Time to Correct Misunderstanding of Mouse

Xian Pan
UMass Lowell

Zhen Ling
Southeast University

Aniket Pingley
Intel Inc.

Wei Yu
Towson University

Kui Ren
University at Buffalo

Nan Zhang
George Washington University

Xinwen Fu
UMass Lowell

Abstract

Logitech made the following statement in a white paper in 2009: "Since the displacements of a mouse would not give any useful information to a hacker, the mouse reports are not encrypted." It is time to correct this misunderstanding. In this paper, we investigate how sensitive user information leaks from displacements of Bluetooth mouse while our results can be easily extended to mouse using other radio links, which are not encrypted either. We begin with presenting multiple ways to sniff unencrypted Bluetooth packets containing raw mouse movement data. We then show that such data, seemingly harmless, may reveal extremely sensitive information, including text-based passwords clicked through software keyboard and graphical passwords such as Windows 8 picture password. Nonetheless, such a Bluetooth-mouse data leakage attack can be challenging to perform because: (i) packet loss is common for sniffing Bluetooth traffic, and (ii) modern operating systems use complex mouse acceleration algorithms, which introduce noise for reconstructing the on-screen cursor coordinates from sniffed mouse movement data. We have conducted a holistic study of these issues over all popular operating systems and analyze how mouse acceleration algorithms and packet loss during sniffing may affect reconstruction results. Our real-world experiments demonstrate the severity of privacy leakage from un-encrypted Bluetooth mouse. We also discuss countermeasures to prevent privacy leaking from Bluetooth mouse. To the best of our knowledge, our work is the first to retrieve sensitive information from sniffed mouse raw data.

1 Introduction

Logitech made the following statement in a white paper published on March 2, 2009 [30]: "Since the displacements of a mouse would not give any useful information to a hacker, the mouse reports are not encrypted." Wireless mouse may use 27 MHz, Proprietary 2.4 GHz, or Bluetooth 2.4 GHz radio link. From our interview with major brand-name manufacturers including Logitech, Microsoft, Apple and Lenovo and the study of bibliography, no wireless mouse encrypts its communication [37, 38]. This practice is also reflected in the design of mouse communication protocols. Bluetooth Human Interface Device (HID) profile [40] requires support of authentication and encryption for keyboards as well as other HIDs such as fingerprint scanner, which transmits identification or biometric information [40, 28, 39], but leaves the support optional for Bluetooth mouse.

In this paper, we show mouse movement data could leak extremely sensitive information. Timings and positions of mouse movements are often used as an entropy source for random number/secret generation. Leaked mouse movement data could reduce the entropy of seeding random number generation. From a reconstructed mouse trajectory on screen, an attacker may build a user computer usage profile, identify applications, or even obtain user passwords. We will use the inference of passwords through a software keyboard and graphical passwords such as the one used by Windows 8 to demonstrate the threat. This problem is particularly serious given that conventional belief of mouse traffic being insensitive lends users a false sense of security.

We will investigate privacy leaking from Bluetooth mouse while our results can be easily extended to mouse using other radio links [37, 38]. Our attack begins with sniffing Bluetooth mouse communication. Various off-the-shelf tools are available to conduct Bluetooth sniffing. In particular, USRP2 (Universal Software Radio Peripheral 2) [16], 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 [33, 42, 43] can be used to determine the MAC address of undiscoverable devices, which can in turn be fed into FTS4BT [18], a commercial product that is able to synchronize with

victim Bluetooth devices. FTS4BT is able to follow the Bluetooth frequency hopping sequence, and thereby sniff an entire communication session. We shall describe how to use such sniffers to sniff Bluetooth communication. One question often raised for Bluetooth attack is the attack distance. Although Bluetooth is designed as a short-range radio technology, researchers have modified Bluetooth devices and successfully implemented the long distance attack from over one mile away [27, 6]. With a customized antenna for USRP2, we were able to successfully sniff Bluetooth packets at a distance of 30m in the corridor of a campus building.

Once raw mouse data are eavesdropped, we introduce a *trajectory-reconstruction technique*, reconstructing the on-screen mouse cursor trajectory and the topology formed by the positions where mouse clicks, denoted as *clicking topology*. Clicking topology may reveal sensitive information, including the text-based password (inputting through software keyboard) and graphical password such as the one used in Window 8. *To the best of our knowledge, our work is the first to retrieve sensitive information from sniffed mouse raw data*.

Our major contributions is summarized as follows. First, we examine mouse data semantics and investigate how mouse events are processed in an operating system and propose *prediction attack* to reconstruct cursor trajectory. Sniffed mouse packets contain raw movement data. However, an operating system uses acceleration algorithms to accelerate the raw movement and produce the cursor movement on screen. To reconstruct an on-screen cursor trajectory, we carefully investigate various mouse acceleration algorithms, and derive their mathematical models. Once these acceleration algorithms are known, we develop an inference algorithm, denoted as *prediction attack*, for estimating the on-screen cursor trajectory. We analyze the impact of packet loss and the timing of mouse packet arrivals on the accuracy of reconstructing cursor trajectory. Because almost all complex mouse acceleration algorithms take into account the packet inter-arrival interval as a factor in accelerating cursor movement, we found a strong correlation between the accuracy of measuring packet arrival time and the accuracy of reconstructing the cursor trajectory. *This is the most challenging part where we spent a long time on analysis and experiments to reach this conclusion*. We have also derived the upper and lower bounds of the complex mouse acceleration to study reconstruction errors.

Second, by analyzing the reconstructed cursor trajectory, we can infer much information about a user's interaction with the computer. Various systems, including Windows, Linux, Mac, and critical applications [1, 23, 45] provide software keyboard as an alternative input method. Users may *click* the software keyboard and input various sensitive information. We use the attack against the soft-keyboard-based authentication scheme as an example to demonstrate the severity of such privacy leakage. Section 4.2.1 explains the motivation in details and Section 5.1 also extends our attack to *graphical passwords*, a selling security feature in Window 8. We develop two approaches to map a clicking topology to a password sequence entered by a user using the software keyboard. In the *basic inferring* approach, all candidate passwords are enumerated from a clicking topology. In the *enhanced inferring* approach, the statistical information of human clicking keys is utilized to reduce the number of candidate passwords from a clicking topology. The entropy of candidates passwords per clicking topology is reduced from around 6 bits by the basic approach to around 1 bit by the enhanced inferring approach, i.e. two passwords per clicking topology. Our experiments on Fedora 13 and OpenSUSE 11.1 show that the basic inferring approach has a success rate of more than 98% recovering passwords, while the 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 on Windows and Mac systems, we propose *replay attack* for reconstructing on-screen cursor trajectory without the knowledge of acceleration algorithms. In a replay attack, sniffed raw data is replayed on a computer installed with the same operating system as the one on the victim computer. In this way, we can derive the clicking topology and apply either the basic inferring approach or the enhanced inferring approach to derive the password. Our real-world experiments show that the success rate of replay attack against software keyboard on Fedora 13, Windows 7 and Mac OSX 10.6.5 achieves 69%, **100%**, and 44%, respectively. Please see the footnotes for videos of successful replay attacks on different target OS: Fedora Core 13[1], Windows 7 default installation[2], and Mac OSX 10.6.5[3]. In these videos, our program replays real raw mouse data sniffed by FTS4BT. The data corresponds to clicks on a software keyboard. For the clarity of demonstrating the impact of the attack, at the beginning of each replay, we move the cursor to the first character of the password and show that the replay attack can correctly derive the positions of the rest of the password character. Please refer to Section 4.2.6 for a detailed introduction to these videos. In addition, the video at this footnote[4] demonstrates the replay attack against the Windows 8 picture password.

Our contributions also include a discussion of potential countermeasures to the proposed attacks. Bluetooth

[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`

[4] Attack Windows 8 picture password: `http://youtu.be/eLUN8_pDuIE`

has four modes for secure pairing, in which secret keys are negotiated between two pairing devices. We suggest the numerical comparison mode for Bluetooth mouse. As a demo, we have implemented the numerical comparison mode for our raw mouse data replay program, i.e. fake mouse, for an Android tablet. More and more people combine tablets, wireless mouse and keyboard as a mobile computing platform. Microsoft developed a Bluetooth mouse (the wedge mouse) for its Surface tablet. Please refer to the video at the footnote[5]. As a lightweight countermeasure, the software keyboard layout can be randomized to resist the attack when users input sensitive information. Most operating systems and applications do not provide such an option for users. Microsoft also needs to reconsider their choice of graphical password system.

The rest of this paper is organized as follows. In Section 2, we discuss our proposed techniques for reconstructing 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 5, we extend our attack to obtain graphical passwords on Window 8, improve the sniffing distance, and discuss countermeasures to the proposed attacks. In Section 6, we briefly introduce the most related work, followed by the conclusion in Section 7.

2 Reconstruction of Cursor Trajectory

Please refer to Appendix A for the principle of sniffing Bluetooth for raw mouse packets. 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 the prediction attack and the replay attack for reconstructing an on-screen cursor trajectory.

2.1 Raw Bluetooth Mouse Data

In this paper, we use Logitech MX 5500 Bluetooth Mouse as an example most of the time. We investigated many other Bluetooth mouses (e.g., Microsoft Bluetooth Mouse 5000) and found mouse under the same brand tends to have the same semantics. The semantics have been understood by reverse engineering and referring to HCI profile specification and related work, including a general introduction to raw mouse data semantics [15].

For comparison, we briefly discuss 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 give the X and Y movement, respectively. This data is expressed in the two's complement form. Thus the corresponding movement will be 1 and -2, i.e, a unit movement on right and two units in the upward direction.

An example of 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. 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 Microsoft Bluetooth Mouse 5000. Specifically, the hexadecimal values $A, \cdots, F$ do not refer to the decimal $10, \cdots, 15$ necessarily. Whenever *A - F* do not represent *10 - 15*, we would refer to the HASH table in Algorithm 1, which shows the algorithm to calculate the raw mouse movement for 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.

Algorithm 1 Raw Mouse Movement Mapping Algorithm for Logitech Mouse

Require: HASH = (F $\rightarrow$ 16, E $\rightarrow$ 32, D $\rightarrow$ 48, C $\rightarrow$ 64, B $\rightarrow$ 80, A $\rightarrow$ 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

We would like to point out that the raw mouse movement in the raw packet does not actually represent the on-screen cursor movement because the operation system handles such mapping with its 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 is treated as a special file, i.e., device file. The device file allows user-space applications to interact with the device driver via standard input/output system calls. In the kernel space, the *mousedev* (PS2-emulator) driver creates these device files, while the *evdev* generic input event driver provides APIs for user-space applications. In the user space, *Xserver* enforces the mouse-cursor acceleration, which artificially increases 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

[5]Secure mouse: `http://youtu.be/781yYdc-3O8`

movement by twice - i.e., $(2\Delta x, 2\Delta y)$.

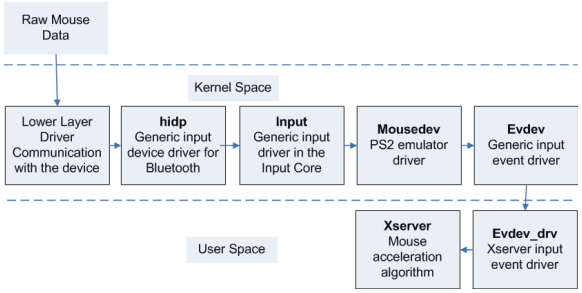


Figure 1: Linux Input Device Driver Stack

To predict the cursor trajectory from the 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 high resolution with minimal physical movement of a mouse. We derive the Linux mouse acceleration from its source code and examine it in detail as an example listed below. Because we cannot obtain the source code of Windows and Mac mouse acceleration algorithms, we will propose the replay attack to reconstruct the on-screen cursor trajectory with no need of understanding the mouse acceleration algorithm being used.

2.2 Linux Mouse Acceleration

An OS may use an acceleration algorithm to calculate the cursor position based on the raw mouse movement data. Based on whether packet arrival time is considered in calculating the cursor movement on screen, we classify mouse acceleration algorithms into two categorizes: (i) lightweight acceleration algorithm, and (ii) complex acceleration algorithm. *Lightweight acceleration algorithm* does not consider the packet arrival time, and it is used in Linux OS with Xserver version before 1.5. *Complex Acceleration Algorithm* takes the packet arrival time into account, and is adopted in Linux OS with Xserver version after 1.5 [11], current Windows and Mac OS X. We now explain these two types of algorithms in details.

2.2.1 Lightweight Acceleration Algorithm

Algorithm 2 illustrates the lightweight acceleration algorithm in Linux: If a mouse is physically moved more than T units, the algorithm amplifies the movement by M times along 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 *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 X and Y axes, respectively.

Algorithm 2 Lightweight Acceleration Algorithm

Require: Raw mouse movement (Δx, Δy); Threshold T; Acceleration Factor M

1: **if** $(|\Delta x| + |\Delta y| \leq T)$ **then**
2: cursor movement = (Δx, Δy);
3: **else**
4: cursor movement = ($M \times \Delta x$, $M \times \Delta y$);
5: **end if**

2.2.2 Complex Acceleration Algorithm

We explain the complex acceleration algorithm based on Linux OS with Xserver version after 1.5. With this algorithm, when a new mouse event arrives and a mouse event is created for the mouse packet, the system first computes the velocity of mouse movement, and then calculates acceleration based on the derived velocity. Based on the raw movement information in the mouse packet and 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: mouse relative 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 $1 \leq k < n$.

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

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

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

where α and β are *velocity scaling* and *velocity softening* parameters with default values as 10 and 1 respectively. Linux command *xinput* returns these parameters.

To compute the current mouse velocity V_n (note that V_n is not 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 length l, whose default value is 16. Denote Z_n as the new mouse event arriving at the queue. We now calculate $V(p,n)$, $V(p+1,n)$, ..., $V(n-1,n)$, 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 happened 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 as follows: 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, $V_n = V(n-2,n)$. If there are more than two past mouse events, $V(j,n)$ could be selected as the current mouse velocity V_n by solving the following problem,

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

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

Figure 2: Mouse Event Queue

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

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

where $S(.)$ is a velocity smoothing function. Because $\mathscr{S}(V_n) \geq 1$, we have $A \geq 1$. Please refer to Appendix B for the 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 and Equation (5) will produce a cursor position that is not an integer. The Linux complex acceleration algorithm takes effort in rounding the coordinate and maintaining the residues. Please refer to Appendix B for details.

2.3 Reconstructing Cursor Trajectory from Raw Mouse Data

Given the raw Bluetooth mouse movement data, if an attacker knows the mouse acceleration algorithm used in an operating system, the attacker can predict the cursor trajectory on the target display of the victim system. However, the attacker may not know the mouse acceleration algorithm before-hand, particularly if the operating system is proprietary. It is not always trivial to reverse engineer those operating systems and derive the hidden mouse acceleration algorithm. Hence, we propose the *replay attack* as well.

The basic idea of the replay attack is to replay the sniffed Bluetooth packets to an impersonating computer, which uses the same OS as the victim computer OS and observes the cursor trajectory on the impersonating computer directly. For example, we can use Computer B to impersonate the victim Bluetooth mouse and connect to the impersonating Computer A. After setting up the connection, the fake mouse, i.e., Computer B will replay the sniffed Bluetooth mouse packets according to their timestamps. Therefore, the cursor movement on Computer A is the reconstructed mouse trajectory that we want. We have implemented the fake mouse on a Linux computer and our fake mouse could emulate various mouse brands. To guarantee that the replayed packet timing is accurate, we use the high resolution timer (*nanosleep* and real time clock) in Linux.

The benefit of replay attack is that we do not need to understand the complex acceleration algorithm on the victim computer if we can impersonate the victim computer in terms of the operating system. 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 various factors that affect the accuracy of reconstructing the mouse cursor trajectory from sniffed raw mouse data. Specifically, we shall focus on two main factors: (i) Bluetooth packet loss during sniffing, and (ii) the randomness of packet arrival time.

3.1 Impact of Bluetooth Packet Loss

Bluetooth sniffer may miss packets due to various fading or interference such as that from wireless LAN. We designed the following experiments with FTS4BT to measure how many pixels may miss from the reconstructed cursor trajectory on screen if a Bluetooth packet is lost. A user is using a computer with a Bluetooth mouse (Logitech MX 5500) for surfing the Internet and playing games. At the same time, we use FTS4BT to sniff the communication between the mouse and computer for 40 minutes. The experiment generates tens of thousands of packets. For example, there are more than 39000 raw mouse packets in one experiment.

For the lightweight acceleration algorithm, our empirical result in Figure 3.left shows that the mean value of absolute raw mouse movement distance incurred by a Bluetooth mouse packet is 4.21 pixels with a confidence interval of $[4.16, 4.26]$ at 95% confidence. From Figure 3.right, which is derived from Figure 3.left using Algorithm 2, the mean value of absolute on-screen cursor movement distance is 6.76 pixels with a confidence interval of $[6.64, 6.86]$ at 95% confidence. Therefore, under the lightweight acceleration algorithm, missing one Bluetooth packet leads to an error of around six pixels in the predicted cursor trajectory. For the complex acceleration algorithm, losing packets deviates the predicted mouse cursor trajectory as well. The impact is more

complicated because 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.

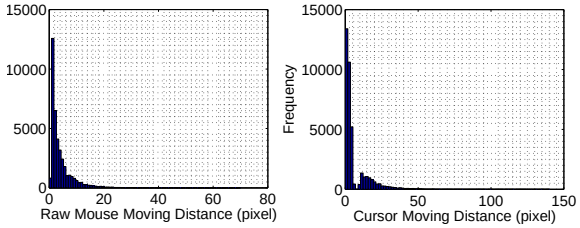


Figure 3: Histogram of Raw Mouse Movement and Cursor Movement

3.2 Impact of Packet Arriving Time

Bluetooth packet inter-arrival interval as shown in Figure 4 has no effect on an operating system that uses the lightweight acceleration algorithm in Algorithm 2, while it affects 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 historic mouse events in the mouse event queue. The current and previous estimated mouse velocity could affect the acceleration in terms of Equation (4). Eventually, the acceleration determines the ultimate on-screen mouse movement based on Equation (5). Therefore, the Bluetooth packet timing and inter-packet interval play an important role in estimating the ultimate mouse movement.

In the prediction attack, packet timestamps recorded during sniffing are not those seen by the victim computer, whose event scheduling algorithm adds randomness into timestamps when packets get into the OS. In the replay attack, we use a high resolution timer to relay the sniffed packets. However, the randomness is added into packet timestamps too when they get into the impersonating computer. There is no guarantee that the impersonating computer behaves the same as the victim computer. The effect of event scheduling on packet timestamps is similar to that in the case of prediction attack. Hence, in both attacks, we cannot obtain the same packet timestamps seen by the victim computer. The Bluetooth packet arrival time is a factor that could affect the accuracy of reconstructing mouse cursor trajectory from *sniffed* raw mouse data in the air.

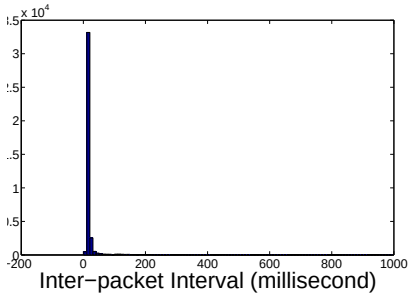


Figure 4: Histogram of Bluetooth Mouse Inter-packet Interval

3.2.1 Bound of Complex Acceleration Algorithm

We now derive bounds of acceleration for the complex acceleration algorithm under Linux in terms of the mouse velocity in order to understand how the error of predicted mouse velocity, caused by packet timing, affects the acceleration, leading to the reconstructed cursor trajectory. Consider the system-default mouse settings with the simple smoothen profile, as discussed in Section 2.2.2 (i.e., $h = 4$ and $a = 2$). 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 as follows. The detailed proof can be found in Appendix C.

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

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 in Equations (7) implies that the packet arrival timing may affect the acceleration, and the cursor trajectory, according to the cursor coordinate calculation 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 timing has a change Δt, the velocity changes to $V'(k,n)$,

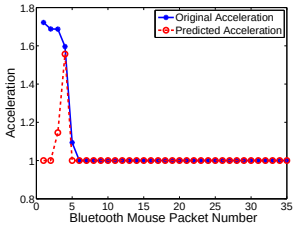
$$V'(k,n) = \frac{D(k,n)}{t_n - t_k + \Delta t} \times \alpha \times \beta. \tag{8}$$

Hence, V_n will change with packet arrival timing as well. 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 is changed.

3.2.2 Impact from Packet Arriving Time

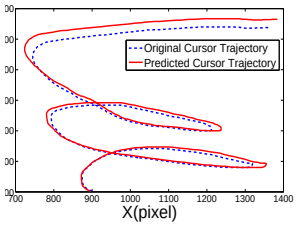
Our experiments demonstrated the error of cursor trajectory reconstruction caused by the difference of arrival timing of Bluetooth packets seen by the target OS and the sniffer. We use the sniffer FTS4BT to capture the Bluetooth traffic between a Bluetooth mouse (Logitech MX 5500) and a Fedora core 13 computer, which adopts the complex acceleration algorithm. Astute readers may question: Since you are evaluating the impact of packet arrival time, what if there is a packet loss during your sniffing by FTS4BT? Actually, to ensure there is no packet loss, we use FTS4BT and a HCI sniffing software called "hcidump" to sniff packets simultaneously. FTS4BT and hcidump capture the same Bluetooth traffic between Computer A and the Bluetooth mouse. Note that hcidump runs on Computer A and is able to sniff all the 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 5 and 6 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.



vspace-4mm

Figure 5: Acceleration in prediction attack



vspace-4mm

Figure 6: Predicted cursor trajectory

4 Evaluation of Reconstructing Cursor Topology by Inferring Passwords

In this section, we evaluate how well the reconstructed cursor trajectory enable an attacker to compromise sensitive information of a user. In particular, to quantify results, we consider the scenario of inferring character sequences from a reconstructed cursor clicking topology when a user is clicking on an on-screen soft keyboard, and 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 the prediction attack and replay attack were deployed against Linux. Because we could not get the mouse acceleration source code for Windows and Mac OS X, the replay attack was mainly deployed against these two operating systems. Although we referred to various materials and gained moderate success with the prediction attack against Windows and Mac OS X, we feel that the replay attack is more general and methodological against these two operating systems.

4.1 Inferring Character Sequence

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

We now introduce the *basic approach* to infer the character sequence from a cursor clicking topology. The basic approach directly maps the clicking topology to an on-screen keyboard. Assume that we have derived the raw mouse data that contain 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 are clicked. To derive all candidates (i.e., all possible character sequences corresponding to the trajectory), we move the cursor clicking topology from top left to 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. Hence, a set of character sequences based on a cursor clicking topology can be derived. We denote 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, an *enhanced inferring approach* is proposed to utilize the statistical information of the area, where people click on the on-screen keyboard. Intuitively, when hitting a key, the user tends to click in the middle region, rather than the edge of the area belonging to the key. We denote this area as the *hot area* for the key. Because the size of keys on the soft keyboard is different, to derive a normalized hot area, we first obtain more than 1000 clicking positions for random characters on the same on-screen keyboard, and then normalize the rectangle area

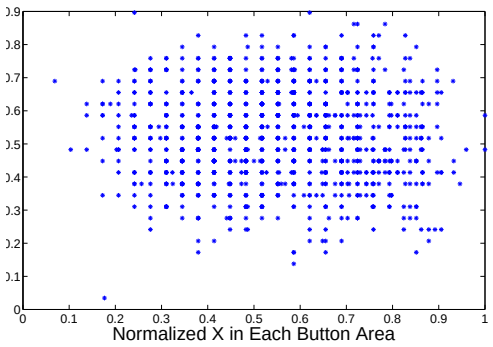


Figure 7: Normalized Clicking Positions on Large On-screen Keyboard

of a key to a 1×1 square area. Figure 7 shows clicking positions for these 1000+ characters on an on-screen keyboard by the normalization method. The hot area is the area that contains 99% of the clicked positions. After obtaining the hot area, we map a cursor clicking topology to an on-screen keyboard from top left to bottom right. A character sequence will be considered as a candidate sequence only if all 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 sequence is significantly reduced.

4.2 Inferring Passwords

To evaluate our method of inferring a character sequence from the reconstructed cursor topology in the prediction attack and replay attack, we conducted extensive experiments. **Please note that all of our analysis and figures in the following are derived from the sniffed data by FTS4BT if not explicitly noted.**

4.2.1 Why the Password Attack is Dangerous

In this paper, we use the example of reconstructing a password clicked on a soft keyboard to demonstrate the privacy leakage from sniffing Bluetooth mouse raw data. We believe that this is an extremely severe threat to user security and a good example that shows a new weakest link of a system, Bluetooth *mouse* communication.

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

To demonstrate many systems are under the threat of attacks proposed in this paper, we now give a brief summary of systems and applications, along with the class of soft keyboards. The classic soft keyboard has been used by various operating systems, including Linux, Windows, Mac., an others. In particular, the well-known anti-virus software Kaspersky [1] 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 Virtual keyboard that allows to avoid interception of sensitive data." Online banking login system including HSBC [23] and Westpac - Australia's First Bank [45] use the classical soft keyboard. 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 [3] and an online chat system QQ [2].

Hence, the attack of reconstructing a password clicked on a soft keyboard is truly realistic in various scenarios. The fact that Bluetooth mouse leaks passwords is significant. We also extended our attack against *graphical passwords* in Section 5.1, which has been adopted by Windows 8. To the best of our knowledge, we believe that the aforesaid hidden vulnerability of Bluetooth mouse was largely ignored. Thus, we intend to sound a warning bell to the industry that unencrypted communication over Bluetooth mouse may be detrimental to user online privacy and security. In Section 5, we discuss the encryption of Bluetooth mouse and a pure randomized soft keyboard as countermeasures to the proposed 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 *success rate*, which is defined as the percentage of real passwords that are included in the set of candidate passwords. The other is *obscurity degree*, which measures the average number of passwords corresponding to a clicking topology. Apparently, an attacker prefers a small number of passwords from a given clicking topology. Assume that each candidate password has the 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 defined as the obscurity degree and is derived by,

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

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

4.2.3 Success Rate without Packet Loss in Prediction Attack

We generated 100 random passwords of 8 characters long (including uppercase letters, lowercase letters, and numbers), and 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 openSUSE 11.1, which uses the lightweight mouse acceleration algorithm. At the same time, the sniffer FTS4BT was used to sniff all the Bluetooth traffic. To check whether our approach works on soft keyboards with different sizes, we conduct the similar set of experiments on a large size soft keyboard, *xvkbd* of size 896×254 pixels.

We evaluate both basic and enhanced inferring approaches for inferring password on different-sized soft keyboards on OpenSUSE 11.1 with the lightweight mouse acceleration algorithm. For both small and large soft keyboards, we achieve a success rate of 100% for basic inferring and 99% for enhanced inferring.

We also evaluate the number of candidate passwords on both small and large soft keyboards and show that the enhanced inferring approach can significantly reduce the number of candidate passwords for both keyboards. Figures 8 and 9 show the histogram of the number of password candidates on the small and large size soft keyboards, respectively through the basic inferring approach. Figures 10 and 11 show the histogram of password candidates from mouse clicking topologies on the two keyboards by the enhanced inferring approach using the hot area. From those figures, we can observe that the enhanced inferring approach sharply reduces the number of candidate passwords for both small and large keyboards. 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 method reduces the number of candidate passwords from the range of $(0,400)$ to $(0,15)$.

From Figures 8 and 9, we can derive obscurity degree. Table 1 compares obscurity degree for basic and enhanced inferring with the lightweight acceleration algorithm. We can see that the enhanced inferring reduces

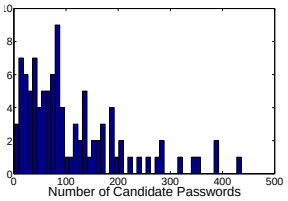


Figure 8: Histogram of password candidates on small on-screen keyboard by basic inferring

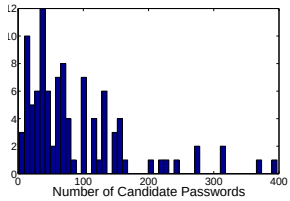


Figure 9: Histogram of password candidates on large on-screen keyboard by basic inferring

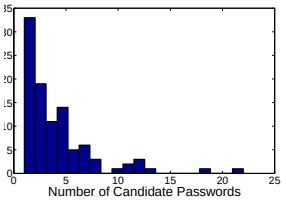


Figure 10: Histogram of password candidates on small on-screen keyboard by enhanced inferring

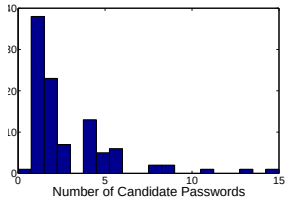


Figure 11: Histogram of password candidates on large on-screen keyboard by enhanced inferring

the obscurity of guessing a password sharply. The basic inferring approach has an obscurity degree of around 6 bits while the enhanced inferring approach has an obscurity degree of around 1 bit, corresponding to two passwords per clicking topology that an attacker has to guess.

Table 1: Obscurity Degree for Basic and Enhanced Inferring for Lightweight Acceleration

	Small keyboard	Large keyboard
Basic inferring	6.1903	5.8845
Enhanced inferring	1.6972	1.1062

4.2.4 Success Rate with Packet Loss in Prediction Attack

Recall that during sniffing, Bluetooth packets may drop due to fading and interference. To reduce the packet loss rate, we use two FTS4BT dongles in the 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 demonstrates that the attack can be deployed stealthily from a reasonable long distance. When the distance is more than 10 meters, the loss increases dramatically. In Section 5,

we will discuss how to use customized devices to further improve the sniffing distance.

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 how packet loss affects success rate of inferring passwords because it is not easy to control loss rate in real-world experiments. The data 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 FTS4BT at a specific loss rate and 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 compute the success rate at the specific packet loss rate. Figure 12 shows the success rate for the basic inferring approach at 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 can achieve a very high success rate of around 80%.

Figure 13 shows the success rate for the enhanced inferring approach at different packet loss rates. The confidence interval for both figures is computed over 10 emulations. When the packet loss rate is less than 1%, the enhanced inferring approach can achieve a success rate near 80%. Comparing Figure 12 with Figure 13, we can see that when the packet loss rate is less than 1%, the success rate will not decrease sharply for the basic and enhanced inferring approaches. When the packet loss rate is more than 1%, the basic inferring approach can achieve 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. Hence, if the packet loss rate is less than 1%, the enhanced inferring can be adopted for a lower uncertainty.

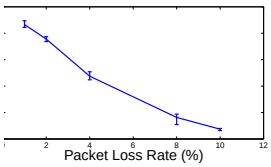


Figure 12: Success rate v.s. packet loss rate by the basic approach

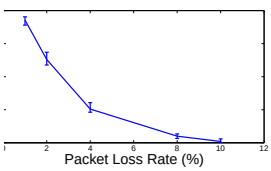


Figure 13: Success rate v.s. packet loss rate by the enhanced approach

4.2.5 Success Rate with Complex Acceleration in Prediction Attack

As we discussed in Section 3, the packet arrival timing affects the attack accuracy on reconstructing the mouse cursor trajectory on screen 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 the packet timing affects inferring passwords. Note that the data for investigating is from the sniffer FTS4BT. To reduce the impact from timing, we should use the data starting at the time when the first click of passwords occurs and this reduces the prediction error according to the discussion in Section 3.

Table 3 compares the results of inferring passwords for 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 entering passwords (clicking an on-screen keyboard) is different from the mouse movement in other situations. 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 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 deviation in the range [0, 25] pixels in both X and Y axes. In only one case, the deviation is 52 pixels on the X direction and 9 pixels in the Y direction. However, the large deviation does not always lead to a failure of password inference, because the predicted clicking topology may be still in the characters' areas on the soft keyboard. We have observed similar results in experiments on the small keyboard.

4.2.6 Replay Attack

To evaluate the replay attack, we conducted the following experiments on Fedora Core 13, Windows 7, and Mac OS X 10.6.5. After sniffing Bluetooth mouse raw data between the Bluetooth mouse and a victim computer by FTS4BT, we used another computer as the attack computer, which was installed Ubuntu 8.04, to replay the sniffed mouse data to an impersonating computer, which is installed with the same OS as the victim computer OS.

We now show the results of replay attack and examine the impact of packet timing change caused by the replay attack. We first provide the result for a victim computer installed with Fedora Core 13. Figures 14 and 15 show that acceleration and cursor trajectory are changed during the reconstruction in the replay attack. Because acceleration in the replay attack deviates from the origi-

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%

nal 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 with a large keyboard for 100 passwords. On Fedora 13, we can see that because of more impact from replayed Bluetooth packet timing, the performance of the replay attack is not as good as the prediction attack. Bluetooth packet timing is seriously distorted during the replay. However, a detection rate of 69% is still achieved when the basic inferring is used. The detection rate for the enhanced inferring is 31%. Hence, the basic inferring is recommended for the replay attack on Linux OS with Xserver version after 1.5.

On Windows 7, we conduct the replay attack on its default soft keyboard. To log the cursor clicking topology, we install RUI, a tool *Recording User Input* from interfaces under Windows and Mac OS X [25], 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 basic inferring approach achieves 100%, while the success rate of enhanced inferring approach reaches 92% with an obscurity degree of only around 2, corresponding to 4 passwords on average for the attacker to choose and be successful in recovering the password. It demonstrates that our replay attack against Windows 7 is feasible and effective.

On Mac OSX 10.6.5, we conduct replay attack on its default soft keyboard. RUI is used to log the cursor clicking topology on the impersonating computer. As we can see from Table 4, the success rate of basic inferring is 44%, while the success rate of enhanced inferring is 14%. It seems that Mac OSX adopts more sensitive mouse acceleration algorithm and 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.

Please see the footnotes for **videos of successful replay attack on different target OS**: Fedora Core 13[6], Windows 7 default installation[7], Mac OSX 10.6.5[8]. These videos show the replay attack process, and do not include the sniffing process. In each demo, two computers are used. One emulates the Bluetooth mouse, denoted as "fake mouse". The other computer is the "impersonating computer", installed with the same OS as the victim computer OS. In the video, the fake mouse is a laptop installed with 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 FTS4BT.

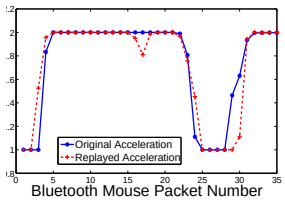


Figure 14: Acceleration in replay attack

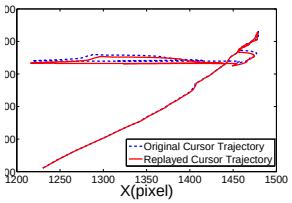


Figure 15: Cursor trajectory in replay attack

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 according to their original time interval to the impersonating computer. For the clarity of demonstrating the attack impact, at the beginning of each replay, 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 a hand. As we can see, the victim's mouse movement trajectory and clicking topology can be reconstructed from the cursor movement on the impersonating computer.

5 Discussion

In this section, we first extend our attack to graphical passwords, and then discuss how to improve the Bluetooth sniffing distance. Finally potential countermeasures are proposed to fight against the proposed attacks.

5.1 Attacking Graphical Passwords

Graphical passwords have attracted great attention as potential alternatives to text-based passwords. Generally

[6]Attack Fedora 13: `http://youtu.be/qnjqgCCTVTk`
[7]Attack Windows 7: `http://youtu.be/FVJK_m3UPj0`
[8]Attack Mac OSX: `http://youtu.be/iFJoHBiYDWg`

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	4.8114	0.5990	7.4084	2.1427	6.2582	1.1304

speaking, graphical passwords can be divided into three categories [7]: (i) recall based, (ii) cued-recall based, and (iii) recognition based techniques. In particular, recall based techniques, including DAS [24], BDAS [14], PassGo [44] and GrIDsure [20], require that users recall and reproduce a drawing or repeat a selection that users create during the personal identification registration phase. In cued-recall systems such as PassPoints [46], users are asked to remember and target specific locations within an image. The image acts as a memory cue to these specific locations selected (clicked) by users. Notice that the difference between the cued-recall and recognition is that: the cued-recall only displays one picture and the user can register a graphical password by choosing different locations in this picture. Hence, the picture itself likes a cue for the user when he/she inputs the graphical password. The recognition will have a bunch of pictures and then the user can pick up some of them as his/her graphical password. The recognition based techniques such as Déjà Vu [10], Story [9] and Passfaces [35] require users to select a set of images during the registration phase and then identify their pre-selected images from a set of decoy images in order to be authenticated.

Our attack proposed in this paper can be applied to various recall based and cued-recall based graphical passwords. Because those recall based and cued-recall based systems take advantage of personal drawing or pre-selected points in an image, the click topology is preserved within the image. If a user uses a Bluetooth mouse as the input device for these graphical passwords, we can capture the user's mouse movement and clicks, and apply either the prediction or the replay attack that we proposed in this paper to recover the passwords. As an example of a commercial recall based graphical password product, GrIDsure [20] presents a user with a 5×5 square grid with 25 cells. During the registration phase, a user chooses a pattern comprising an ordered subset of the 25 cells by *clicking* the corresponding cells as a personal identification pattern. During the login phase, the user is presented with the fully populated grid filled with random numbers in cells. The user input numbers corresponding to her personal identification pattern as a one-time password. In this case, if the user adopts a Bluetooth mouse during the registration phase and the attacker could capture the mouse movement data, it will be trivial to disclose the personal identification pattern and the attacker could login a victim computer as the user after the attack.

The video at the footnote[9] demonstrates the replay attack against Windows 8 graphical password, which is a standard cued-recall based graphical password system. Under Windows 8, a user first chooses a picture and then draws three gestures by using a mouse on PC. The three gestures could be any combination of circles, straight lines, and taps. The video shows the reconstructed cursor trajectory could reveal those gestures and leaks user graphical password effectively. This demonstrates that our proposed attack against Windows 8 is feasible and effective, and Windows 8 should reconsider their choice of graphical password system.

5.2 Potential Countermeasures

We have demonstrated that eavesdropping Bluetooth mouse communication is feasible and may incur serious security and privacy breaches. Hence, we recommend encrypting Bluetooth mouse communication as a potential countermeasure [28, 31].

Bluetooth has four modes for secure pairing in which secret keys are negotiated between two pairing devices: (i) The *numeric comparison* mode is used if both pairing devices have displays. A user accepts the pairing if numbers on both displays are equal. This mode is designed to resist the man-in-the-middle (MITM) attack. (ii) The *just works* mode is designed for devices without displays. It is similar to the numeric comparison mode, but without number comparison and cannot defend against the MITM attack. (iii) The *out of band* mode is used if an extra channel exists between pairing devices. (iv) The *passkey entry* mode is designed for "scenarios where one device has input capability but does not have the capability to display six digits and the other device has output capabilities" [28].

We now discuss which mode is appropriate for encrypting the communication between a mouse and computer. Passkey mode is not appropriate since it is awkward to equip a mouse with a keypad or software keypad. The *out of band* mode cannot be used because there is no additional channel between a mouse and computer. The *just works* mode is subject to the MITM attack. Lindell

[9] Attack Windows 8 picture password: `http://youtu.be/eLUN8_pDuIE`

[29] has proved that the numeric comparison mode for device pairing in Bluetooth version 2.1 (or later) is secure. Arming a mouse with a small display does not look very prohibitive. If such a display shows 6 to 20 digital numbers or characters, the numeric comparison can be applied for Bluetooth mouse to prevent the MITM attack, showing whether there is a MITM or not. 2.6 or higher Linux kernel with Bluez 4.x fully supports Bluetooth secure simple pairing, including the numerical comparison mode. As a demo, we have implemented the numerical comparison mode for our raw mouse data replay program, i.e. fake mouse, for an Android tablet. More and more people combine tablets, wireless mouse and keyboard as a mobile computing platform. Microsoft developed a Bluetooth mouse (the wedge mouse) for its Surface tablet. Please refer to the video at the footnote[10].

We also propose randomization of the key layout of a soft keyboard as a countermeasure to the proposed attacks. Surprisingly, no major operating systems provide a choice of randomized keyboard, neither do most applications. We did find that a few applications use randomized soft keyboard. However, those applications including ones used by the State Bank of Travancore in India [3] and an online chat system QQ [2] often adopt some rules to alternate a limited number of key layouts. For example, the rules may be based on a state machine. This implies that the entered characters are not purely random in terms of on-screen positions. An attacker who is familiar with those rules may still reconstruct the password from sniffed raw mouse data. Hence, the soft keyboard should be completely randomized while users input sensitive information or careful analysis should be performed to study the security of those randomization strategies. We leave such analysis as our future work.

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 such as passwords*. Bluetooth sniffing has been investigated in [43, 12, 33, 17]. Attacks on the pairing procedure for deriving link keys are introduced in [39, 28]. Attacks against Bluetooth keyboard are investigated in [8, 32]. For a comprehensive study of Bluetooth security and related attacks, please refer to [22, 31].

Mouse movement can also be used as behavioral biometrics. Behavioral biometrics, as a biometric authentication technology, has proven useful in authenticating a user. For example, Pusara and Brodley [36] used mouse dynamics for conducting re-authentication. Due to limited experiments with only eleven users, they concluded that mouse biometrics might not be sufficient for user re-authentication. Aimed and Traore [4, 5] 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 [4, 5, 47] achieved 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*. [48] 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.

[10]Secure mouse: `http://youtu.be/781yYdc-3O8`

7 Conclusion

In this paper, we first conducted a holistic investigation of privacy leakage from unencrypted Bluetooth mouse traffic. By reviewing the process of establishing Bluetooth connections, we demonstrated how one can sniff Bluetooth traffic through multiple sniffers or a single sniffer. We then examined the Bluetooth mouse packet semantics and presented the prediction attack and replay attack. The two attacks are able to 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 loss and variations of packet arrival timing may affect the accuracy of reconstructed cursor trajectories. Finally, we performed an extensive evaluation of an application of Bluetooth mouse sniffing - the inference of passwords that a user enters through an on-screen soft keyboard and the inference of graphical passwords used by Window 8. We proposed two approaches for password inference: a basic inferring approach to enumerate all candidate passwords from the clicking topology and an enhanced inferring approach that utilizes 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 mouse.

We also discussed potential countermeasures to the proposed attacks. We recommend the use of numerical comparison mode for encrypting Bluetooth mouse traffic to prevent the man-in-the-middle attack. A randomized software keyboard can also resist the attack against software keyboard while we suggest Microsoft choose a better graphical password system for Windows 8. Our future work includes the development of a full-band Bluetooth sniffer using USRP2s. We also plan to give demos at various technical and academic security conferences and appeal to the Bluetooth/RF mouse manufacturers to encrypt its data and enforce use of more secure device pairing mechanisms.

References

[1] Kaspersky internet security. http://support.kaspersky.com/kis2012/service?page=2&qid=208286483, 2012.

[2] Qq international. http://www.imqq.com/, 2012.

[3] State bank of travancore - virtual keyboard. https://www.sbtonline.in/sbijava/sbt/virtualkeyboard.html#, 2012.

[4] A.A.E.Ahmed and I.Traore. Anomaly intrusion detection based on biometrics. In *Proceedings of the IEEE Workshop on Information Assurance*, 2005.

[5] A.A.E.Ahmed and I.Traore. A new biometric technology based on mouse dynamics. In *IEEE Transactions on Dependable and Secure Computing*, pages 165–179, 2007.

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

[7] R. Biddle, S. Chiasson, and P. V. Oorschot. Graphical passwords: Learning from the first twelve years. *ACM Computing Surveys*, 44(4), 2012.

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

[9] D. Davis, F. Monrose, and M. K. Reiter. On user choice in graphical password schemes. In *Porceedings of the 13th USENIX Security Symposium (Security)*, 2004.

[10] R. Dhamija and A. Perrig. Déjà: A user study using images for authentication. In *Porceedings of the 9th USENIX Security Symposium (Security)*, 2000.

[11] DinoMorelli. Pointer acceleration. http://www.x.org/wiki/Development/Documentation/PointerAcceleration, 2012.

[12] D.Spill and M. Ossmann. Gr-bluetooth. http://sourceforge.net/projects/gr-bluetooth/, 2010.

[13] 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.

[14] P. Dunphy and J. Yan. Do background images improve "draw a secret" graphical passwords? In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS)*, 2007.

[15] T. Engdahl. Pc mouse information. http://www.epanorama.net/documents/pc/mouse.html, 2012.

[16] M. Ettus. Usrp produce. http://www.ettus.com/, 2012.

[17] Frontline Test Equipment, Inc. . Frontline test system fts4bt user manual. http://www.fte.com/docs/fts4bt%20user%20manual.pdf, 2012.

[18] Frontline Test Equipment, Inc. . Fts4bt bluetooth protocol analyzer and packet sniffer. http://www.fte.com/products/fts4bt.aspx, 2012.

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

[20] GrIDsure corporate website. Gridsure. http://gridsure-security.co.uk/, 2012.

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

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

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

[24] I. Jermyn, A. Mayer, F. Monrose, M. K. Reiter, and A. Rubin. The design and analysis of graphical passwords. In *Proceedings of the 8th USENIX Security Symposium (Security)*, 1999.

[25] 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), 2006.

[26] J.-P. Lang. Gnu radio. http://gnuradio.org/redmine/projects/gnuradio/wiki, 2012.

[27] 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.

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

[29] 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.

[30] Logitech. 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.

[31] 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.

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

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

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

[35] Passfaces Corporation. The science behind passfaces. http://www.realuser.com/published/The%20Science%20Behind%20Passfaces.pdf, 2012.

[36] 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*, 2004.

[37] T. Schroeder and M. Moser. Keykeriki: Universal wireless keyboard sniffing for the masses. phneutral7d9, 2009.

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

[39] Y. Shaked and A. Wool. Cracking the bluetooth pin. In *In Proceedings 3rd USENIX/ACM Conf. Mobile Systems, Applications, and Services (MobiSys)*, pages 39–50, 2005.

[40] B. SIG. Bluetooth human interface device profile. http://www.bluetooth.com, 2003.

[41] B. SIG. Adopted bluetooth core specifications. https://www.bluetooth.org/Technical/Specifications/adopted.htm, 2012.

[42] 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.

[43] D. Spill and A. Bittau. Bluesniff: Eve meets alice and bluetooth. In *In Proceedings of USENIX Workshop on Offensive Technologies (WOOT)*, 2007.

[44] H. Tao and C. Adams. Pass-go: A proposal to improve the usability of graphical passwords. *International Journal of Network Security*, 7(2), 2008.

[45] WESTPAC. Westpac online banking. http://www.westpac.com.au/personal-banking/westpac-online/, 2012.

[46] S. Wiedenbeck, J. Waters, J.-C. Birget, A. Brodskiy, and N. Memon. Passpoints: design and longitudinal evaluation of a graphical password system. *International Journal of Human-Computer Studies*, 63(1-2), 2005.

[47] Y.Nakkabi, I.Traore, and A.A.E.Ahmed. Improving mouse dynamics biometric performancs using varaiance reduction via extractors with separate features. In *IEEE Transactions on Systems, Man, and Cybernetics*, pages 1345–1353, 2010.

[48] 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*, 2011.

Appendix A

In this Appendix, we present an overview of Bluetooth and discuss how to sniff Bluetooth traffic.

Introduction to Bluetooth: Bluetooth works in the unlicensed 2.4GHz Industrial Scientific Medical (ISM) band. In USA, Bluetooth divides the ISM band into 79 1MHz-wide channels and uses frequency hopping for communication. For a thorough introduction to Bluetooth, please refer to the core specifications of Bluetooth [41]. In the following, we focus on technical details related to sniffing Bluetooth traffic [19, 21, 41].

Figure 16 shows how a laptop equipped with a Bluetooth adapter communicates with a Bluetooth mouse and forms a Bluetooth network, i.e., *piconet*. Assume that the laptop has never connected with the mouse before. Initially, both the laptop and the mouse are in the state of *standby*, which is a low power mode. In this state, both devices run at their native clocks independently.

To find the mouse and other Bluetooth devices nearby, a user (using an application) commands the laptop Bluetooth adapter to enter the substate of *inquiry* and send out

inquiry messages consciously over the inquiry hopping sequence of channels. The inquiry hopping sequence is determined by the General Inquiry Access Code (GIAC) specified in the standard and known to all devices. It consists of two groups of frequencies: train A and train B, each of which is 16 frequencies long. In Bluetooth, the device that initiates the communication is the master. In our case, the laptop is the master, while the mouse is the slave. To make the mouse discoverable, a user pushes the button on the mouse to have the mouse enter the connecting substate of *inquiry scan*. The Bluetooth specification does not specify how a device leaves the state of standby or connection to perform inquiry. The decision is up to the device manufacturer and implementor.

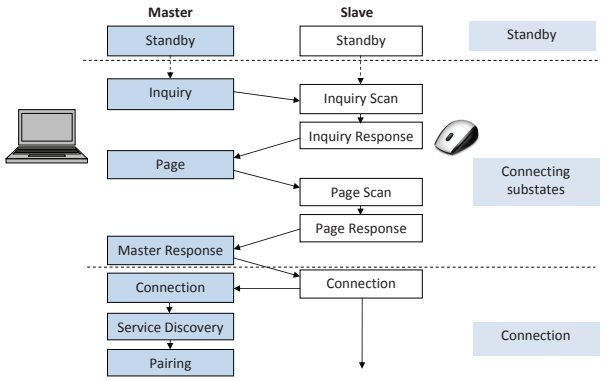


Figure 16: Establishing a Bluetooth Connection

To improve the chance that the mouse receives inquiry messages transmitted at different frequencies, the mouse listens for $T_{w\_inquiry\_scan}$ seconds at one frequency every $T_{inquiry\_scan}$ seconds, where $T_{w\_inquiry\_scan}$ is large enough for receiving inquiry messages transmitted at one train of 16 frequencies. If the inquiry message is not received with the current scan window, the mouse will listen at next frequency, following inquiry scan hopping sequence determined by GIAC as well. Once the inquiry message is received, the mouse gets into the substate of *inquiry response* and sends a Frequency Hopping Synchronization (FHS) packet to the laptop, which is also scheduled to listen for the FHS packet at the same frequency. The FHS packet contains the mouse's MAC address and clock information. Bluetooth designs the inquiry strategy so that at most 10.24 seconds are required for the two devices to find each other [13, 19] if the two devices are close to each other.

When the laptop receives the FHS packet from the mouse, the laptop is ready to build a connection with that particular mouse. At this point, the laptop enters the substate of *page* and runs at the paging-specific frequency hopping sequence, which is computed from the mouse's MAC address. The mouse is in the substate of *page scan*. The procedures of page and page scan are similar to those of inquiry and inquiry scan. The paging procedure is normally shorter than the inquiry procedure since the laptop can estimate the mouse's hopping sequence and phase (where "phase" refers to which frequency the device currently stays at with regard to the hopping sequence) from the FHS packet and has a better chance to catch up with the mouse. The worst case page delay is 2.56 $seconds + r$, where r is a random variable uniformly distributed between 0ms and 10ms. Once the mouse receives the paging packet from the laptop, it enters the substate of *page response* and sends a response packet. When the laptop receives the response, it sends its own FHS packet to the mouse. This FHS packet contains the laptop Bluetooth adapter's MAC address and clock. The mouse then acknowledges the FHS packet.

Once the laptop's FHS packet is acknowledged by the mouse, the laptop and mouse have built the connection and can run upper-layer applications such as *service discovery* and *pairing*. During the state of *connection*, traffic exchange follows the channel hopping sequence, determined by the MAC address of the master device, i.e., the laptop in our example. The master's clock determines the phase in the channel hopping sequence.

Sniffing Bluetooth: To provide privacy, Bluetooth supports optional encryption at the link layer. Bluetooth human interface devices, such as keyboard, mouse, and remote monitoring devices, follow the Bluetooth Human Interface Device (HID) Profile, which defines the protocols, procedures and features. This profile requires support for authentication and encryption for keyboards and other HIDs that transmit identification or biometric information [40]. Encryption is optional for other types of HIDs such as mouse. In many scenarios, Bluetooth mouse traffic is sent without encryption because of three possible reasons: (i) The Bluetooth mouse manufacturer does not encrypt the traffic by default, (ii) people void the encryption for Bluetooth mouse for convenience of use, or (iii) the mouse encryption is often weak [28, 39]. This leaves the chance for an attacker to sniff Bluetooth mouse communication and exploit the mouse cursor information.

One challenge to sniff Bluetooth communication is how to deal with the channel hopping. There are two possible ways to deal with this problem:

1. Sniffing all the 79 frequencies via multiple Bluetooth sniffers. With the advancement of hardware, sniffing the whole ISM band is not impossible. In particular, the Universal Software Radio Peripheral2 (USRP2) [16] is a software-defined radio device and works with the GNU Radio [26], which is a free software toolkit for building Software-defined Radio devices and can be used to demodulate and process Bluetooth packets. A USRP2 with a 2.48GHz daughterboard can be tuned to any Bluetooth channel. One USRP2 can detect 25 channels si-

multaneously. Thus, four USRP2s are enough to sniff all 79 Bluetooth channels. Multiple Ubertooths [33] can be used for sniffing all 79 frequency channels as well [34].

2. Obtaining the hopping sequences of the target piconet. FTS4BT [17] is a commercial Bluetooth sniffer, which uses this approach. To sniff Bluetooth communication between two devices, FTS4BT needs the MAC addresses of both as input, which can be collected through Ubertooth [33] with plugins for Kismet and Wireshark. FTS4BT has a few modes for sniffing. The default mode is *slave inquiry*, in which the sniffer performs an inquiry of the slave device to obtain its Bluetooth clock and enters the page scan mode. The sniffer can then pretend to be the salve as it can use the slave's Bluetooth clock and MAC address to calculate the correct page scan frequencies. Then, when the master pages the slave, the sniffer can switch to the master's Bluetooth clock and follow the master's frequency hopping sequence to capture all Bluetooth packets.

Another challenge to sniffing Bluetooth is that all Bluetooth packets are *whitened* by default [43]. That is, data in the header and payload are scrambled before transmission. The sniffed mouse data must be unwhitened to obtain the original mouse data. The whitening process uses the 6 bits of the clock as input to a linear feedback shift register (LFSR) in order to get a pseudorandom sequence, and then does an XOR of the sequence with the packet data. Fortunately for the attacker, there are only 64 possible starting statuses of LFSR, making it easy to un-whiten a packet in a bruteforce manner. Notice that the Header Error Code (HEC), which is in the packet's header, is also calculated based on the LFSR and initialized with the UAP (upper address part) of the master device, and thus also needs to be un-whitened. Spill *et al*. [43, 42] proposed a mechanism to un-white Bluetooth data, which is used by Ubertooth. FTS4BT emulates the whole process of Bluetooth communication and can un-white the Bluetooth data automatically.

In this paper, we use FTS4BT to conduct sniffing in all experiments. We leave the development of a full band Bluetooth sniffer using USRP2s as our future work.

Appendix B

In this appendix, we introduce how acceleration A in Section 2.2.2 is derived in detail. Before computing mouse acceleration, mouse velocity V_n will be smoothed. Let h and a be the acceleration threshold and acceleration factor, respectively. Default values of h and a are 4 and 2, respectively. a is derived by acceleration numerator divided by acceleration denominator, with default values of 2 and 1, respectively. Notice that the acceleration threshold, numerator and denominator can be set in a configuration file (i.e., /usr/share/X11/xorg.conf.d/10-evdev.conf). In addition, $\mathscr{F}(x)$ in Equation (10) is used to compute the penumbral gradient,

$$\mathscr{F}(x) = 0.5 + \frac{(2x-1)\sqrt{1-(2x-1)^2} + \arcsin(2x-1)}{\pi}. \tag{10}$$

The smoothed mouse velocity $\mathscr{S}(V_n)$ is derived as follows,

$$\mathscr{S}(V_n) = \begin{cases} \mathscr{F}(0.5*(1+V_n))*2-1 & , \quad 0 < V_n < 1, \\ 1 & , \quad 1 \le V_n \le h, \\ 1+\mathscr{F}(\frac{V_n}{ah})*(a-1) & , \quad h < V_n < h*a, \\ a & , \quad V_n \ge h*a, \end{cases} \tag{11}$$

and

$$\text{if } \mathscr{S}(V_n) < 1,\ \mathscr{S}(V_n) = 1. \tag{12}$$

Hence, we know that

$$\mathscr{S}(V_n) \ge 1. \tag{13}$$

Simpson's rule is then used to compute the mouse acceleration A as follows,

$$A = \frac{\mathscr{S}(V_n) + \mathscr{S}(V_{n-1}) + 4*\mathscr{S}(\frac{V_n+V_{n-1}}{2})}{6}. \tag{14}$$

Because $\mathscr{S}(V_n) \ge 1$, we have $A \ge 1$. Let (X, Y) be the current cursor coordinate. **If** $A = 1$, the system will not accelerate the mouse speed. We can derive the cursor coordinate after the raw mouse movement $(\Delta x_n, \Delta y_n)$ as follows,

$$\begin{aligned} X &= X + \Delta x_n, \\ Y &= Y + \Delta y_n. \end{aligned} \tag{15}$$

If $A > 1$, the system will accelerate the mouse speed. Before accelerating the mouse speed, the system first softens the mouse relative motion Δx_i and Δy_i as follows.

$$\Delta x'_n = \begin{cases} \Delta x_n - 0.5 & , \quad \Delta x_n > \Delta x_{n-1}, \\ \Delta x_n + 0.5 & , \quad \Delta x_n < \Delta x_{n-1}, \end{cases} \tag{16}$$

and

$$\Delta y'_n = \begin{cases} \Delta y_n - 0.5 & , \quad \Delta y_n > \Delta y_{n-1}, \\ \Delta y_n + 0.5 & , \quad \Delta y_n < \Delta y_{n-1}. \end{cases} \tag{17}$$

Based on $\Delta x'_n$ and $\Delta y'_n$, we can obtain the accelerated mouse movement as follows,

$$\begin{aligned} \Delta x''_n &= \Delta x'_n * A + \mathscr{R}_x, \\ \Delta y''_n &= \Delta y'_n * A + \mathscr{R}_y, \end{aligned} \tag{18}$$

where $\mathscr{R}_x$ and $\mathscr{R}_y$ are the last remainder of mouse motion.

The system will then update the remainders,

$$\begin{aligned} \mathscr{R}_x &= \Delta x''_n - round(\Delta x''_n), \\ \mathscr{R}_y &= \Delta y''_n - round(\Delta y''_n). \end{aligned} \tag{19}$$

Finally, we can obtain the cursor coordinate on screen

$$\begin{aligned} X &= X + round(\Delta x''_n) \\ Y &= Y + round(\Delta y''_n) \end{aligned} \tag{20}$$

Appendix C

In this Appendix, we provide the detail of computing the upper bound and lower bound of the complex acceleration strategy in Linux. We assume that a user uses the default mouse setting in the original system configuration file. That is, the *Simple Smooth Profile* will be used and the default values of the acceleration threshold h and acceleration factor a are $h = 4$ and $a = 2$, respectively. Based on Equations (11) and (10), we rewrite $S(V_n)$ as follows:

$$S(v) = \begin{cases} 2 \times \frac{V_n\sqrt{1-V_n^2}+arcsinV_n}{\pi}, 0 \le V_n < 1, \\ 1, 1 \le V_n \le 4, \\ 1.5 + \frac{(\frac{V_n}{4}-1)\sqrt{1-(\frac{V_n}{4}-1)^2}+arcsin(\frac{V_n}{4}-1)}{\pi}, 4 < V_n < 8, \\ 2, V_n \ge 8. \end{cases} \tag{21}$$

We now prove the monotonicity of function $S(V_n)$ in each subdomain. From the monotonicity of function $S(V_n)$, we can derive the upper bound and lower bound of $S(V_n)$ in each subdomain.

Case 1. When $0 \le V_n < 1$, from Equation (21), we can derive $S(V_n)'$, the derivative of $S(V_n)$, as follows

$$\begin{aligned} S(V_n)' &= \frac{2}{\pi} * (\sqrt{1-V_n^2} - \frac{V_n^2}{\sqrt{1-V_n^2}} + \frac{1}{\sqrt{1-V_n^2}}), & (22) \\ &= \frac{4 * \sqrt{1-V_n^2}}{\pi}. & (23) \end{aligned}$$

When $0 \le V_n < 1$, we have $S(V_n)' > 0$. That is, $S(V_n)$ is monotonically increasing. The bound is

$$0 < S(V_n) < 1, \text{ when } 0 < V_n < 1. \tag{24}$$

Case 2. When $4 < V_n < 8$, from (21), we have

$$\begin{aligned} S(V_n)' &= \frac{1}{\pi} * (\frac{\sqrt{1-(\frac{V_n}{4}-1)^2}}{4} - \frac{(\frac{V_n}{4}-1)^2}{4 * \sqrt{1-(\frac{V_n}{4}-1)^2}} \\ &\quad + \frac{1}{4 * \sqrt{1-(\frac{V_n}{4}-1)^2}}), & (25) \\ &= \frac{1}{\pi} * \frac{1-(\frac{V_n}{4}-1)^2}{2 * \sqrt{1-(\frac{V_n}{4}-1)^2}}, & (26) \\ &= \frac{\sqrt{1-(\frac{V_n}{4}-1)^2}}{2\pi}. & (27) \end{aligned}$$

Hence, $S(V_n)' > 0$ and $S(V_n)$ is monotonically increasing, when $4 < V_n < 8$. We have the bound

$$1.5 < S(V_n) < 2, \text{ when } 4 < V_n < 8. \tag{28}$$

Case 3. When $1 \le V_n \le 4$, $S(V_n)$ is a constant. We have

$$S(V_n) = 1, \text{ when } 1 \le V_n \le 4. \tag{29}$$

Case 4. Similar to Case 3, when $V_n \ge 8$, $S(V_n)$ is a constant. Then we have

$$S(V_n) = 2, \text{ when } V_n \ge 8. \tag{30}$$

Combining Equations (24), (28), (29), and (30), we can derive the range of $S(V_n)$

$$\begin{cases} 0 < S(V_n) < 1 & , \quad 0 < V_n < 1, \\ S(V_n) = 1 & , \quad 1 \le V_n \le 4, \\ 1.5 < S(V_n) < 2 & , \quad 4 < V_n < 8, \\ S(V_n) = 2 & , \quad V_n \ge 8. \end{cases} \tag{31}$$

According to Equations (12) and (13), we can simplify the range of $S(V_n)$ as follows,

$$\begin{cases} S(V_n) = 1 & , \quad 0 < V_n \le 4, \\ 1.5 < S(V_n) < 2 & , \quad 4 < V_n < 8, \\ S(V_n) = 2 & , \quad V_n \ge 8. \end{cases} \tag{32}$$

Hence, based on Equation (4), which calculates the acceleration from the current and last velocity as V_n and V_{n-1}, we can numerically derive the bounds of the acceleration as follows,

$$\begin{cases} A = 1, \ 0 < V_n \le 4, 0 < V_{n-1} \le 4, \\ 1.083 < A < 1.167, \ (0 < V_n \le 4, 4 < V_{n-1} < 8, \text{ or} \\ \quad 4 < V_n < 8, 0 < V_{n-1} \le 4), 2 < \frac{V_n + V_{n-1}}{2} < 4, \\ 1.417 < A < 1.703, \ (0 < V_n \le 4, 4 < V_{n-1} < 8, \text{ or} \\ \quad 4 < V_n < 8, 0 < V_{n-1} \le 4), 4 < \frac{V_n + V_{n-1}}{2} < 6, \\ 1.5 < A < 2, \ 4 < V_n < 8, 4 < V_{n-1} < 8 \\ 1.583 < A < 2, (V_n \ge 8, 4 < V_{n-1} < 8), \text{ or} \\ \quad (4 < V_n < 8, V_{n-1} \ge 8) \\ A = 2, \ V_n \ge 8, V_{n-1} \ge 8. \end{cases} \tag{33}$$