

USB Keyboard Sniffing with TD

chpie275@gmail.com

<http://beist.org/>

Thanks to "[@Externalist](#)"

0. Contents

1. Intro
2. Design
3. Technique Evaluation
4. Reference

1. Intro

The standard keyboard interfaces used these days are PS/2 and USB. Almost all desktops rely on USB, and laptops use both. The latest laptops models are internally connected to USB. This document introduces the process of sniffing USB keyboards.

There are 3 major types of USB controllers. UHCI(Universal Host Controller Interface), OHCI(Open Host Controller Interface) and EHCI(Enhanced Host Controller Interface). When some type of USB device is installed on a system, the ones that require fast processing are connected to EHCI, but legacy devices such as keyboards are connected to UHCI.

After the connection is established, the UHCI communicates with the operating system via Shared Memory. This document explains how an attacker can log key inputs by intercepting the UHCI packets that go through Shared Memory, and one kind of defense. This technique enables the attacker to monitor the packets, and also manipulate them. Furthermore, the whole process happens even before the operating system detects the packets, so the USB filter drivers don't stand a chance to such an attack.

2. Design

Let's first look at the operation of UHCI. The operating system must reserve a portion of the main memory to use it for Shared Memory. The UHCI data used for the Shared Memory is in the form of a Frame List Header, Queue, and a Linked List. You can consult the UHCI tech sheet for details.

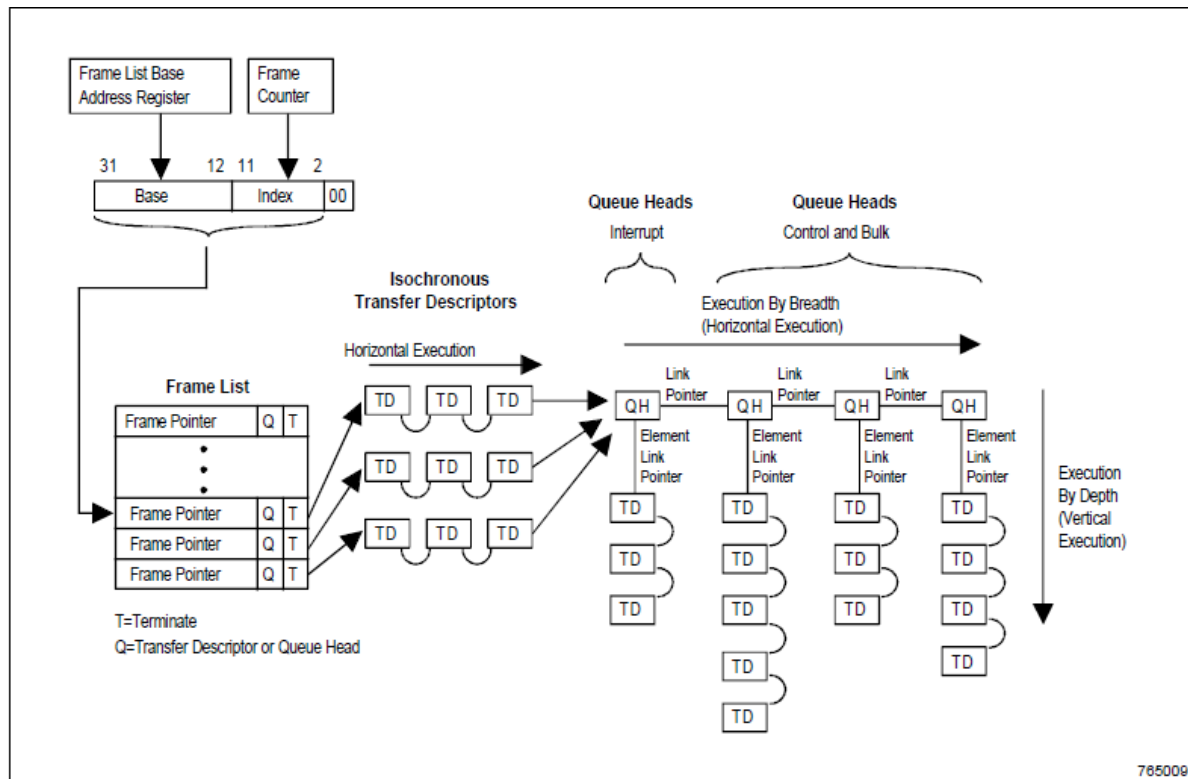


Figure 9. Sample Schedule Layout

<Figure 1. From 'Intel® Universal Host Controller Design Guide'>

Each TD square box corresponds to a packet, and called a 'Transfer Descriptor'(or in short, TD). In the case of Isochronous Transfer, this packet gets connected to a Frame List Header, and the other packets, namely Interrupt, Control, Bulk packets get inserted into a queue and wait for their turn. Those of you who want more info on the terms Isochronous, Interrupt, Control, and Bulk, please refer to the USB tech sheet. USB keyboards are driven by Interrupt or Bulk packets so there is no need to care about Isochronous TDs.

Each Frame List entry corresponds to a Frame, and the current Frame is allocated a certain amount of time and when that time elapses, the next entry is processed. Normally, an Isochronous TD would be connected to an entry, but if there are no Isochronous transfers, then it could be directly connected to the Interrupt Queue.

All data structures in the above picture need to be prepared by the operating system, and the

UHCI reads/writes from this predefined data so that the operating system and the USB devices can communicate.

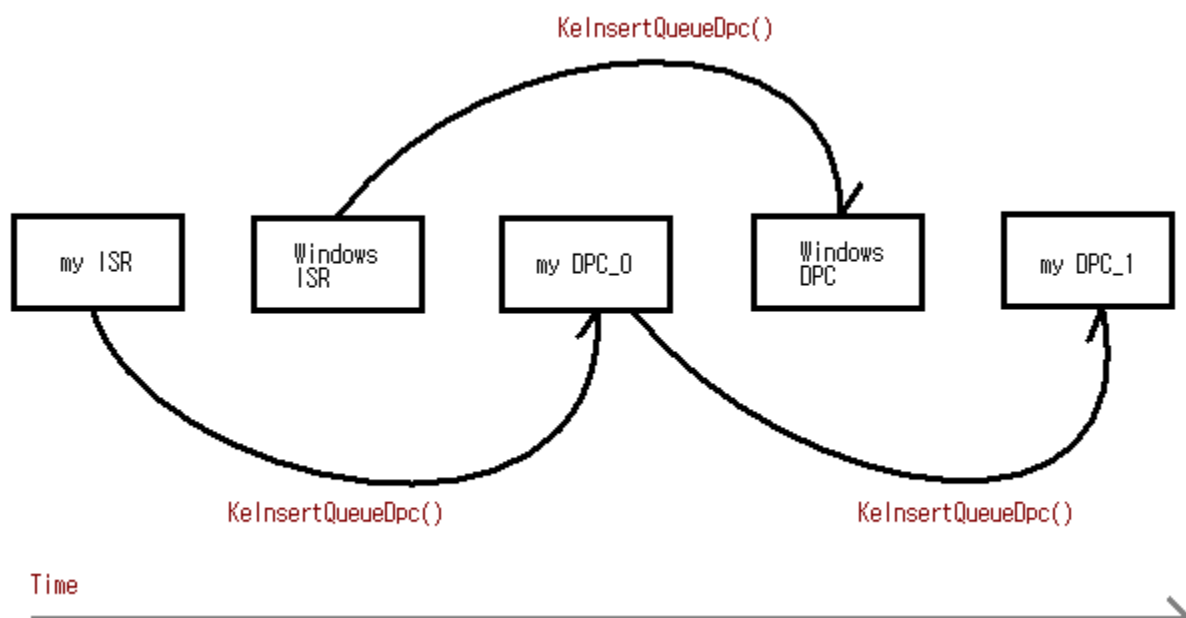
It is possible to get an entry for the current frame using the Frame List Base Address Register and Frame Number Register(Frame Counter in the picture). The Frame Number Register increments by 1 after the timer expires. When each frame is processed, UHCI interrupts the system, and the same happens when each Transfer Descriptor is processed. Which one is used depends on the configuration of the UHCI, and Windows uses both.

The Transfer Descriptors connected to the Queue are purged after the UHCI is done processing them. This part is important, since the queue being empty means all the keyboard Transfer Descriptors are fully processed.

Whenever a UHCI interrupt occurs, Windows uses an internal list in the Virtual Address Space to find the Transfer Descriptors that have already been processed, instead of scanning the whole Shared Memory. All the data structures in the Shared Memory are constructed by Physical Address Pointers, and Windows has a separate copy in a Virtual Address Space.

Therefore, even if one can find the exact time when a specific Transfer Descriptor was processed by hooking the interrupt handler, it is virtually impossible to get the actual Transfer Descriptor address by normal means, such as scanning the Shared Memory.

UHCI is timer based device that activates when a timer expires, and is very slow compared to the CPU. It is possible to execute a timing attack, but only when the Transfer Descriptor has not yet been processed, so one must be able to retrieve the Transfer Descriptor before it is processed.



<Figure 2 – A conceptual graph on how to retrieve the address of the Transfer Descriptor>

In this timing attack, DPC_1 is a Deferred Procedure Call delayed two times, and executes after the Windows DPC. Windows DPC gets the key input data from a Transfer Descriptor that's been processed, and submits the Transfer Descriptor in the Frame List, but most of the Transfer Descriptors are not yet processed when DPC_1 is ready to be executed. This means that the attack uses a race condition, and because of that, some of the key input data can be missed.

Even though not 100% reliable, it is possible to get most of the Transfer Descriptors out of the Queue, and after saving those, use them to retrieve the data from the Transfer Descriptors in DPC_0 when the next interrupt occurs. The already processed Transfer Descriptors have a non zero value in their data size field, so it is easy to differentiate from others. Also, DPC_0 runs even before Windows DPC so that enables the attacker to intercept, and manipulate the Transfer Descriptor before Windows.

What's more, one can easily find out what key has been pressed by getting the Transfer Descriptors from DPC_1. This is because Windows doesn't initialize the data buffer to 0 before submitting, and the earlier data transferred is still in the buffer. But this method does not process the Transfer Descriptor before Windows, so it can only be used to monitor the data, and not manipulate it.

3. Technique Evaluation

Advantages

- The ability to hijack the key input data before any security application that rely on Windows or Filter Drivers
- Multiple keyboards installed on the system does not disturb this technique
- Design is based on hardware so is applicable on other operating systems

Disadvantages

- Not 100% reliable
- Can't use the functionality of the abstract layers of USB

This concludes the process of keyboard sniffing via Transfer Descriptors. As described above, this method intercepts the key input data before any other component does, so one possible defense is to retrieve the data from the Transfer Descriptor, and set it to 0. But this also doesn't result in a 100% success. Therefore, there are still some obstacles to overcome in order to create a reliable security product.

4. Reference

0. USB Keyboard Sniffer with TD, Proof of Concept code
http://beist.org/research/public/usb_keyboard_sniffer/InpHID_withTD.zip
1. Intel® Universal Host Controller Interface Design Guide Revision 1.1
<http://download.intel.com/technology/usb/UHCI11D.pdf>
2. Programming the Microsoft Windows Driver Model 2 (Book)
3. Bochs : Open Source IA-32 Emulation Project
<http://bochs.sourceforge.net/>
4. USB Snoopy
<http://sourceforge.net/projects/usbsnoop/>
5. Inle - USB Keyboard Hooking.pdf (kor)
<http://avinew.cafe24.com/>