Introduction to USB and Fuzzing

edition

Matt DuHarte

@Crypto_Monkey

To be clear USB 2.0 is

- so sayth the content at http://www.usb.org/home, the home of all good specifications and fount of all USB knowledge
- However, the opinions expressed herein are mine, or sombody else's or my dogs (she is very smart)

Agenda

- quick intro to USB hardware
- intro to USB protocols and software
- quick intro to face dancer and umap.py

- This is a slightly cut down version of the presentation given during the much longer USB Fuzzing Workshop at BSides Seattle 2014

Introduction to USB

Hardware

Types of Connectors - Electrical

- Most USB use 4 wires
  - +5V DC for powering devices
  - Power Ground
  - Two data connections automatically terminated when not in use
    - Differential Data +
    - Differential Data -
- Some connectors have a On The Go (OTG) wire called the ID pin
  - OTG allows the device to act as both a target Device and have limited host functionality
  - Not in scope for what we are discussing today

Types of Connectors – Physical

left to right

- Micro USB3
- Micro B USB2
- Mini B USB2 (5 wire OTG)
- Type B USB2
- Type A USB2

USB 2 Connector Facts

- Only Type A should put power on the cable
  - Making a A to A cable rare and pretty dangerous
- The 5-wire version of the Mini /Micro Type B (with the indents) is used for On The Go
  - This is common on things like phones that can act as either a host and target

Best Practice

- Use the shortest USB cable you can find
  - we do not waste current on long wires for devices just sitting next to our laptop.
  - shorter cables also allow for less potential interference
    - Yes, USB signaling uses a differential pair of signal wires but we don't want to rely on that.
    - **Ben Heck's Tech Time Out: What is Differential Signaling?**
    - (https://www.youtube.com/watch?v=J8imfqsUGCk)

Cheap Amazon Cables Rule!

Roll over image to zoom in

6 INCH USB 2.0 Certified 480Mbps Type A Male to Mini-B/5-Pin Male Cable

by My Cable Mart

Be the first to review this item

Price: $2.46 + $4.02 shipping

Note: Not eligible for Amazon Prime.

In Stock.

Ships from and sold by My Cable Mart.

Estimated Delivery Date: Thursday, April 23 when you choose Two-Day Shipping at checkout.

- SUGGESTION: Order this item in QUANTITY due to light weight to minimize TOTAL shipping costs
- SHIPPING WEIGHT (Each): 0.06 lbs.

http://www.amazon.com/Certified-480Mbps-Mini-B-5-Pin-Cable/dp/B00U1SL7Q2/

Introduction to USB

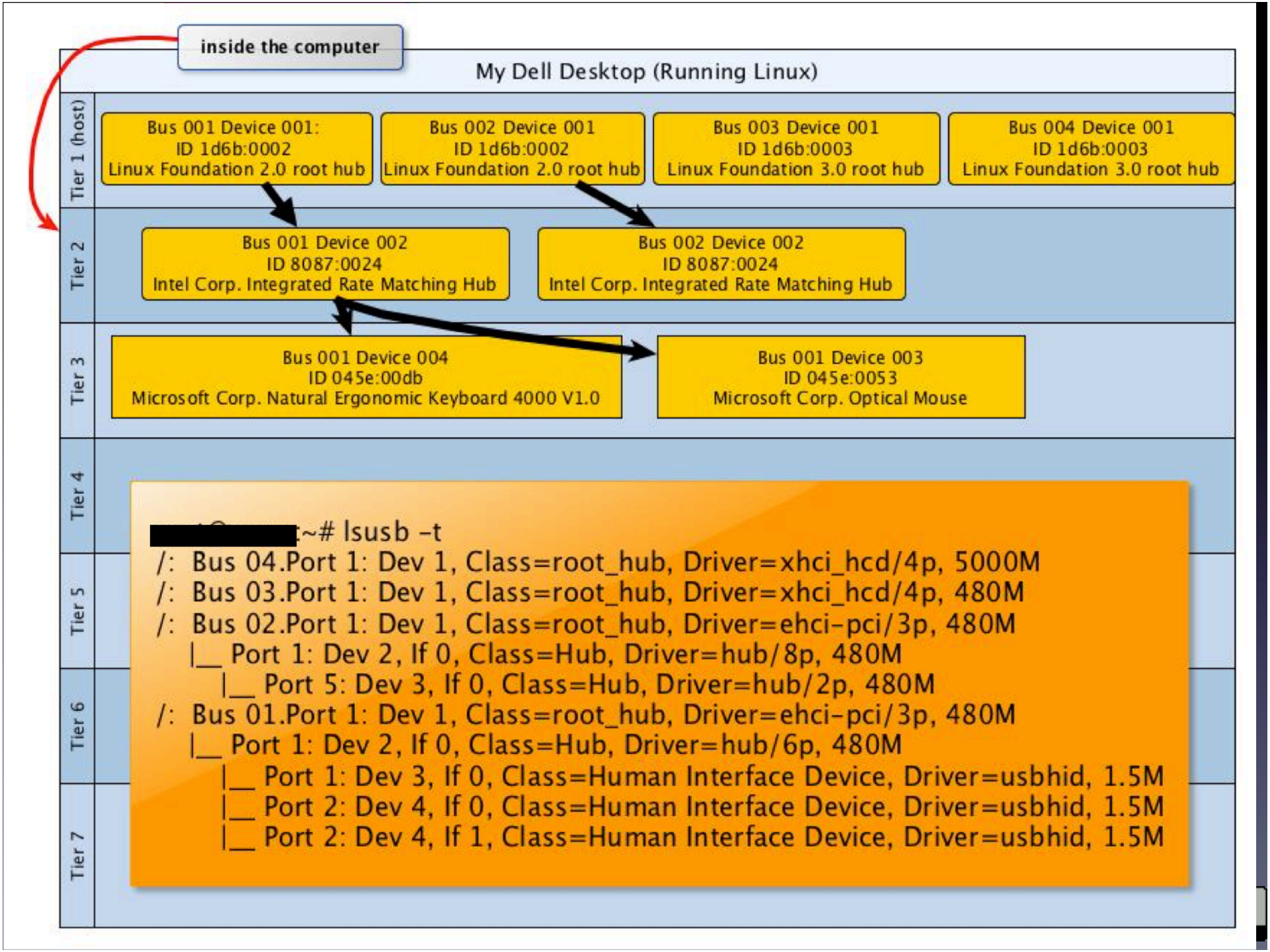
Hardware and Connection Knowledge

Connection Types in USB 2.0

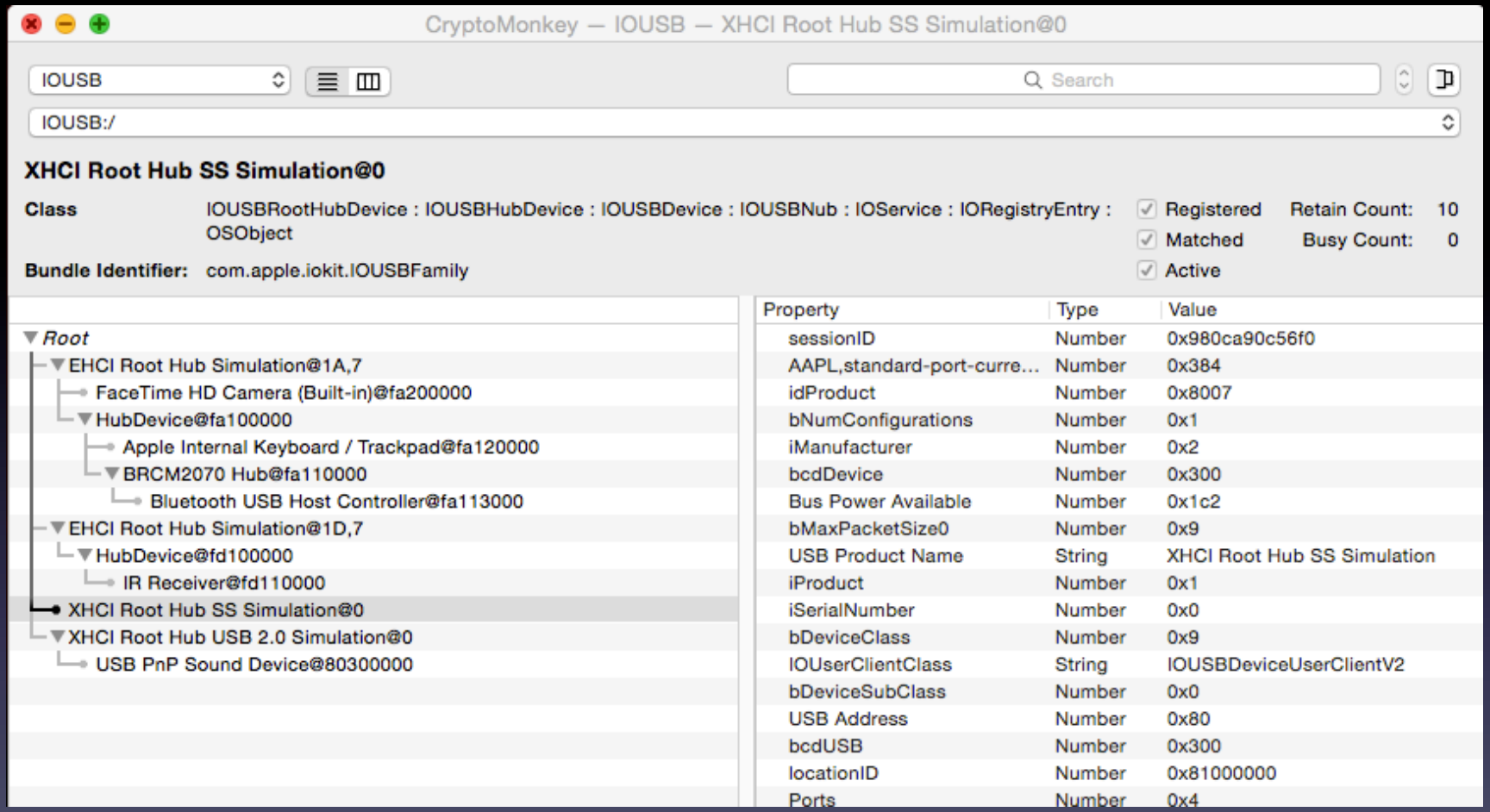
- Low Speed (1.5 Mb/s) (USB 1.1)
  - interactive devices like mice, keyboards
- Full Speed (12 Mb/s)
  - audio, microphones
- High Speed (480 Mb/s)
  - storage devices, USB hubs

The USB Network

- one Host
  - the 'root hub"
- multiple Hubs
  - up to 5 allowed between any device and the root hub
  - that makes up to 7 tiers total
- up to 127 devices (addresses)


inside the computer
My Dell Desktop (Running Linux)
Tier 1 (host)
Bus 001 Device 001:
ID 1d6b:0002
Linux Foundation 2.0 root hub
Bus 002 Device 001
ID 1d6b:0002
Linux Foundation 2.0 root hub
Bus 003 Device 001
ID 1d6b:0003
Linux Foundation 3.0 root hub
Bus 004 Device 001
ID 1d6b:0003
Linux Foundation 3.0 root hub
Tier 2
Bus 001 Device 002
ID 8087:0024
Intel Corp. Integrated Rate Matching Hub
Bus 002 Device 002
ID 8087:0024
Intel Corp. Integrated Rate Matching Hub
Tier 3
Bus 001 Device 004
ID 045e:00db
Microsoft Corp. Natural Ergonomic Keyboard 4000 V1.0
Bus 001 Device 003
ID 045e:0053
Microsoft Corp. Optical Mouse
Tier 4
Tier 5
Tier 6
Tier 7
[redacted]:~# lsusb -t
/: Bus 04.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/4p, 5000M
/: Bus 03.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/4p, 480M
/: Bus 02.Port 1: Dev 1, Class=root_hub, Driver=ehci-pci/3p, 480M
|__ Port 1: Dev 2, If 0, Class=Hub, Driver=hub/8p, 480M
|__ Port 5: Dev 3, If 0, Class=Hub, Driver=hub/2p, 480M
/: Bus 01.Port 1: Dev 1, Class=root_hub, Driver=ehci-pci/3p, 480M
|__ Port 1: Dev 2, If 0, Class=Hub, Driver=hub/6p, 480M
|__ Port 1: Dev 3, If 0, Class=Human Interface Device, Driver=usbhid, 1.5M
|__ Port 2: Dev 4, If 0, Class=Human Interface Device, Driver=usbhid, 1.5M
|__ Port 2: Dev 4, If 1, Class=Human Interface Device, Driver=usbhid, 1.5M

OSX



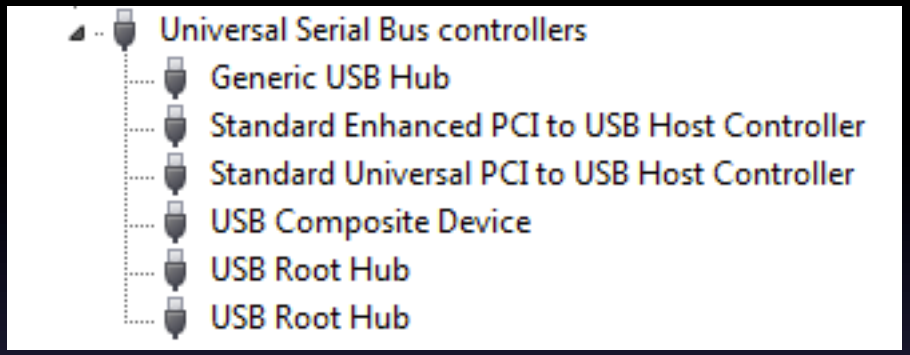
- from Apple Developer "Hardware Tools For Xcode" use IORegExplorer
  - or download USB Prober from the Apple IOUSBFamily package
- or use "system_profiler SPUSBDataType" from a terminal

Did you notice the SuperSpeed host on my Macbook Pro?

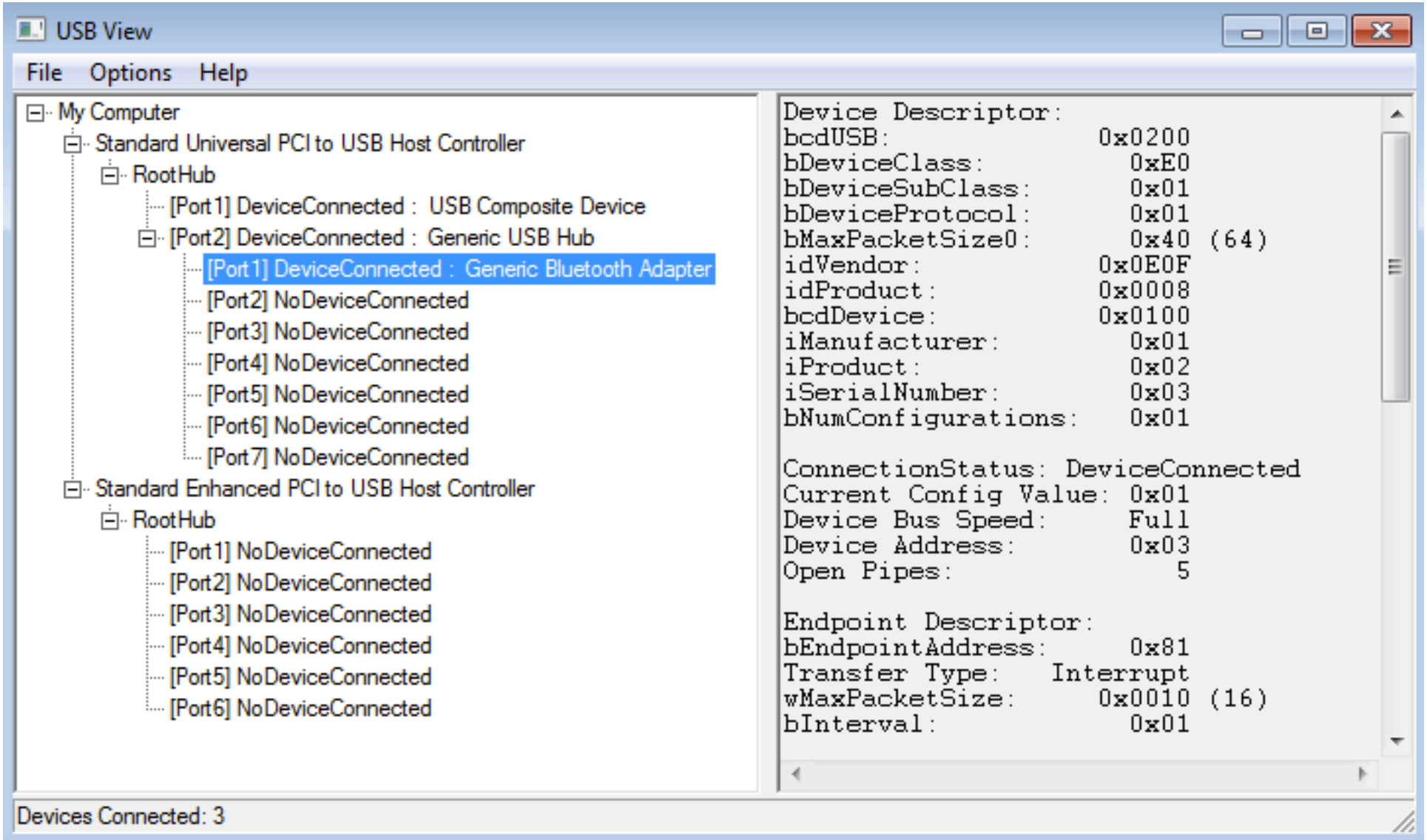
- That is USB3 and there is no USB3 port on the laptop
- system_profiler shows that those are the USB3 ports on my Thunderbolt dock
- And it will step down to USB2 for us to attack if we want
- USB encapsulated in Thunderbolt, possibilities?

```
USB 3.0 SuperSpeed Bus:

  Host Controller Location: Thunderbolt
  Host Controller Driver: AppleUSBXHCI
  PCI Device ID: 0x1100
  PCI Revision ID: 0x0001
  PCI Vendor ID: 0x1b73
  Bus Number: 0x40

USB 3.0 Hi-Speed Bus:

  Host Controller Location: Thunderbolt
  Host Controller Driver: AppleUSBXHCI
  PCI Device ID: 0x1100
  PCI Revision ID: 0x0001
  PCI Vendor ID: 0x1b73
  Bus Number: 0x40
```

Windows



- Device Manager is pretty sparse on info



- There is an old tool called USBView that works better
- http://www.techrepublic.com/blog/windows-and-office/map-and-troubleshoot-your-usb-ports-with-microsoft-usb-view/

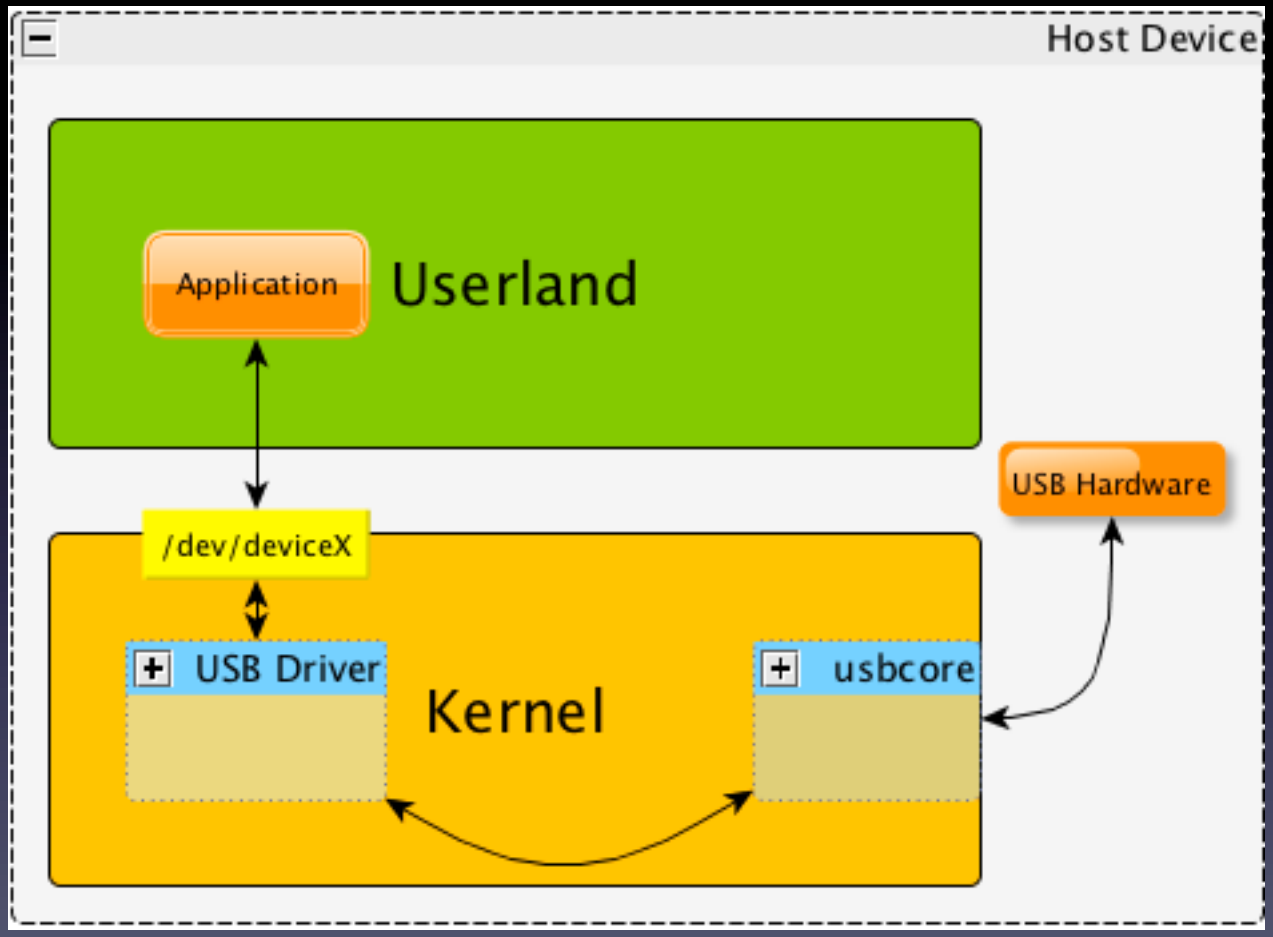
Introduction to USB

Software Components and Kernel Knowledge

Kernels and Drivers

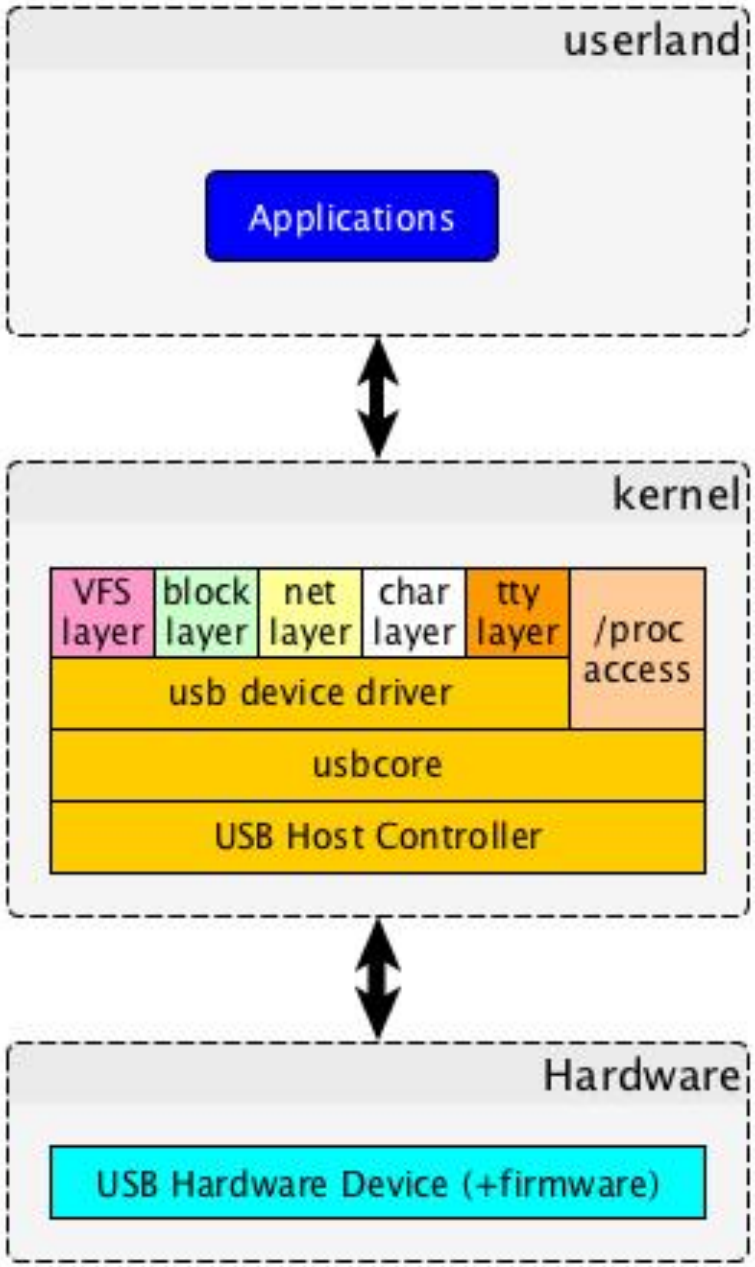
- We will be discussing the Linux 3.x kernel
  - not because it is better
  - but because the source is available for us to work with easily
  - but Windows and OSX work in a similar manner

A Linux Host Information Flow



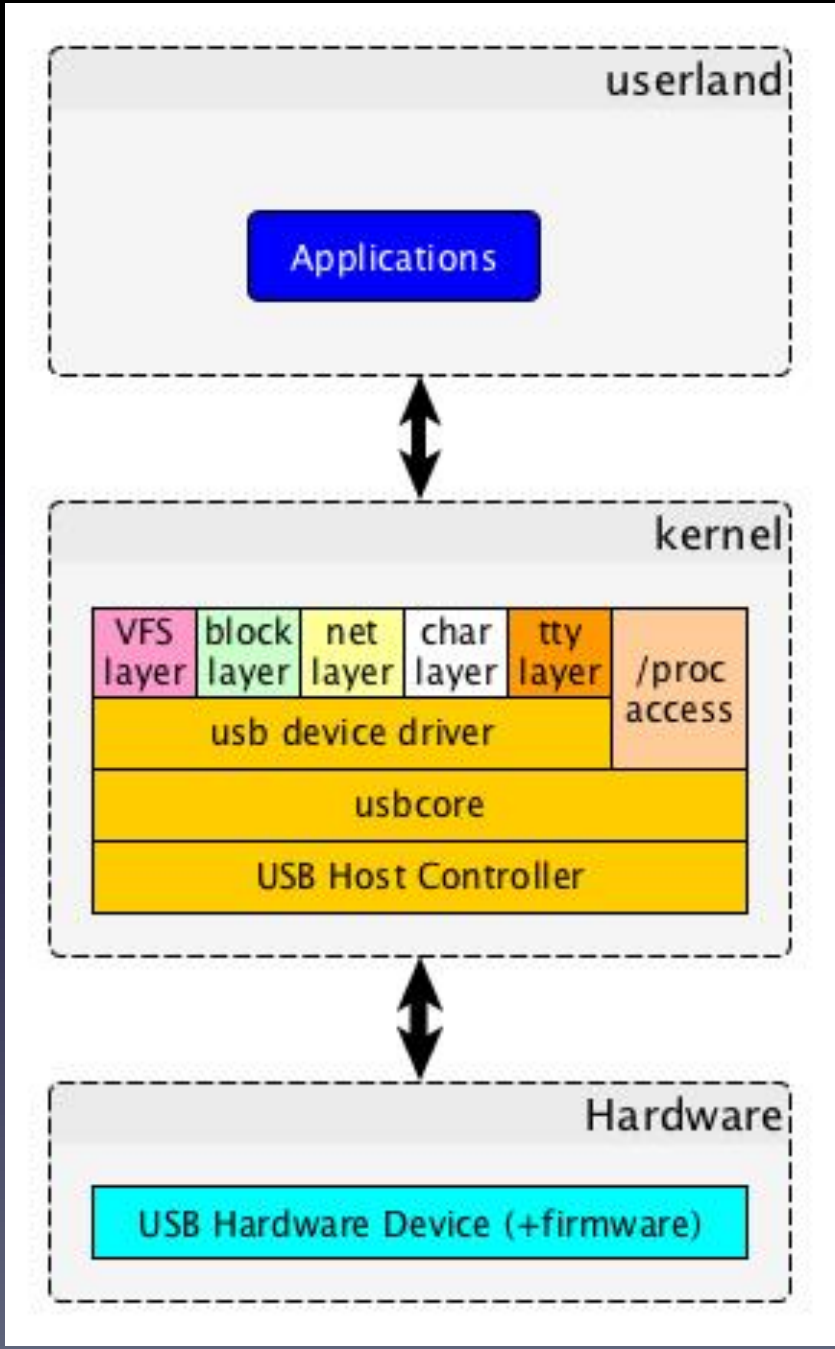
- The kernel has several pieces of code to handle USB

While this covered the overall flow we need to look closer at the parts and where they come from to understand how to test them



The OS View of USB

- an excellent reference is Jonathan Corbet, Greg Kroah-Hartman, and Alessandro Rubini's Linux Device Drivers 3rd edition, chapter 13 where we find this fine figure
  - http://lwn.net/images/pdf/LDD3/ch13.pdf
- The chief take away here is that kernel code (usbcore) and multiple drivers stack to create a path to userland
- you can attack any level if you craft you code carefully



Picking Targets

- There are targets you get protocol level access to
- And ones you get structural level access to

What kernel modules can be loaded?

- finding all USB modules available to load on Linux
  - find /lib/modules/`uname -r` | grep -i usb
- Find all currently loaded modules
  - lsmod | grep usb
  - use modinfo for more info about a module
- use insmod and rmmod to load and unload a driver
  - look for issues and messages in dmesg (tail –f /var/log/messages)

finding what is loaded

- Find where the USB bus is on the PCI bus

```
    ;:~$ lspci
00:00.0 Host bridge: Intel Corporation Xeon E3-1200 v2/3rd Gen Core processor DRAM Controller (rev 09)
00:01.0 PCI bridge: Intel Corporation Xeon E3-1200 v2/3rd Gen Core processor PCI Express Root Port (rev 09)
00:14.0 USB controller: Intel Corporation 7 Series/C210 Series Chipset Family USB xHCI Host Controller (rev 04)
00:16.0 Communication controller: Intel Corporation 7 Series/C210 Series Chipset Family MEI Controller #1 (rev 04)
00:16.3 Serial controller: Intel Corporation 7 Series/C210 Series Chipset Family KT Controller (rev 04)
00:19.0 Ethernet controller: Intel Corporation 82579LM Gigabit Network Connection (rev 04)
00:1a.0 USB controller: Intel Corporation 7 Series/C210 Series Chipset Family USB Enhanced Host Controller #2 (rev 04)
00:1b.0 Audio device: Intel Corporation 7 Series/C210 Series Chipset Family High Definition Audio Controller (rev 04)
00:1d.0 USB controller: Intel Corporation 7 Series/C210 Series Chipset Family USB Enhanced Host Controller #1 (rev 04)
00:1e.0 PCI bridge: Intel Corporation 82801 PCI Bridge (rev a4)
00:1f.0 ISA bridge: Intel Corporation Q77 Express Chipset LPC Controller (rev 04)
00:1f.2 RAID bus controller: Intel Corporation 82801 SATA Controller [RAID mode] (rev 04)
00:1f.3 SMBus: Intel Corporation 7 Series/C210 Series Chipset Family SMBus Controller (rev 04)
01:00.0 VGA compatible controller: Advanced Micro Devices, Inc. [AMD/ATI] Caicos XT [Radeon HD 7470/8470 / R5 235 OEM]
01:00.1 Audio device: Advanced Micro Devices, Inc. [AMD/ATI] Caicos HDMI Audio [Radeon HD 6400 Series]
```

We can see the USB controller on the PCI bus as device ID 1D

finding info about connected devices

- Locate the data about the loaded devices

```
:~$ ls /sys/devices/pci0000\:00/0000\:00\:1d.0/usb2/
2-0:1.0                bmAttributes          devpath          product
2-1                    bMaxPacketSize0       driver           quirks
authorized             bMaxPower             ep_00            removable
authorized_default     bNumConfigurations    firmware_node    remove
avoid_reset_quirk      bNumInterfaces        idProduct        serial
bcdDevice              busnum                idVendor         speed
bConfigurationValue    configuration         ltm_capable      subsystem
bDeviceClass           descriptors           manufacturer     uevent
bDeviceProtocol        dev                   maxchild         urbnum
bDeviceSubClass        devnum                power            version
```

more device details

- On Fedora variants usb device info is available via the usbfs (/proc/bus/usb/devices)
- On Debian variants try the usb-devices command

```
:~$ usb-devices

T:  Bus=01 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1 Spd=480 MxCh= 3
D:  Ver= 2.00 Cls=09(hub  ) Sub=00 Prot=00 MxPS=64 #Cfgs=  1
P:  Vendor=1d6b ProdID=0002 Rev=03.13
S:  Manufacturer=Linux 3.13.0-40-generic ehci_hcd
S:  Product=EHCI Host Controller
S:  SerialNumber=0000:00:1a.0
C:  #Ifs= 1 Cfg#= 1 Atr=e0 MxPwr=0mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
```

OSX has the same sort of commands (but no /proc)

```
$ kextstat | grep -i usb
  39   15 0xffffff7f80fc7000 0x65000    0x65000    com.apple.iokit.IOUSBFamily (705.4.14) <12 7 5 4 3 1>
  41    0 0xffffff7f8121f000 0x19000    0x19000    com.apple.driver.AppleUSBEHCI (705.4.14) <39 12 7 5 4 3 1>
  54    0 0xffffff7f811fd000 0x1b000    0x1b000    com.apple.driver.AppleUSBHub (705.4.1) <39 5 4 3 1>
  76    1 0xffffff7f81192000 0xa000     0xa000     com.apple.driver.AppleUSBComposite (705.4.9) <39 4 3 1>
  77    0 0xffffff7f811f6000 0x7000     0x7000     com.apple.driver.AppleUSBMergeNub (705.4.0) <76 39 4 3 1>
  78    4 0xffffff7f81189000 0x9000     0x9000     com.apple.iokit.IOUSBHIDDriver (705.4.0) <39 33 5 4 3 1>
  82    0 0xffffff7f82224000 0x4000     0x4000     com.apple.driver.AppleUSBTCKeyboard (240.2) <78 39 33 7 6 5 4 3 1>
  84    0 0xffffff7f8222f000 0x13000    0x13000    com.apple.driver.AppleUSBMultitouch (245.2) <78 39 33 6 5 4 3 1>
  90    0 0xffffff7f8222a000 0x3000     0x3000     com.apple.driver.AppleUSBTCButtons (240.2) <78 39 33 7 6 5 4 3 1>
 116    1 0xffffff7f8104c000 0x3000     0x3000     com.apple.iokit.IOUSBUserClient (705.4.0) <39 7 5 4 3 1>
 124    1 0xffffff7f81643000 0x23000    0x23000    com.apple.iokit.IOBluetoothHostControllerUSBTransport (4.3.1f2) <110
39 12 11 7 5 4 3 1>
 125    0 0xffffff7f81666000 0x9000     0x9000     com.apple.iokit.BroadcomBluetoothHostControllerUSBTransport (4.3.1f2)
 <124 110 39 12 11 7 5 4 3>
 154    0 0xffffff7f8365f000 0x8000     0x8000     org.virtualbox.kext.VBoxUSB (4.3.18) <153 116 39 7 5 4 3 1>
 232    0 0xffffff7f836e4000 0x20000    0x20000    com.apple.driver.AppleUSBXHCI (705.4.14) <39 12 7 5 4 3 1>
 234    0 0xffffff7f83708000 0x4c000    0x4c000    com.apple.driver.AppleUSBAudio (295.22) <92 39 12 5 4 3 1>
```

Introduction to USB

Addressing USB Devices and Dumping Traffic

The protocol

FS	8	0:05.817.174	8 B	00	00	SETUP txn	80 06 00 01 00 00 40 00
FS	9	0:05.817.174	3 B	00	00	SETUP packet	2D 00 10
FS	10	0:05.817.177	11 B	00	00	DATA0 packet	C3 80 06 00 01 00 00 40 00 DD 94
FS	11	0:05.817.186	1 B	00	00	ACK packet	D2

- Transactions vs. packets
  - Transaction are the smallest unit of command
  - most are made of about three packets
    - They can be 1,2, 3 or 4 packets though
  - Each packet has a Packet Identifier (PID)
    - describes the purpose of the packet

Understanding Transactions

- Begin with a token packet (4 bytes)
  - defines device, endpoint quality and direction for the transaction
- Data packet (Max 8 bytes-low, 1023 bytes-full, 1024 bytes-high)
  - Umm, it moves data in bytes sizes
- Handshake Packet (1 byte)
  - Acknowledgement of data receipt

Token PID (XX01)

Token	OUT	0001B	Address + endpoint number in host-to-function transaction
	IN	1001B	Address + endpoint number in function-to-host transaction
	SOF	0101B	Start-of-Frame marker and frame number
	SETUP	1101B	Address + endpoint number in host-to-function transaction for SETUP to a control pipe

*Note: PID bits are shown in MSb order. When sent on the USB, the rightmost bit (bit 0) will be sent first.

- from **Universal Serial Bus Specification Revision 2.0** Table 8-1 at http://www.usb.org/developers/docs/usb20_docs/

Why do we need IN and OUT?

- Only the host initiates transactions
  - there is only one address in the transaction
  - So IN and OUT tell us which party the information is for

Data PID (XX01)

Data	DATA0	0011B	Data packet PID even
	DATA1	1011B	Data packet PID odd
	DATA2	0111B	Data packet PID high-speed, high bandwidth isochronous transaction in a microframe (see Section 5.9.2 for more information)
	MDATA	1111B	Data packet PID high-speed for split and high bandwidth isochronous transactions (see Sections 5.9.2, 11.20, and 11.21 for more information)

*Note: PID bits are shown in MSb order. When sent on the USB, the rightmost bit (bit 0) will be sent first.

- This is bulk data transfer, other PIDs also can move data
- from **Universal Serial Bus Specification Revision 2.0** Table 8-1 at http://www.usb.org/developers/docs/usb20_docs/

Handshake PID (XX10)

Handshake	ACK	0010B	Receiver accepts error-free data packet
	NAK	1010B	Receiving device cannot accept data or transmitting device cannot send data
	STALL	1110B	Endpoint is halted or a control pipe request is not supported
	NYET	0110B	No response yet from receiver (see Sections 8.5.1 and 11.17-11.21)

*Note: PID bits are shown in MSb order. When sent on the USB, the rightmost bit (bit 0) will be sent first.

- from **Universal Serial Bus Specification Revision 2.0** Table 8-1 at http://www.usb.org/developers/docs/usb20_docs/

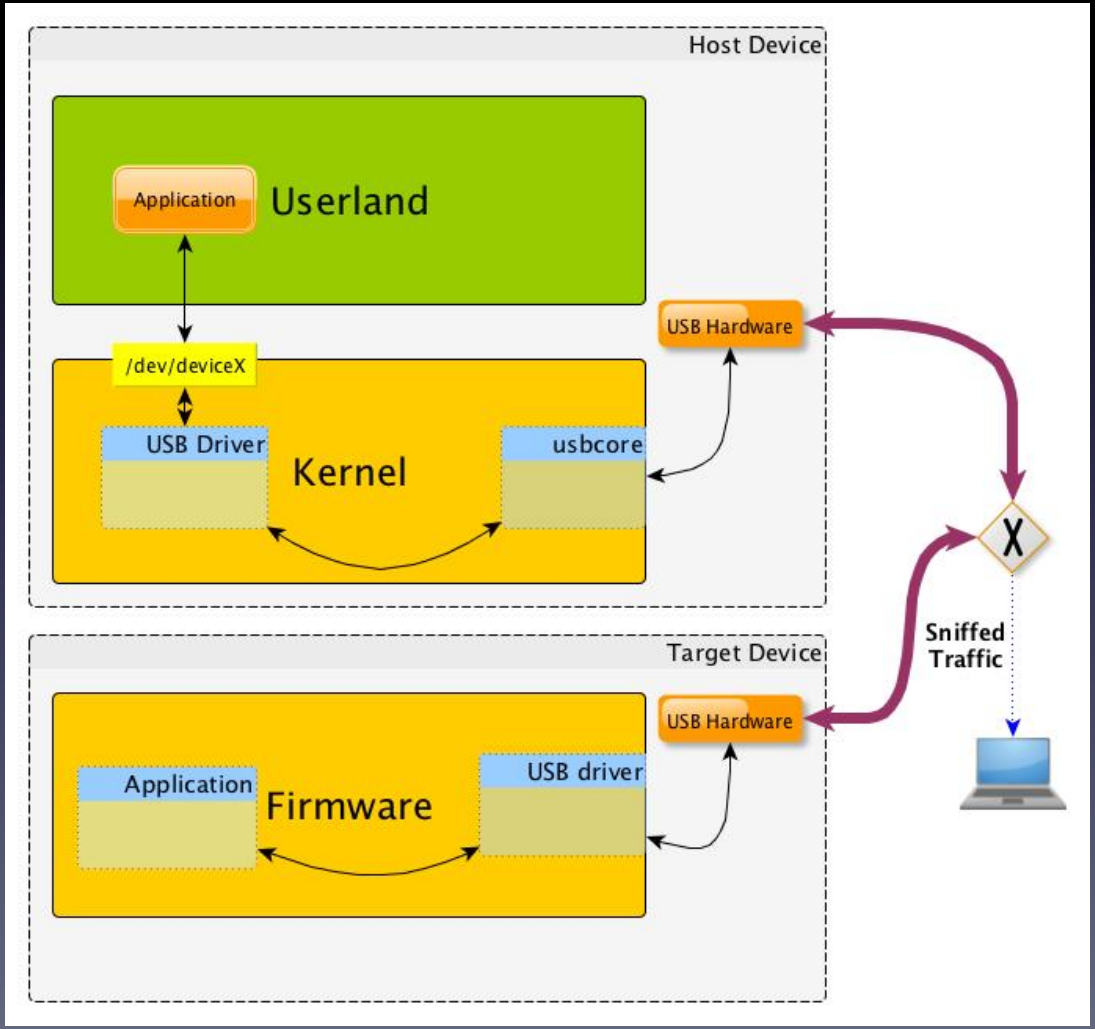
Special PID (XXoo)

Special	PRE	1100B	(Token) Host-issued preamble. Enables downstream bus traffic to low-speed devices.
	ERR	1100B	(Handshake) Split Transaction Error Handshake (reuses PRE value)
	SPLIT	1000B	(Token) High-speed Split Transaction Token (see Section 8.4.2)
	PING	0100B	(Token) High-speed flow control probe for a bulk/control endpoint (see Section 8.5.1)
	Reserved	0000B	Reserved PID

*Note: PID bits are shown in MSb order. When sent on the USB, the rightmost bit (bit 0) will be sent first.

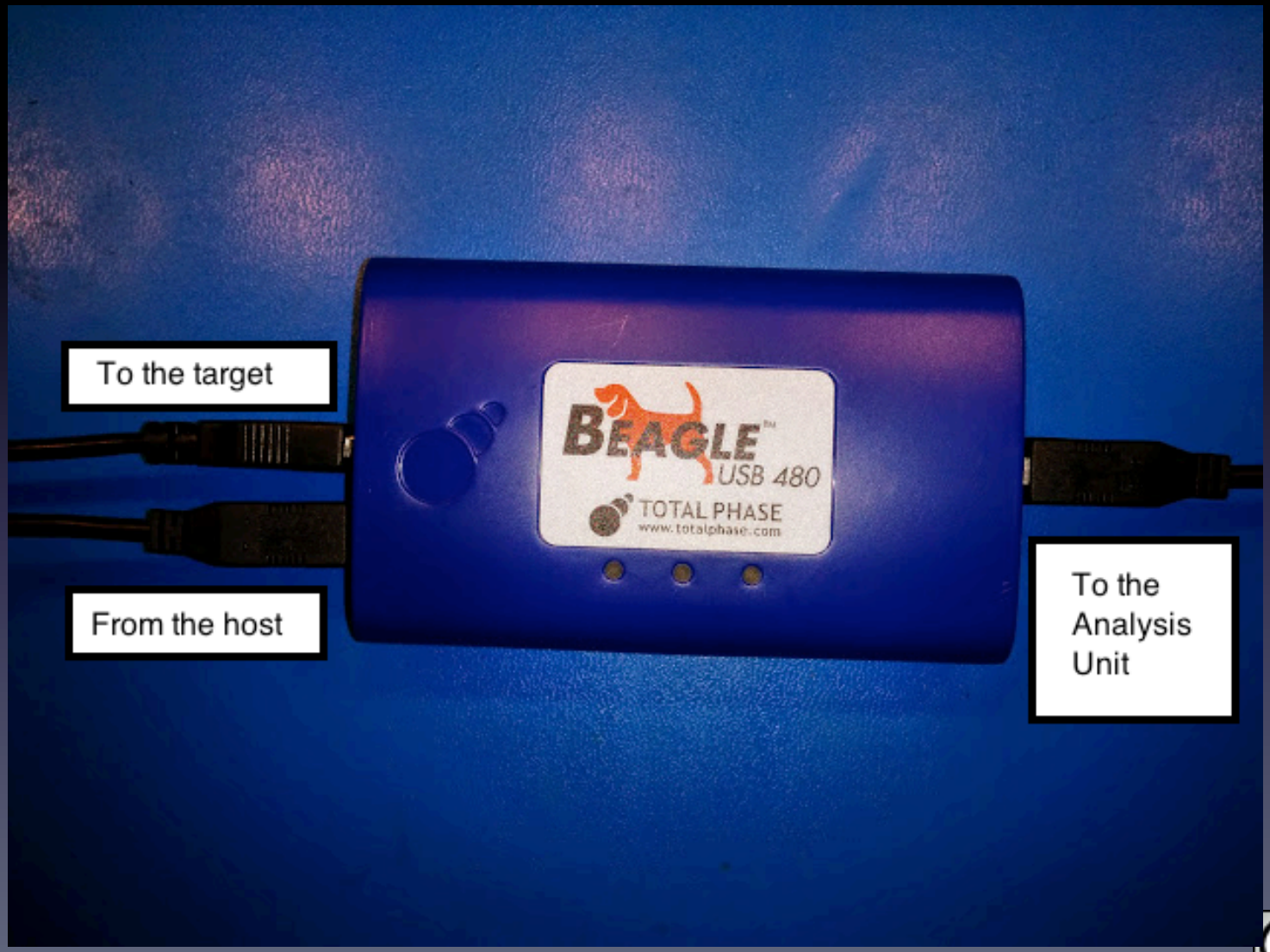
- from **Universal Serial Bus Specification Revision 2.0** Table 8-1 at http://www.usb.org/developers/docs/usb20_docs/

How we are going to look at the USB packets



- We use a hardware sniffer to grab data in the middle of the wire
- In this case a Totalphase Beagle 480 USB
- The resulting sniffed data is sent to a separate device via a 3rd USB port

Wiring up a Hardware Sniffer



The First Transactions

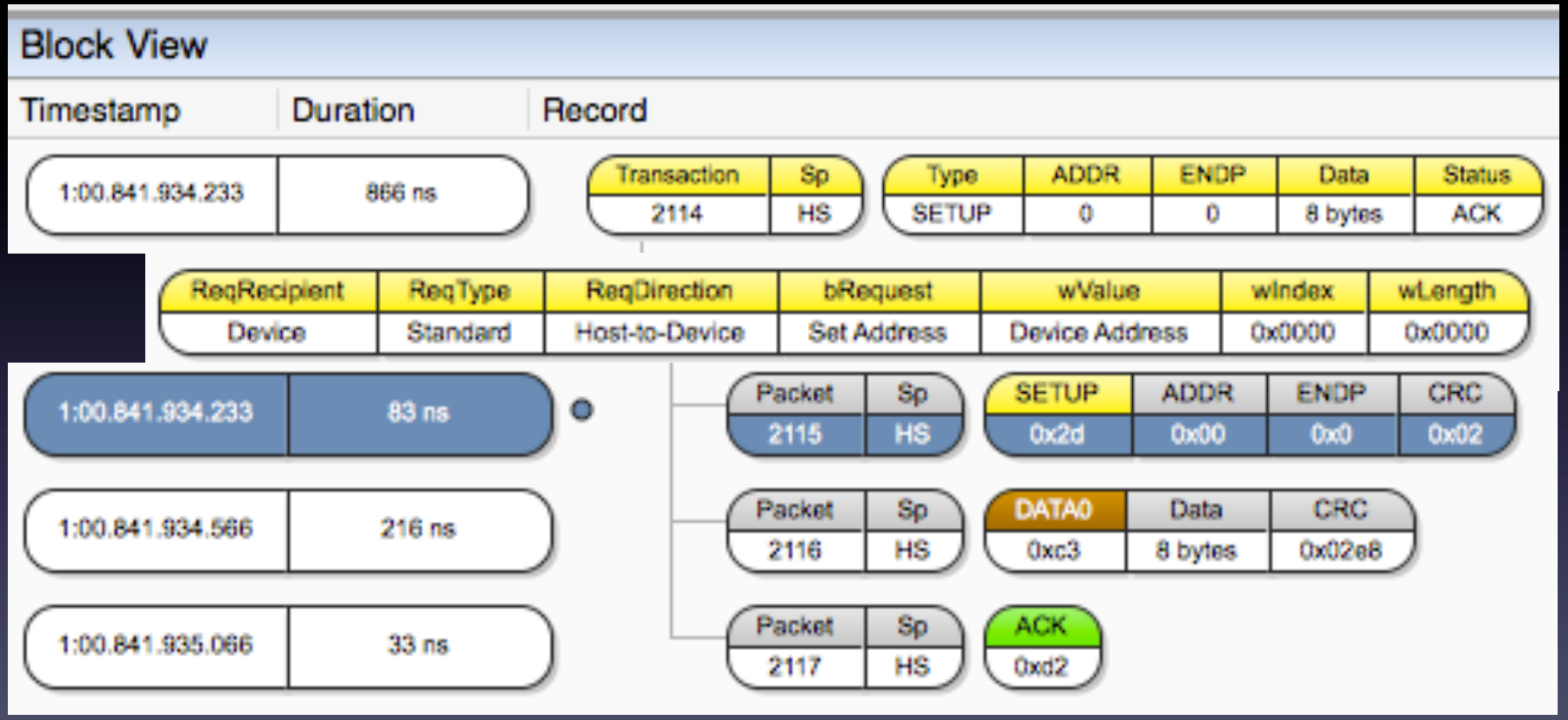
Sp	Index	m:s.ms.us	Len	Err	Dev	Ep	Record	Summary
HS	2111	1:00.785.400	227 us				<Reset>	
HS	2112	1:00.785.628					<High-speed>	
HS	2113	1:00.785.628	56.2 ms				[451 SOF]	[Frames: 1004.x - 1060.3]
HS	2114	1:00.841.934	8 B		00	00	SETUP txn	00 05 16 00 00 00 00 00
HS	2118	1:00.842.010	66 ns				[1 SOF]	[Frame: 1060.4]
HS	2119	1:00.841.935	0 B		00	00	IN txn [5 POLL]	
HS	2124	1:00.842.135	15.6 ms				[126 SOF]	[Frames: 1060.5 - 1076.2]

- first, hardware handshaking
- then the first transactions
  - SOF
  - Setup
  - IN
- all this on Device 0 and endpoint 0
  - where is our source and destination address like in Ethernet?

Obtaining An Address

- All devices are assigned an address on the network by the host
- The system initially talks on device 0 to endpoint 0 to set up communications
- This allows the host to assign it an device number on the bus

Setup



Address 0, endpoint 0 (the host) wants to set an address 0x16 (22 decimal on the device)

How a target gets a address

HS	2111	1:00.785.400	227 us			<Reset>	
HS	2112	1:00.785.628				<High-speed>	
HS	2113	1:00.785.628	56.2 ms			[451 SOF]	[Frames: 1004.x - 1060.3]
HS	2114	1:00.841.934	8 B	00	00	SETUP txn	00 05 16 00 00 00 00 00
HS	2118	1:00.842.010	66 ns			[1 SOF]	[Frame: 1060.4]
HS	2119	1:00.841.935	0 B	00	00	IN txn [5 POLL]	
HS	2124	1:00.842.135	15.6 ms			[126 SOF]	[Frames: 1060.5 - 1076.2]
HS	2125	1:00.857.803	8 B	22	00	SETUP txn	80 06 00 01 00 00 12 00
HS	2129	1:00.857.887	250 us			[3 SOF]	[Frames: 1076.3 - 1076.5]
HS	2130	1:00.857.807	18 B	22	00	IN txn [17 POLL]	12 01 00 02 00 00 00 40 51
HS	2135	1:00.858.207	0 B	22	00	OUT txn	

00000000b	SET_ADDRESS (5)	Device Address	Zero	Zero

- Here you see the Setup packet containing a SET_ADDRESS command (bRequest = 5) and assigning the address of 22 (wValue = 0x16)
- The second setup packet show the new address in use

What happens next?

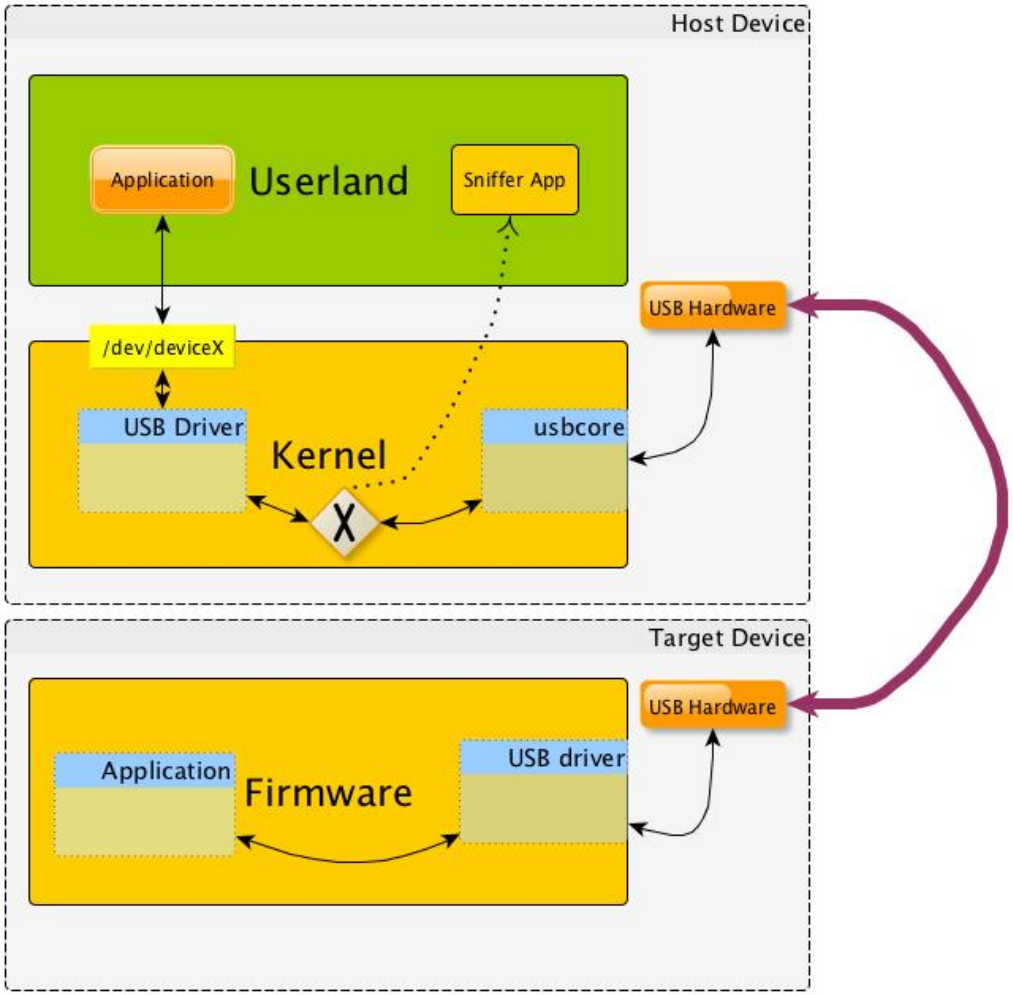
- There will be a bunch of new transactions that transfer what the target device is in both a abstract (I'm a mass storage device) and concrete (I'm a Kingston DataTraveler II) sense

Do we care about hardware handshakes and power?

Sp	Index	m:s.ms.us	Len	Err	Dev	Ep	Record	Summary
HS	2111	1:00.785.400	227 us				<Reset>	
HS	2112	1:00.785.628					<High-speed>	
HS	2113	1:00.785.628	56.2 ms				[451 SOF]	[Frames: 1004.x - 1060.3]
HS	2114	1:00.841.934	8 B		00	00	SETUP txn	00 05 16 00 00 00 00 00
HS	2118	1:00.842.010	66 ns				[1 SOF]	[Frame: 1060.4]
HS	2119	1:00.841.935	0 B		00	00	IN txn [5 POLL]	
HS	2124	1:00.842.135	15.6 ms				[126 SOF]	[Frames: 1060.5 - 1076.2]

- The green data is about the electronic connection
- You need a hardware sniffer to see this

A second way of dumping data



- we can also get a look at the data by intercepting the data in the Linux usbmon kernel module
- No fancy hardware required

wireshark and tshark can do usb

No.	Time	Source	Destination	Protocol	Length	Info
11	0.000028000	1.0	host	USB	82	GET DESCRIPTOR Response DEVICE
12	0.000035000	1.1	host	USB	64	URB_INTERRUPT in

```
▶Frame 1: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface 0
▼USB URB
  URB id: 0xffff880402b47c00
  URB type: URB_SUBMIT ('S')
  URB transfer type: URB_CONTROL (0x02)
 ▶Endpoint: 0x80, Direction: IN
  Device: 1
  URB bus id: 3
  Device setup request: relevant (0)
  Data: not present ('<')
  URB sec: 1418344504
  URB usec: 821973
  URB status: Operation now in progress (-EINPROGRESS) (-115)
  URB length [bytes]: 4
  Data length [bytes]: 0
  [Response in: 2]
  [bInterfaceClass: Unknown (0xffff)]
▼URB setup
 ▶bmRequestType: 0xa3
  bRequest: GET_STATUS (0x00)
 ▶wValue: 0x0000
 ▶wIndex: 1
  wLength: 4
```

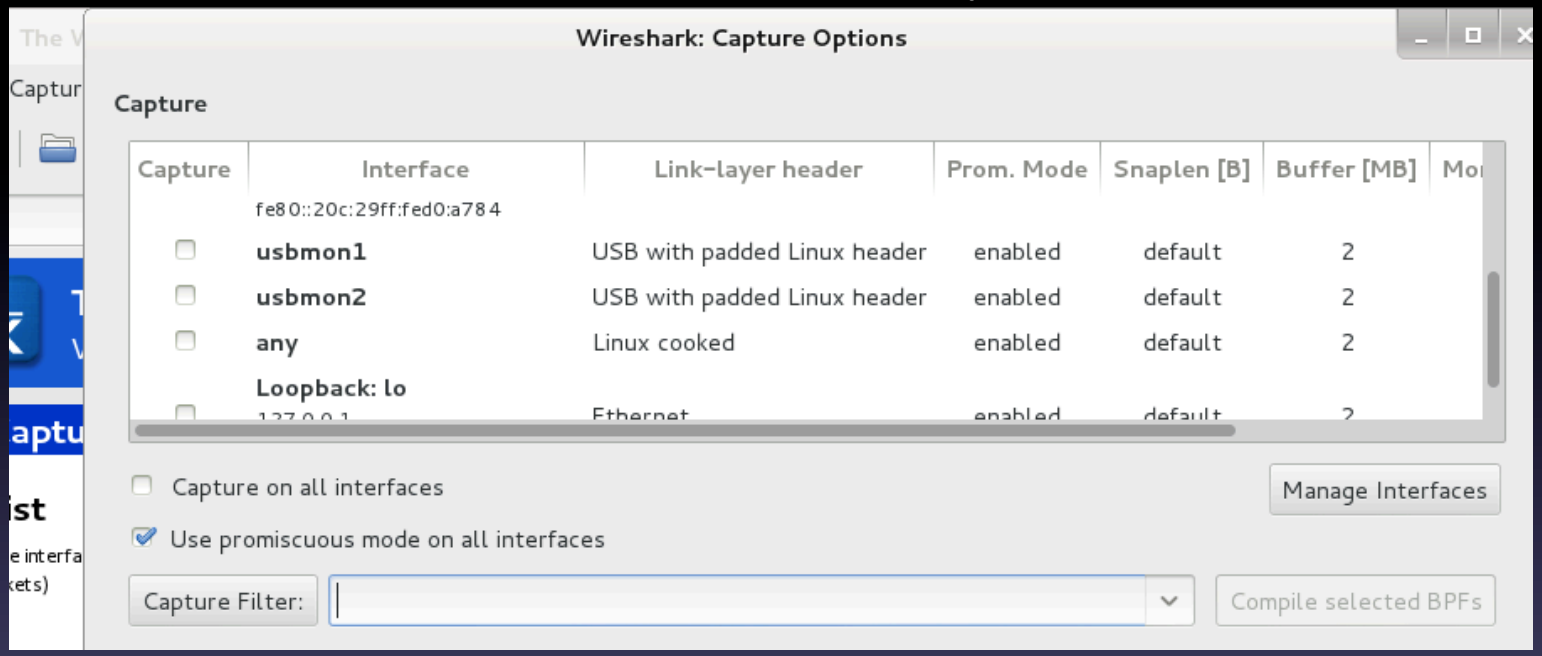
USB dumping with wireshark

- On Linux you need to install the usbmon kernel module

  sudo modprobe usbmon

- it is standard on most modern kernels
- the latest wireshark versions support sniffing USB via usbmon

USB dumping with wireshark



- On windows you need to use USBPcap to capture USB traffic in wireshark
  - http://desowin.org/usbpcap/

Dumps from VMWare

- Edit the .vmx to say

  monitor = "debug"

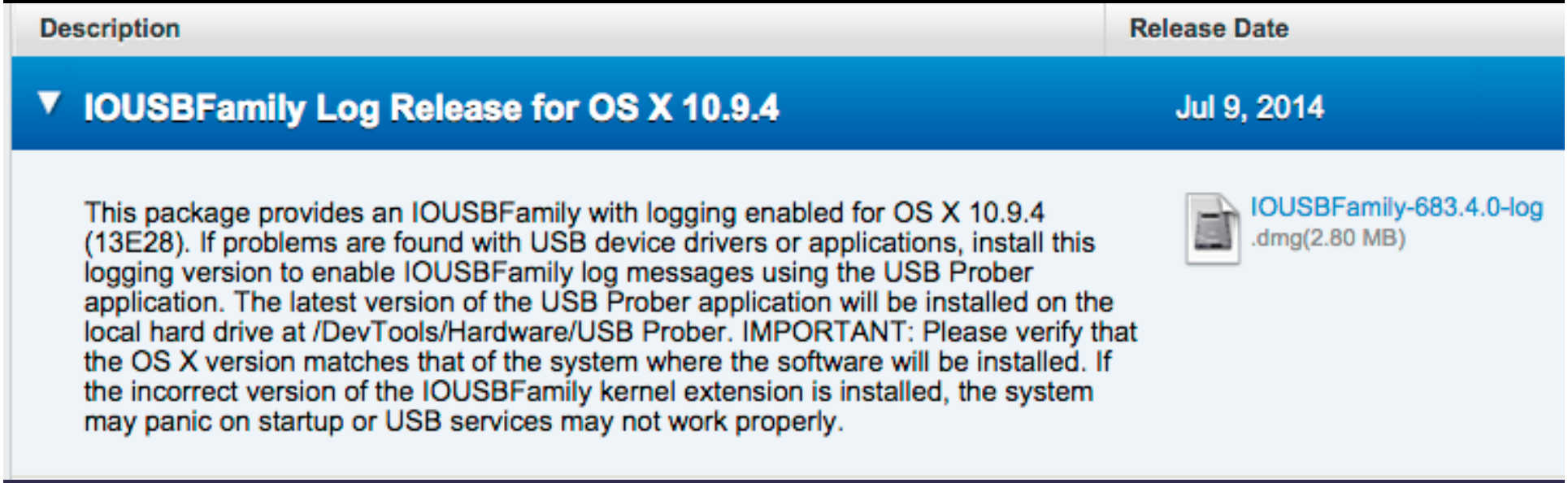
  usb.analyzer.enable = TRUE

  usb.analyzer.maxLine = 8192

  mouse.vusb.enable = FALSE

- Use vusb-analyzer to read the dumps
  - http://vusb-analyzer.sourceforge.net/tutorial.html
  - Supported log formats:
    - VMware VMX log file (*.log)
    - Exported XML from Ellisys Visual USB (*.xml)
    - Linux usbmon log, "1u" (raw text data) format (*.mon)

Dumps from OSX



Description	Release Date
▼ **IOUSBFamily Log Release for OS X 10.9.4**	Jul 9, 2014
This package provides an IOUSBFamily with logging enabled for OS X 10.9.4 (13E28). If problems are found with USB device drivers or applications, install this logging version to enable IOUSBFamily log messages using the USB Prober application. The latest version of the USB Prober application will be installed on the local hard drive at /DevTools/Hardware/USB Prober. IMPORTANT: Please verify that the OS X version matches that of the system where the software will be installed. If the incorrect version of the IOUSBFamily kernel extension is installed, the system may panic on startup or USB services may not work properly.	IOUSBFamily-683.4.0-log .dmg(2.80 MB)

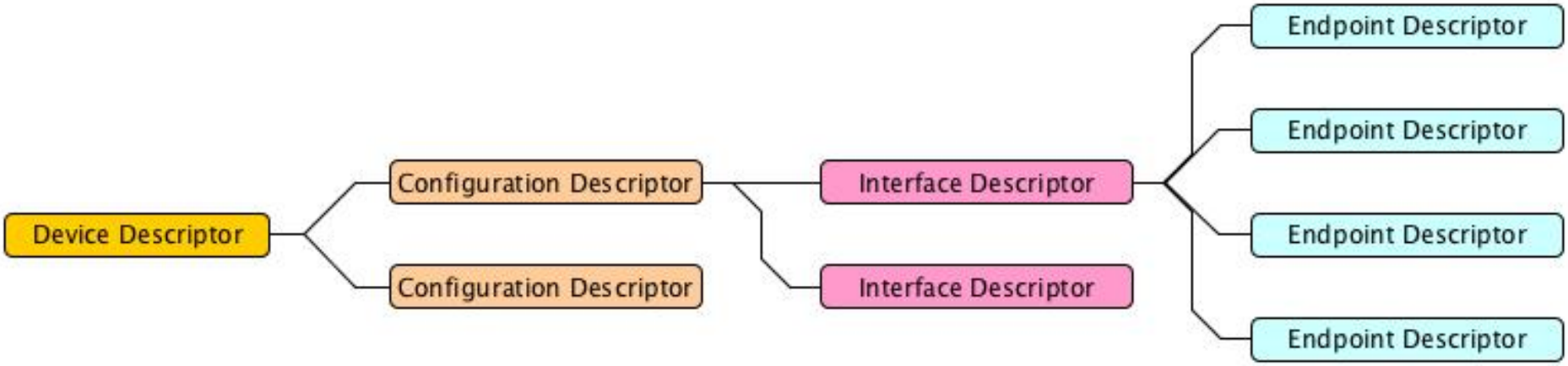
- Supposedly USBProber can do this but we are on version 10.10.1 and I'm not installing a mis-matched kext
- this is the latest version as of December 2014

Collecting data aside, what is next?

```
▼Full Speed device @ 1 (0x80300000): ..............  Composite device: "USB PnP Sound Device"
  ▶Port Information:                                  0x201a
  ▼Number Of Endpoints (includes EP0):
      Total Endpoints for Configuration 1 (current):  4
  ▼Device Descriptor
      Descriptor Version Number:                      0x0110
      Device Class:                                   0   (Composite)
      Device Subclass:                                0
      Device Protocol:                                0
      Device MaxPacketSize:                           8
      Device VendorID/ProductID:                      0x0D8C/0x013A   (C-MEDIA ELECTRONICS INC.)
      Device Version Number:                          0x0100
      Number of Configurations:                       1
      Manufacturer String:                            1 "C-Media Electronics Inc.      "
      Product String:                                 2 "USB PnP Sound Device"
      Serial Number String:                           0 (none)
  ▼Configuration Descriptor (current config)
    ▶Length (and contents):                           253
      Number of Interfaces:                           4
      Configuration Value:                            1
      Attributes:                                     0x80 (bus-powered)
      MaxPower:                                       100 mA
    ▶Interface #0 - Audio/Control
    ▶Interface #1 - Audio/Streaming
    ▶Interface #1 - Audio/Streaming (#1)
    ▶Interface #2 - Audio/Streaming
    ▶Interface #2 - Audio/Streaming (#1)
    ▶Interface #3 - HID
```

We need to tackle this concept next

Hierarchy of Descriptors



- These are the key data structures that define what we can fuzz
- You should have a handout containing the pages of the USB 2.0 spec that cover the fields
- The key takeaway is - One Physical device can have more than one function and therefore more than one driver

Endpoints

- A single USB device can actually set-up a variety of channels (called endpoints) within an address
- All end points are one direction only
  - They are designated either
    - OUT = host -> device
    - IN = device -> host

Endpoint Types

- Control
  - configuration data, status, information about the device
  - always endpoint 0
  - guaranteed delivery

Endpoint Types

- Interrupt
  - set size reply whenever the host asks
  - small amounts of data (think HID devices)
  - guaranteed delivery

Endpoint Types

- Bulk and Isochronous
  - transfer large amounts of data
  - Isochronous is used for situations where the system can handle loss of data
    - audio/video streams

Endpoint Types

- Interrupt and Isochronous are polled a regular intervals
  - bandwidth is reserved for them
- Control and bulk are used for asynchronous data

Descriptors

- A group of endpoints are bundled into an interface
  - interfaces have only one type of function
  - meaning an interface maps to a single driver
  - a physical device can have more than one function
    - meaning that one physical device can be multiple descriptors and therefore lead to the loading of more than one driver

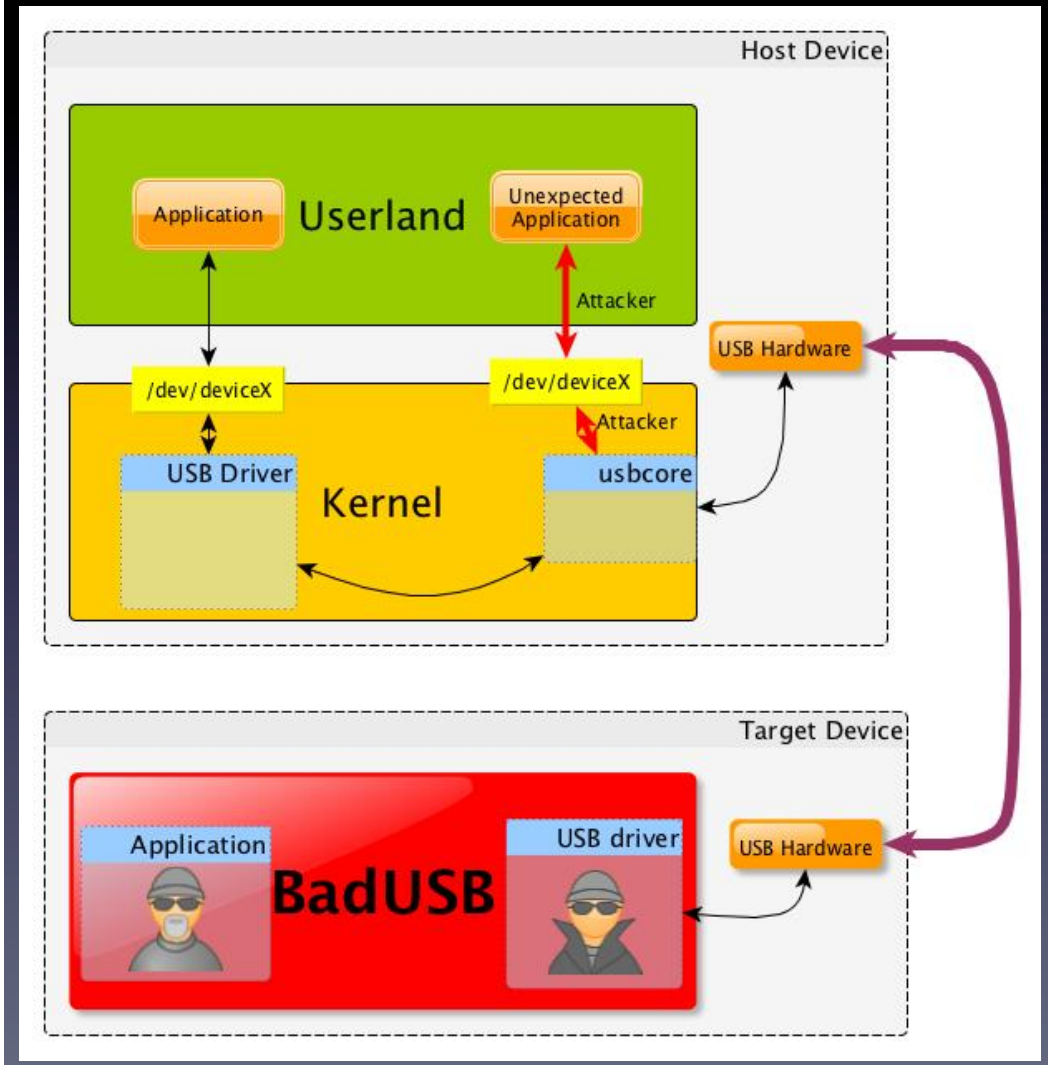
Clarification

- at this point you may be a tad confused about descriptors vs. hosts on the network
- a descriptor is used to define multiple functions of one physical device
  - ex. a printer/fax/scanner
  - we will see why this is dangerous in the description of BadUSB
- a hub allow you to have multiple physical devices attached to a single port

Important Data Used for Enumeration contained in a Device Descriptor

- bDeviceClass, bDeviceSubClass, bDeviceProtocol
  - These tell us what type of device it is
  - HID – keyboard, mass storage, camera
- idVendor, idProduct
  - These tell us who made the device and the model
  - this determines what driver is loaded

BadUSB



- Attack the device firmware and replace it
- Suddenly the the USB device has new functions the user did not expect

BadUSB

- The concept that a single physical device can have multiple interfaces and endpoints opens up interesting possibilities
- What if a malicious device could have hidden functions and interface descriptors?

BadUSB Takeaways

- By not locking the firmware to the device the manufacturer left it open to reprogramming
- The attacks actually use the protocol as it was intended and just install additional functionality
- The key it what that additional functionality can do in the delivery of malware and unexpected behavior

Bad USB

- Karsten Nohl and Jakob Lell introduced this attack at BlackHat USA 2014
  - Presentation: https://srlabs.de/blog/wp-content/uploads/2014/07/SRLabs-BadUSB-BlackHat-v1.pdf
  - Video: https://www.youtube.com/watch?v=nuruzFqMgIw

Attacking Systems via USB

So there I was…

- Way back when I started playing with USB I had a Beagle 480 and a Teensy
- Then I saw Andy Davis present at Defcon 21
  - https://www.defcon.org/html/links/dc-archives/dc-21-archive.html#Davis
- When I spoke to him after his talk he pointed me to Travis Goodspeed's facedancer21 board

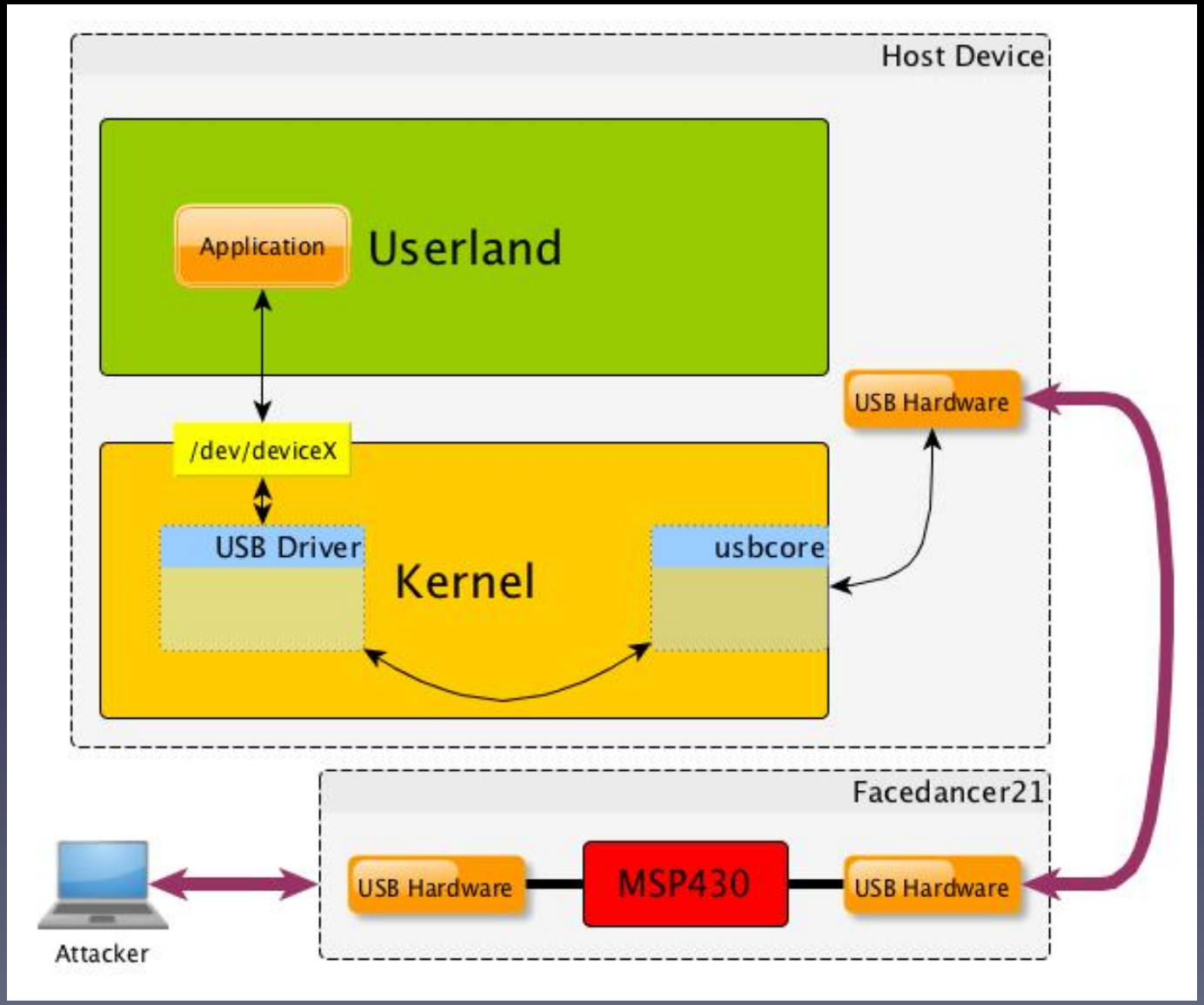
The Facedancer21 board

- From the same guy who brought us the GoodFET
- Totally open design
  - http://goodfet.sourceforge.net/hardware/facedancer21/
- Travis has presented many times on the boards use
  - https://www.youtube.com/watch?v=x-7ezoFju6I

Getting closer

- The facedancer21 board is much more interactive and useful for our fuzzing
- It has the ability to down the USB port, change identity, then come back up as a whole new device
  - unlike a teensy
- accessible in python
  - Emulating USB Devices with Python Blog Entry
  - http://travisgoodspeed.blogspot.com/2012/07/emulating-usb-devices-with-python.html
- The source even include a scapy interface

The facedancer hardware allows full hardware control of the interactions with the Host



Problem one – building a facedancer

- Open designs are great and the PCBs are available for $5 from Travis
- But the guys at int3.cc (who are really those fun hardware people at http://www.xipiter.com) will sell you an already assembled board

Facedancer software

- All the software we need is available on github –
- To install it

```
git clone https://github.com/travisgoodspeed/goodfet
goodfet
cd goodfet/client
sudo make link
```

User contributed software

- There is also a basic set of scapy bindings available in
- goodfet/contrib/facedancer/scapy

- This is a good start for basic fuzzing of USB from the outside but it is not accessible to the masses yet…

We have a tool and software

- People do use it this way, Daniel Mende used this and his dizzy fuzzing software to do the work he presented at Troopers 2014
  - Implementing an USB Host Driver Fuzzer
  - https://www.youtube.com/watch?v=h777lF6xjs4

Loading the facedancer firmware

- If you install all the MSP430 build dependencies you can compile the firmware locally and upload that
  - but using the –fromweb option just grabs the latest code from the Internet
- Video of how it should look:
  - programming_the_facedancer.mp4

Loading the facedancer firmware

- This works for all the goodfet boards including our facedancer21
- the code is in goodfet/client/ for your inspection

```
export board=facedancer21
./goodfet.bsl --dumpinfo > info.txt # just in case we need it
./goodfet.bsl --fromweb
```

The next layer of software

- Remember when I mentioned Andy Davis and his Defcon presentation?
  - It took a while but he posted the library to github
  - github umap page - https://github.com/nccgroup/umap
  - Revealing Embedded Fingerprints: Deriving intelligence from USB stack interactions
    - https://www.defcon.org/html/links/dc-archives/dc-21-archive.html
  - Lessons learned from 50 bugs: Common USB driver vulnerabilities
    - https://www.nccgroup.com/media/190706/usb_driver_vulnerabilities_whitepaper_january_2013.pdf

umap.py opens USB fuzzing up to the talented amateur

```
--------------------------------------
 _   _ _ __ ___   __ _ _ __
| | | | '_ ` _ \ / _` | '_ \
| |_| | | | | | | (_| | |_) |
 \__,_|_| |_| |_|\__,_| .__/
                      |_|

The USB host assessment tool
Andy Davis, NCC Group 2013
Version: 1.03

Based on Facedancer by Travis Goodspeed

For help type: umap.py -h
--------------------------------------

Usage: umap.py

Options:
  --version    show program's version number and exit
  -h, --help   show this help message and exit
```

- umap.py is a basic framework for USB fuzzing and enumeration
- It is a Python3 application
  - it also needs python3-serial
- Makes basic fuzzing easy for a beginner

Recon Tools

```
-L          List device classes supported by umap
-i          identify all supported device classes on connected host
-c CLS      identify if a specific class on the connected host is supported
            (CLS=class:subclass:proto)
-O          Operating system identification
```

- It brings us a framework of basic recon tools
- If you want to add your own, umap IDs device classes via the data in
  - device_class_data.py

umap device descriptor scanning

- python3 umap.py –P <facedancer port> -i
  - scan the host for supported devices
  - Video (umap side)
    - umap_scan_-i.mp4
  - Video (host side)
    - device_scan.mp4 (too fast it kills the host unit)
    - slow_device_scan.mp4 (using –d 3 to slow the rate of scanning and allow the scan to finish)

umap device class scanning

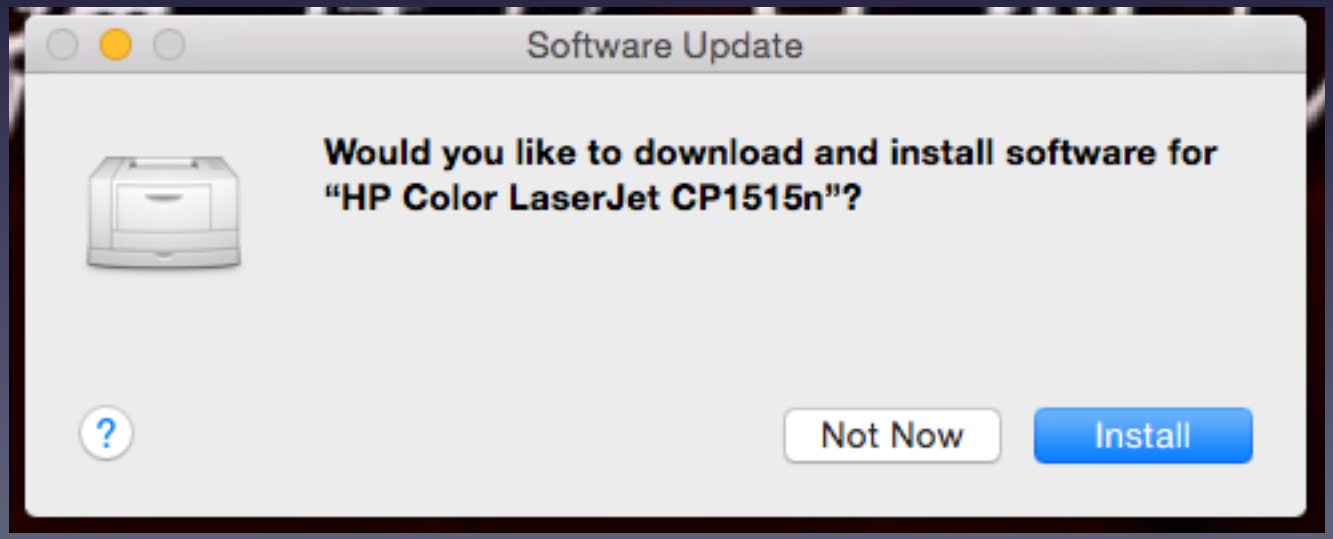
- So here we are asking if the host supports this device type
  - regardless of manufacturer
  - that is one type of driver loaded
- You can see in the video how scanning too fast gave bad results
  - we need to use –d 3 to slow the scan down

The pain of automatic driver loading

- In your test and the video we see that Linux just loads the drivers as they are needed
  - no user confirmation is required
- Windows and OSX will do this too for known (signed in some cases) drivers

If the correct driver is not on the system

- You get pop-ups
- This can mess you device class enumeration scan if you are not aware of the pending dialogs on the host device



Fingerprinting a system

- How a system reacts to device class enumeration and other factors can server as a fingerprint to the running OS
- This scan looks very similar to the device scan but does not cover all devices and adds a few deeper device checks

umap OS detection

- python3 umap.py –P <facedancer port> -O
- The file umap-device-fingerprints.json contains the checks that are done if you cant to add your own
- Video (umap side)
  - umap_OS_detection.mp4
- Video (host side)
  - os_scan.mp4

Fuzzing Tools

```
-e DEVICE     emulate a specific device (DEVICE=class:subclass:proto)
-n            Start network server connected to the bulk endpoints (TCP port
              2001)
-v VID        specify Vendor ID (hex format e.g. 1a2b)
-p PID        specify Product ID (hex format e.g. 1a2b)
-r REV        specify product Revision (hex format e.g. 1a2b)
```

- You can choose to test a specific device (vendor/product/version) or device class (class, subclass, protcol)
- Very useful with the network server option to tunnel traffic through USB to an endpoint

Fuzzing Tools

```
-f FUZZC      fuzz a specific class (FUZZC=class:subclass:proto:E/C/A[:start
              fuzzcase])
-s FUZZS      send a single fuzz testcase
              (FUZZS=class:subclass:proto:E/C:Testcase)
```

- umap also has a simple fuzzing engine and lets you run all the tests or pick just one
- the fields to fuzz are in
- umap/testcases.py
- You should fine a strong correspondence between this list and the printouts from the the USB Spec

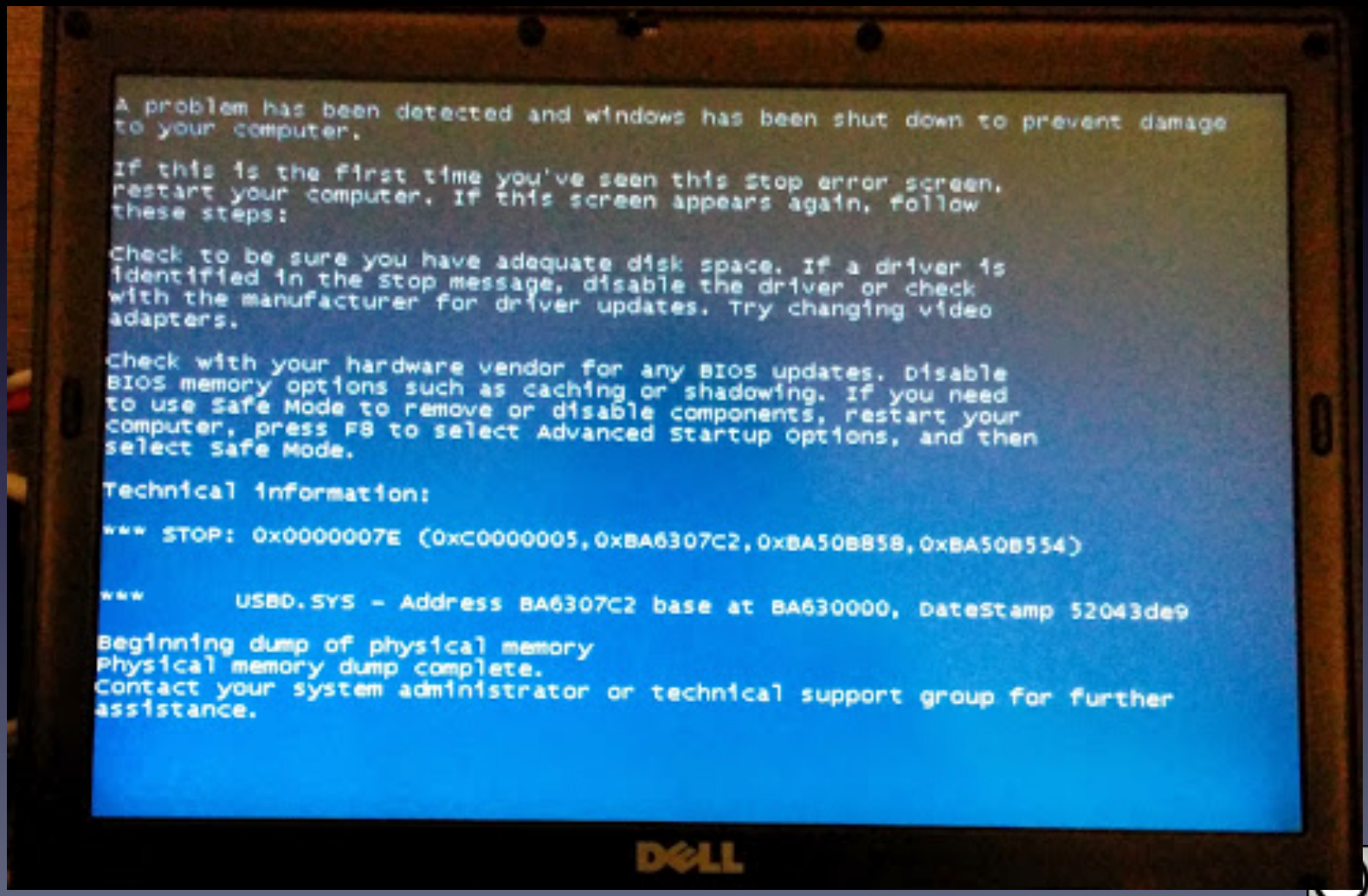
Fuzzing run example

- Video (umap side)
  - umap_HID_fuzz.mp4
- Video (host side)
  - faster_audio_scan.mp4

Crashing

- Fuzzing is never a fast process and pretty boring to watch
- Mass scanning is not always fun
- You need to monitor the target system for crashes and restarts of running processes
  - I keep a watch on the pid of the process I'm testing because sometimes they have watchdogs that restart them

Until you see this…



Converting Crashes to Exploits

- While beyond the scope of what we are discussing today there are many good resources available to help you with this
- But remember that a reproducible crash is often sufficient to prove a problem

Who to talk to…

- report driver issues to the right people
  - I'm sure that Microsoft is tired of having issues in 3rd party drivers they did not distribute reported to them
  - Remember that USB drivers are often layered with different groups owning different parts of the systems

Some Excellent Free Resources

- Check out the courses at
  - http://opensecuritytraining.info
  - https://www.corelan.be/index.php/category/security/exploit-writing-tutorials/
- Seattle Area Groups
  - Neg9 - https://neg9.org/
  - Black Lodge Research - https://www.blacklodgeresearch.org/

Thanks

I hope this was interesting and let me know if you have any questions!