Project Report - Bluetooth Keyboard Input Analysis

Group Members:
Dalton Brown, Charles Segers, Christian Andrews

Work Distribution and Work Details:

Dalton

- Bought all essential equipment for the Raspberry Pi, including the Raspberry Pi
- Created the shared workspace on Google Drive and kept that environment organized

Charles

- Bought the bluetooth keyboard
- Outlined the methods of internal mechanism analysis, wrote the steps.sh script.

Christian

- Navigation and analysis of bluetooth packet captures

All

- Helped create the bluetooth keyboard packet captures to analyze
- Helped analyze and write about the packet captures we obtained
- Authored the powerpoint presentation and strategized a live demonstration

Objectives and Motivations:

- Our main objective is to analyze exactly what is going on behind the scenes whenever a user creates a keystroke on a bluetooth keyboard.
- We will connect a bluetooth keyboard to a raspberry pi to capture packets of various keystrokes.
- It is important to understand what a keyboard actually sends a computer to determine how the operating system handles different keystrokes.

Project Overview:

_____This project consists of the applied activity of switching between hardware configurations on a machine, followed afterwards with a two-pronged analysis of the resulting information produced by the hardware switch recorded by our choice programs in the contexts of both the network traffic and the involved internal mechanisms of the operating system.

Project Description:

The *applied activity* of this project consists of switching the keyboard configuration of a chosen Linux system between a usb keyboard and a bluetooth keyboard of choice, all while making sure to use each respective keyboard for a sufficient period of time while each is dominantly configured for use on the system. The applied activity portion of the project is rather simple given the items have been set up

and configured correctly, and we proceed into the complicated actions in the analysis steps.

The *network analysis* of this project consists of the dissection and analysis of a wireless network capture taken from within the Linux system, wherein we filter for bluetooth packets and identify their correspondence with the configuration and use of the bluetooth keyboard. <add more detail here>

The *internal analysis* of this project consists of identifying key places within the operating system that indicate the state of the hardware and the system's interpretation and handling of such (specifically the keyboard). We determined that the following items/attributes of the chosen Linux system met this criteria and were promisingly useful:

- Keyboard configuration file
- /dev/input/ directory contents
- /proc/bus/input/devices file
- /dev/input/<*event*> output stream
- /dev/input/by-id directory contents
- /var/log/kern.log log file
- dmesg output

Among these, the majority produce distinctive states before and after the keyboard switch, which is why most of the methods of observation of these mechanisms include sampling their states before and after the applied activity. The exceptions to this include the event output stream, the kernel log, and dmesg (driver message) outputs. From the three dynamic items, the event output stream is unique in how it must be recorded, because during the applied activity, the likelihood of the event corresponding with the initial keyboard also corresponding to the final keyboard is uncertain and may vary between every run of the applied activity, even if various runs are done in the exact same environment. As a result, the event data stream is the only item we are recording both using real-time techniques and in the before/after fashion. The characteristics that cause this are given more detail in the results analysis. The means of information gathering in the context of these items is explained in the implementation details.

Implementation Details:

System:

We used a Raspberry Pi 3+ Model B V1.2 with Raspbian OS to connect the bluetooth keyboard to and capture the packets. The main reason we used this version was because it had bluetooth capabilities.

Keyboards:

For the USB keyboard, we used a generic logitech MK120 keyboard.

For our bluetooth keyboard, we used the Unisen IPazzPort Mini Bluetooth Keyboard KP-810-21BTS. This device offers not only an entire keyboard but mouse clicks and navigation as well, amongst some miscellaneous impulses it is capable of sending.

Traffic Collection:

We ran **tcpdump -D** to obtain a list of TCP-enabled devices available on the system, which yielded bluetooth0 as the bluetooth interface. We then ran **tcpdump -i bluetooth0 -w btCapture1.pcap** multiple times to create three different packet captures which we could examine and dissect for correspondence to the configuration of and active bluetooth keyboard activity.

Tracking of Internals:

The items and locations of interest described earlier are tracked using a step-by-step series of commands. If the reader wishes to recreate our activity in their own lab environment, they have their choice between either manually running these commands or utilizing the program included with this submission which automates away the overwhelming majority of these steps (a more detailed description is available in the README.txt file). The steps and their brief reasoning are as follows (all should be done as root):

1. **cp /etc/default/keyboard keyboard.before**
    - This command saves a copy of the keyboard configuration file in its state *before* the keyboard switch.
2. **echo ls /dev/input > input.before**
    - This command lists the contents of the directory /dev/input, and instead of printing to the terminal, conveniently pipes the output into the file input.before.
3. **cp /proc/bus/input/devices devices.before**
    - This command saves a copy of the file reflecting the state of input devices at the time it is copied. In this file, identify which **event** in **/dev/input** is associated with the current keyboard in use and run **cat <event>** in another terminal.
4. **echo ls /dev/input/by-id > by-id.before**
    - This command lists the contents of the directory /dev/input/by-id, and instead of printing to the terminal, conveniently pipes the output into the file by-id.before.

5. Run **tail -f /var/log/kern.log** and **dmesg | grep -E "memory|dma|usb|tty"** in two terminals.
    - The kernel log file is often too large to navigate efficiently, so we utilize the command **tail -f** to ensure that the output printed to the terminal contains strictly log entries of a relevant time frame and is guaranteed to be a superset of the entries of interest that correspond with the device activity.
    - **dmesg** is a command used to output kernel and hardware-level message buffer contents in real time. Passing the output through **grep** as we did enabled us to comb through strictly RAM, HD, USB, and serial port related messages.
6. In an arbitrary location such as a text file (wherever you can witness characters being typed), type a little on the usb [initial] keyboard, switch over to the bluetooth [final] keyboard, and type a little on that one too.
7. Repeat steps 1-4, using the suffix **.after** to distinguish the files created as recording the state of said items *after* the keyboard switch.

Challenges and Problems Met:

We did not have any trouble setting up our environment or capturing the packets. We learned to become adept with the active use of the keyboards, because the bluetooth keyboard, once connected and active as the primary keyboard, is capable of disconnecting itself if not used intermittently. We quickly learned to prevent this, as neglecting the keyboard and allowing it to disconnect would confound the evidence/data we used to draw conclusions about bluetooth and the operating system which was predicated on that particular hardware being active and primary.

Experimental Setting:

In order to analyze the packets that a bluetooth keyboard sends, we initially connected the Raspberry Pi to a usb keyboard and mouse and a monitor. The only programs we used were the terminal, to look at the system internals and capture the packets, as well as Wireshark to actually view the packets we captured.

Results Analysis:

Wireless:

In capturing and analyzing the packets sent and received during the keyboard test, we discovered many interesting things about the underlying workings of keyboards, not just bluetooth ones.

In particular, we saw an example of how repeated keystrokes are handled. When a key is pressed, it sends a signal to the driver informing it of which key has been activated. Once said key is released, it then sends an additional signal informing the

driver that a key has been released. The purpose of this <action key up> command is so the driver knows to not repeatedly send the same signal to the operating system multiple times. In contrast, when a key is pressed and held down, only a single signal is sent without a matching <action key up>. It will repeat the command given until it receives said <action key up>. Thus, it is possible to have a massive flood of a single character from a single packet.

Similarly, key modifiers such as SHIFT and CTRL are handled by encoding specific bits within a two byte portion of the packet payload. As long as the modifier keys are held down, the appropriate bits are set to 1. The packet can also differentiate between left and right modifier keys, since each byte corresponds to a different modifier key.

Capslock, numlock, and scroll lock work in a similar fashion, but instead of the keyboard doing nothing but sending commands, the driver actually sends a command to the keyboard once these locks are activated. This signal is used to turn the LED indicators on the keyboard for the locks on and off. This was interesting to learn, as we had assumed the keyboard was only able to send signals, instead of being able to send and receive them.

Each packet also had a protocol associated with it. For the keyboard commands, they were encoded by the HID protocol, which is short for ‘Human Interface Device’. Essentially it just refers to basic I/O hardware such as keyboards and mice. In received packets (such as the ones used to trigger the lock LEDs), they utilize the HCI (host controller interface) protocols, which are Bluetooth specific to handle communication between the operating system and the Bluetooth integrated circuit.

After capturing the packets in a pcap file, we noticed and examined the 5 different sections of communication between the host (the raspberry pi) and the controller (the bluetooth keyboard). The first section is lines 1 through 17 and it shows how a connection is established between the host and controller. The second section, lines 18 through 28, shows how the host authenticates the controller and how they should encrypt data. The third section, lines 29 through 56, is mainly configuration of system settings and which standards to use. The fourth section, lines 57 through 92, show how the controller tells the host which services it has available and how the host decides which services to allow/enable. One of those services was Plug-and-Play and it seemed as though the host disabled that feature. The fifth and final section were the packets of all the keystrokes we entered.

Internals:

We were fortunate to find that kernel logs are more intuitive and humanly readable than we anticipated, the most obvious example being the double timestamp which includes both a conventional date/time as well as the *uptime* in square brackets.

The uptime, though in a universal format, has a subjective base unit across systems. This is to accommodate some systems (such as embedded systems) which may not use a clock, and instead rely solely on timing circuits. On our system, the base unit the second, hence the kernel log records the timing of events down to the microsecond (1/1,000,000 second). We found that the device name is registered at the kernel level, and that the unique hardware ID is taken into account and constructs the identity of the hardware on the system. The lines in the kernel log express the hardware being registered and recorded in locations buried deep in the /devices/platform/soc directory. The lines written to the kernel log during our experimental run pre-presentation are below:

Dec 2 13:47:28 raspberrypi kernel: [1491.048002] Bluetooth: HIDP (Human Interface Emulation) ver 1.2
Dec 2 13:47:28 raspberrypi kernel: [1491.048037] Bluetooth: HIDP socket layer initialized
Dec 2 13:47:28 raspberrypi kernel: [1491.054278] hid-generic 0005:04E8:7021.0003: unknown main item tag 0x0
Dec 2 13:47:28 raspberrypi kernel: [1491.055630] input: iPazzPort Bluetooth Mouse as
/devices/platform/soc/3f201000.serial/tty/ttyAMA0/hci0/hci0:11/0005:04E8:7021.0003/input/input2
Dec 2 13:47:28 raspberrypi kernel: [1491.058126] input: iPazzPort Bluetooth Keyboard as
/devices/platform/soc/3f201000.serial/tty/ttyAMA0/hci0/hci0:11/0005:04E8:7021.0003/input/input3
Dec 2 13:47:28 raspberrypi kernel: [1491.059486] input: iPazzPort Bluetooth Consumer Control as
/devices/platform/soc/3f201000.serial/tty/ttyAMA0/hci0/hci0:11/0005:04E8:7021.0003/input/input4
Dec 2 13:47:28 raspberrypi kernel: [1491.059884] input: iPazzPort Bluetooth System Control as
/devices/platform/soc/3f201000.serial/tty/ttyAMA0/hci0/hci0:11/0005:04E8:7021.0003/input/input5
Dec 2 13:47:28 raspberrypi kernel: [1491.060236] hid-generic 0005:04E8:7021.0003: input,hidraw2: BLUETOOTH HID v1.1b Mouse [iPazzPort Bluetooth] on b8:27:eb:2d:46:6d

Interestingly enough, the relevant kernel log lines also presented themselves in similar form throughout in the dmesg output:

[1491.055630] input: iPazzPort Bluetooth Mouse as
/devices/platform/soc/3f201000.serial/tty/ttyAMA0/hci0/hci0:11/0005:04E8:7021.0003/input/input2
[1491.058126] input: iPazzPort Bluetooth Keyboard as
/devices/platform/soc/3f201000.serial/tty/ttyAMA0/hci0/hci0:11/0005:04E8:7021.0003/input/input3
[1491.059486] input: iPazzPort Bluetooth Consumer Control as
/devices/platform/soc/3f201000.serial/tty/ttyAMA0/hci0/hci0:11/0005:04E8:7021.0003/input/input4
[1491.059884] input: iPazzPort Bluetooth System Control as
/devices/platform/soc/3f201000.serial/tty/ttyAMA0/hci0/hci0:11/0005:04E8:7021.0003/input/input5
[1926.573690] input: iPazzPort Bluetooth Mouse as
/devices/platform/soc/3f201000.serial/tty/ttyAMA0/hci0/hci0:11/0005:04E8:7021.0004/input/input6
[1926.576593] input: iPazzPort Bluetooth Keyboard as
/devices/platform/soc/3f201000.serial/tty/ttyAMA0/hci0/hci0:11/0005:04E8:7021.0004/input/input7
[1926.581511] input: iPazzPort Bluetooth Consumer Control as
/devices/platform/soc/3f201000.serial/tty/ttyAMA0/hci0/hci0:11/0005:04E8:7021.0004/input/input8
[1926.582088] input: iPazzPort Bluetooth System Control as
/devices/platform/soc/3f201000.serial/tty/ttyAMA0/hci0/hci0:11/0005:04E8:7021.0004/input/input9
[2137.401029] input: iPazzPort Bluetooth Mouse as
/devices/platform/soc/3f201000.serial/tty/ttyAMA0/hci0/hci0:11/0005:04E8:7021.0005/input/input10
[2137.401907] input: iPazzPort Bluetooth Keyboard as
/devices/platform/soc/3f201000.serial/tty/ttyAMA0/hci0/hci0:11/0005:04E8:7021.0005/input/input11
[2137.411951] input: iPazzPort Bluetooth Consumer Control as
/devices/platform/soc/3f201000.serial/tty/ttyAMA0/hci0/hci0:11/0005:04E8:7021.0005/input/input12

[2137.412451] input: iPazzPort Bluetooth System Control as /devices/platform/soc/3f201000.serial/tty/ttyAMA0/hci0/hci0:11/0005:04E8:7021.0005/input/input13

The remarkable thing that we found here is that nearly identical system messages present themselves both through the kernel log and dmesg with only slight variations in their presentation and format, which supports the educated guess that system log outputs would be a superset of messages giving an exhaustive description of the switch as long as it was timed correctly.

We found that the keyboard configuration file is descriptive of the keyboard in the context of the X-session of the system. X, or X11, is the lowest substrate of the communication of visual information providing a framework for GUI development. The keyboard specifies the XKB configuration details like so:

XKBMODEL=pc105
XKBLAYOUT=us
XKBVARIANT=
XKBOPTIONS=
BACKSPACE=guess

Observation suggests that the keyboard is processed as a transiently virtual device which consists partially as a construct of the operating system and is *driven* by appropriate hardware.

The “event”s in /dev/input are virtual creations of the OS as part of X that can both combine inputs of similar devices, and create multiple inputs for a single device. They can also be created on startup for automatic self-testing. In our situation, adding the bluetooth keyboard to the system lead to new events being created. Since the events are characteristically comparable to the keyboard configuration file in their partial abstraction, the most substantiated explanation for the addition of new events coinciding with the addition of a bluetooth keyboard is that the esoteric nature of the mobile keyboard+mouse hardware is not universally accommodated to by relatively simple means. The exact events added to /dev/input as a consequence of the switch during our run were: **event2 event3 event4 event5 mouse1**

Future Work:

_____A part of this project which may potentially be innovated upon is the automation of system process/attribute monitoring in a way that extends upon the basic script we made to accompany our demonstration. The next step closer to a higher realization of what we learned from this project would be a script that has the capacity to initiate switches between hardware in a manner informed by proper and intelligent analysis of the specimens we chose to analyze. An example of this could be a set of keystrokes that signal the system to revert back to the USB keyboard, which would be detected by

the script running tcpdump and running commands to disengage the keyboard given an invocation. This is also one way to potentially make use of the features of the bluetooth keyboard which are inapplicable to an average Linux installation.

References:

Bluetooth menu for raspberry pi:
https://www.cnet.com/how-to/how-to-setup-bluetooth-on-a-raspberry-pi-3/

How to turn a raspberry pi into a network monitoring node:
https://www.networkworld.com/article/2225683/cisco-subnet-raspberry-pi-as-a-network-monitoring-node.html

How to capture bluetooth packets:
https://www.agnosticdev.com/content/how-capture-bluetooth-traffic-tcpdump-linux

Bluetooth protocols:
https://www.electronics-notes.com/articles/connectivity/bluetooth/host-l2cap-sdp-gap.php
https://en.wikipedia.org/wiki/List_of_Bluetooth_protocols#Host_Controller_Interface_(HCI)
https://www.linuxtechi.com/10-tips-dmesg-command-linux-geeks/
https://wiki.ubuntu.com/LaptopTesting/Keycodes
https://unix.stackexchange.com/questions/103230/capturing-key-input-from-events-device-and-mapping-it-toggle-touchpad-key-is-un

Misc:
https://stackoverflow.com/questions/2775461/linux-keyboard-event-capturing-dev-inputx
https://unix.stackexchange.com/questions/340430/dev-input-what-exactly-is-this
https://stackoverflow.com/questions/31107161/what-is-meaning-of-the-rightmost-numbers-in-the-var-log-kern-log