|  |  |  |
| --- | --- | --- |
| Network Analysis Skills | | |
|  |  |  |
|  | | |

# Description:

**S0184**: Skill in analyzing traffic to identify network devices.

**S0245**: Skill in navigating network visualization software.

**S0199:** Skill in creating and extracting important information from packet captures.

Description of challenge….

**Skills Required:**

* Scripting
* Wireshark/Tshark

# Solving the Challenge:

The first thing to do when given a PCAP file for analysis is to open it up in Wireshark.

A screenshot of a cell phone

Description automatically generatedThis traffic looks way different than the normal PCAP traffic, and we can see that the protocol is USB, and USBHID mixed in, so we know the capture is related to some USB device. With the USBHID protocols mixed in we can also assume that the activity we’re looking for is regarding that, and the rest is just various USB traffic. USBHID is (USB-Human Interface Device), things like keyboards, mice, trackpads, etc.

Looking farther into the traffic, we can see that there is a long string of communication that is from the source to the host without any interruption, which we can guess is the HID being used.

A screenshot of a cell phone

Description automatically generated

Looking further into this traffic, it is easy to tell that the ‘IRP\_ID’ header is flip flopping every packet from ‘0xffffa5045d1653c0’ to ‘0xffffa50460432a60’ which seems like a KEY\_UP, and KEY\_DOWN event. Each of the packets follows this same format, and nothing seems to differ for the first 10 or so, until eventually the ‘Leftover Capture Data’ field begins populating with random bytes, and then the next packet that field will be blank. This further suggests that the events are KEY\_UP and KEY\_DOWN and that ‘0xffffa504d1653c0’ corresponds to KEY\_DOWN (pressed), this leftover field data is most likely the data corresponding to the key press.

A close up of text on a white background

Description automatically generatedA short google search later reveals this gist paste with C Macros for defining hexcodes to keys on the keyboard. A little more googling reveals what the format of the ‘Leftover Capture Data’ field is:

00:00:00:00:00:00:00:00

\_Byte 0: The first byte is for modifier keys (LSHIFT, LCTRL, ALT, etc.)

\_Byte 1: Reserved Byte

\_Byte 2-7: Each byte represents a key pressed in the frame, can hold up to 6 key presses in one frame.

So for example the string:

02:00:05:00:00:00:00:00

represents a capital B, because the initial byte is 0x02 (LSHIFT) and the keypress field is 0x05 (b).

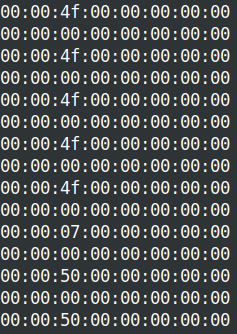
A close up of text on a black background

Description automatically generatedUsing all of this information it will be easy to write a parser to take in these strings and return their key representation. After a little formatting with VIM using s replace, and norm we can take that C code and get a python dictionary representing the mappings.

Now that we have a mapping, all we need is the data for us to input into python, since python will be easier to work with than wireshark. Using tshark, we can extract only the desired fields and get a nice output of the data to work with

tshark -T fields -e usb.capdata -r deadly\_arthropod.pcap > parse.txt

will give us the ’usb.capdata’ fields from the capture and write them out to our parsefile, which will look like this.



After some python scripting and a bit of debugging, we run the file through, and retrieve this string from the dataset.

A close up of a logo

Description automatically generated

Well that wasn’t so hard, that looks like our flag ‘Th1sC0uldB3MyR3alP@ssw0rd’, just add HTB in front and enclose it in curly braces and it looks perfect. Sadly it wasn’t that easy, after a lot of fiddling with different caps combinations and initially not having the shift modifiers for numbers (LSHIFT + ‘2’ == @) I came to the conclusion that was not the flag, but a red herring. So the flag must be that string at the bottom we initially though was gibberish (I even named that part of the file ‘gibberish.txt’).

So obviously we’ll have to do some sort of formatting with this gibberish string to achieve a flag, and after further analysis we can some characters that would be part of the flag, htb{}.

A close up of a logo

Description automatically generated

After what seemed like ages of trying some bit shifting, incrementing and decrementing the characters corresponding to the ‘<’ and ‘>’ with nothing working it finally clicked. The gibberish string is one long keyscan corresponding to the input from the USBHID, it had nothing to do with encoding/formatting, it was really just keyboard input. It doesn’t really make sense without an example so here is what the first couple characters would be.

input: QK<\_>.<<<<H>5

so the user would explicitly be entering that sequence of keys, so when you follow it exactly, with left and right arrows corresponding to the left and right arrow keys you get the following (‘|’ represents where the user’s cursor is, and the numbers indicate the current keypress we’re at):

1. Q |
2. QK |
3. Q | K
4. Q \_ | K
5. Q \_ K |
6. Q \_ K . |
7. Q \_ K | .
8. Q \_ | K .
9. Q | \_ K .
10. | Q \_ K .
11. H | Q \_ K .
12. H Q | \_ K .
13. H Q 5 | \_ K .

Final String: HQ5\_K.

So when going through the whole string like it is an input from the keyboard, we are given the following string:

HTB{If\_It\_Quack5\_It’s\_a\_K3yboard…}

Scripts used to solve the challenge:

**Key Parser:**

**A screenshot of a computer

Description automatically generated**

**Keywalk Parser:**

**A screenshot of a cell phone

Description automatically generated**