27] Y. Lindell. Comparison-based key exchange and the security of the numeric comparison mode in Bluetooth v2.1. In *Proceedings of the RSA Conference on Topics in Cryptology*, 2009.

[28] F. Mazzenga, D. Cassioli, P. Loreti, and F. Vatalaro. Evaluation of packet loss probability in bluetooth networks. In *Proceedings of IEEE International Conference on Communications (ICC)*, June 2002.

[29] Y. Nakkabi, L. Traore, and A. A. E. Ahmed. Improving mouse dynamics biometric performancs using varaiance reduction via extractors with separate features. *IEEE Transactions on Systems, Man, and Cybernetics*, 40(6):1345–1353, November 2010.

[30] National Institute of Standards and Technology (NIST). Guide to Bluetooth security. http://csrc.nist.gov/publications/drafts/800-121r1/Draft-SP800-121_Rev1.pdf, September 2011.

[31] M. Ossmann. Bluetooth keyboards: who owns your keystrokes. http://ossmann.com/shmoo-2010/, 2012.

[32] M. Ossmann. Project Ubertooth. http://ubertooth.sourceforge.net, 2012.

[33] M. Ossmann and D. Spill. Building an all-channel bluetooth monitor. ShmooCon - an American hacker convention organized by The Shmoo Group, 2009.

[34] M. Pusara and C. E. Brodley. User re-authentication via mouse movements. In *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*, October 2004.

[35] T. Schroeder and M. Moser. Keykeriki - universal wireless keyboard sniffing for the masses. DeepSec Security Conference, 2009.

[36] T. Schroeder and M. Moser. Practical exploitation of modern wireless devices. CanSecWest, 2010.

[37] Y. Shaked and A. Wool. Cracking the Bluetooth PIN. In *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, June 2005.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2015.2413410, IEEE Transactions on Dependable and Secure Computing

[38] D. Spill. Final report: Implementation of the Bluetooth stack for software defined radio, with a view to sniffing and injecting packets. www.cs.ucl.ac.uk/staff/a.bittau/dom.pdf, May 2007.

[39] D. Spill and A. Bittau. Bluesniff: Eve meets Alice and Bluetooth. In *Proceedings of USENIX Workshop on Offensive Technologies (WOOT)*, August 2007.

[40] D. Spill and M. Ossmann. GR-Bluetooth. http://sourceforge.net/projects/gr-bluetooth/, 2014.

[41] D. Stefan, X. Shu, and D. Yao. Robustness of keystroke-dynamics based biometrics against synthetic forgeries. *Elsevier Computers & Security*, 31:109–121, February 2012.

[42] K. Xu, H. Xiong, C. Wu, D. Stefan, and D. Yao. Data-provenance verification for secure hosts. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 9(2):173–183, March/April 2012.

[43] Z. Zhao, G. J. Ahn, J. J. Seo, and H. Hu. On the security of picture gesture authentication. In *Proceedings of the 22nd USENIX Conference on Security*, August 2013.

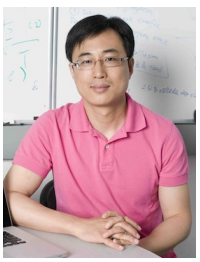
[44] N. Zheng, A. Paloski, and H. Wang. An efficient user verification system via mouse movements. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, October 2011.

Xian Pan is a Senior Engineer at SimpliVity Corporation. She is also an active researcher in computer privacy and security. She received her B.S. degree in Computer Science and M.S. degree in Electrical Engineering from Shaanxi Normal University, China and University of Shanghai for Science and Technology, China respectively. She received her Ph.D. degree in Computer Science from University of Massachusetts Lowell in 2013.

Zhen Ling is a lecturer at the School of Computer Science and Engineering at the Southeast University, Nanjing, China. He received the BS degree (2005) and PhD degree (2014) in Computer Science from Nanjing Institute of Technology, China and Southeast University, China, respectively. He joined Department of Computer Science at the City University of Hong Kong from 2008 to 2009 as a research associate, and then joined Department of Computer Science at the University of Victoria in 2011 as a visiting scholar. His research interests include network security, privacy, and forensics.

Aniket Pingley works a software engineer at Intel Corporation in Oregon since 2011. He received his PhD from The George Washington University, Washington DC, in 2011 under the advisement of Dr. Nan Zhang. His research focus was privacy protection in location-based services.He received Bachelors and Masters degree in Computer Science from Nagpur University, India in 2005 and from the University of Texas at Arlington in 2008, respectively.

Wei Yu received the B.S. degree in electrical engineering from Nanjing University of Technology, Nanjing, China, in 1992, the M.S. degree in electrical engineering from Tongji University, Shanghai, China in 1995, and the Ph.D. degree in computer engineering from Texas A&M University in 2008. He is currently an associate professor with the Department of Computer and Information Sciences, Towson University. He received U.S. National Science Foundation (NSF) CAREER Award in 2014 and the University System of Maryland (USM) Regents Faculty Award for Excellence in Scholarship, Research, or Creative Activity in 2015. His research interests include cyber security, computer networks, and cyber-physical systems.



Kui Ren is currently an associate professor of computer science and engineering at State University of New York at Buffalo. He received his PhD degree from Worcester Polytechnic Institute and BE and ME degrees from Zhejiang University. Kui's research interests include Cloud Security, Wireless Security, Internet of Things, and Mobile Crowdsourcing Systems. His research has been supported by NSF, DoE, AFRL, Amazon, and MSR. He is a recipient of NSF CAREER Award in 2011 and Sigma Xi/IIT Faculty Research Excellence Award in 2012. Kui also received the Best Paper Award from IEEE ICNP 2011. Kui serves as an associate editor for a number of journals including IEEE Transactions on Information Forensics and Security. Kui is also an Area TPC Chair for IEEE INFOCOM 2015 and a Distinguished Lecturer of IEEE VTS. Kui is a senior member of IEEE and a member of ACM.

Nan Zhang received the B.S. degree in computer science from Peking University in 2001 and the PhD degree in computer science from Texas A&M University in 2006. He is an Associate Professor of computer science at The George Washington University. His current research interests include databases and information security/privacy. Nan Zhang received the NSF CAREER Award in 2008. He is a member of the IEEE.

Xinwen Fu is an associate professor in the Department of Computer Science, University of Massachusetts Lowell. He received B.S. (1995) and M.S. (1998) in Electrical Engineering from Xi'an Jiaotong University, China and University of Science and Technology of China respectively. He obtained Ph.D. (2005) in Computer Engineering from Texas A&M University. Dr. Fu's current research interests are in network security and privacy, network forensics, and computer forensics. He received the Best Paper Award from IEEE ICC 2008, 2013 and WASA 2013. He is an editor of IEEE Communications Letters. His research was featured on national and international media including CNN and CCTV. He is a member of the IEEE.