You wouldn’t share a syringe. Would you share a USB port?

Travis Goodspeed, Sergey Bratus
Thank you kindly

- Searchio
- Dmitry Nedospasov
- Shout-out:
  Andy Davis “50 Lessons learned from USB bugs”
Wright’s Law

“Security doesn’t get better until tools for practical exploration of the attack surface are made available” - Joshua Wright
Which port is scarier?
“It’s all a network!”

* **Networks:**
  * packets are routed based on data in them
  * have layers of abstraction (OSI)
  * we scan them for vulnerable endpoints
  * we inject crafted packets into them

* **Buses:**
  * well... *all of the above?*
Which stack is higher?
More brittle stacks, angrier packets
These birds are so damn angry

Angry birds glorify attackers!

To improve cyber, we need "Peaceful Pigs Building Solid Defensive Structures"

Those birds are so damn angry.
Not your tame TCP/IP birds...

**IP Header**

- **Field**: IP version (4 bits), Hdr Len (16 bits), Type of Service (TOS) (8 bits), Total Length (16 bits), Identification (Fragment ID) (16 bits), Fragment Offset (13 bits), Time-To-Live (TTL) (8 bits), Protocol (8 bits), Header Checksum (16 bits), Source IP Address (32 bits), Destination IP Address (32 bits), Options (if any, variable length, padded with 0's, 40 bytes max length), Data

**TCP Header**

- **Field**: Source Port Number (16 bits), Destination Port Number (16 bits), Sequence Number (32 bits), Acknowledgement Number (32 bits), Header Length (4 bits), Reserved (6 bits), Window Size (16 bits), Urgent Pointer (16 bits), Options (if any, variable length, padded with 0's), Data (if any)

**Table**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bLength</td>
<td>18</td>
<td>Valid Length</td>
</tr>
<tr>
<td>bDescriptorType</td>
<td>1</td>
<td>DEVICE</td>
</tr>
<tr>
<td>bcdUSB</td>
<td>0x0200</td>
<td>Spec Version</td>
</tr>
<tr>
<td>bDeviceClass</td>
<td>0xEF</td>
<td>Miscellaneous</td>
</tr>
<tr>
<td>bDeviceSubClass</td>
<td>0x02</td>
<td>Common Class</td>
</tr>
<tr>
<td>bDeviceProtocol</td>
<td>0x01</td>
<td>Interface Association Descriptor</td>
</tr>
<tr>
<td>bMaxPacketSize0</td>
<td>64</td>
<td>Max EP0 Packet Size</td>
</tr>
<tr>
<td>idVendor</td>
<td>0x046D</td>
<td>Logitech Inc.</td>
</tr>
<tr>
<td>idProduct</td>
<td>0x0821</td>
<td>Unknown</td>
</tr>
<tr>
<td>bcdDevice</td>
<td>0x0010</td>
<td>Device Release No</td>
</tr>
<tr>
<td>iManufacturer</td>
<td>0</td>
<td>Index to Product Manufacturer (none)</td>
</tr>
<tr>
<td>iProduct</td>
<td>0</td>
<td>Index to Product String (none)</td>
</tr>
<tr>
<td>iSerialNumber</td>
<td>1</td>
<td>Index to Serial Number String</td>
</tr>
<tr>
<td>bNumConfigurations</td>
<td>1</td>
<td>Number of Possible Configurations</td>
</tr>
</tbody>
</table>
const unsigned char CD[] =
{
    0x09,
    0x02,
    0x22, 0x00,
    0x01,
    0x01,
    0x00,
    0xE0,
    0x01,
    // CONFIGURATION Descriptor
    // bLength
    // bDescriptorType = Config
    // wTotalLength(L/H) = 34 bytes
    // bNumInterfaces
    // bConfigValue
    // iConfiguration
    // bmAttributes. b7=1 b6=self-powered b5=RWU supported
    // MaxPower is 2 ma

    // INTERFACE Descriptor
    0x09,
    0x04,
    0x00,
    0x00,
    0x01,
    // bAlternate Setting
    0x03,
    0x00, 0x00,
    // bInterfaceClass = HID
    // bInterfaceSubClass, bInterfaceProtocol
    0x00,
    // iInterface

    // HID Descriptor--It's CD[18]
    0x09,
    0x21,
    0x10, 0x01,
    0x00,
    0x01,
    // bLength
    // bDescriptorType = HID
    // bcdHID(L/H) Rev 1.1
    // bCountryCode (none)
    // bNumDescriptors (one report descriptor)
    0x22,
    // bDescriptorType (report)
    43, 0,
    // CD[25]: wDescriptorLength(L/H) (report descriptor size is 43 bytes)

    // Endpoint Descriptor
    0x07,
    0x05,
    0x83,
    0x03,
    // bLength
    // bDescriptorType (Endpoint)
    // bEndpointAddress (EP3-IN)
    // bmAttributes (interrupt)
    64, 0,
    // wMaxPacketSize (64)
    10,
    // bInterval (poll every 10 msec)
Guess the parser bug

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bLength</td>
<td>52</td>
<td>Descriptor length (including the bLength field)</td>
</tr>
<tr>
<td>bDescriptorType</td>
<td>3</td>
<td>String descriptor</td>
</tr>
<tr>
<td>bString</td>
<td>“HP Color LaserJet CP1515n”</td>
<td>The string to be stored (in Unicode format i.e. two bytes per character)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bLength</td>
<td>9</td>
<td>Descriptor length (including the bLength field)</td>
</tr>
<tr>
<td>bDescriptorType</td>
<td>2</td>
<td>Configuration descriptor</td>
</tr>
<tr>
<td>wTotalLength</td>
<td>55</td>
<td>Total combined size of this set of descriptors</td>
</tr>
<tr>
<td>bNumInterfaces</td>
<td>2</td>
<td>Number of interfaces supported by this configuration</td>
</tr>
<tr>
<td>bConfigurationValue</td>
<td>1</td>
<td>Value to use as an argument to the SetConfiguration() request to select this configuration</td>
</tr>
<tr>
<td>iConfiguration</td>
<td>0</td>
<td>Index of String descriptor describing this configuration</td>
</tr>
<tr>
<td>bmAttributes (Self-powered)</td>
<td>1</td>
<td>Self-powered</td>
</tr>
<tr>
<td>bmAttributes (Remote wakeup)</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>bmAttributes (Other bits)</td>
<td>0x80</td>
<td>Valid</td>
</tr>
<tr>
<td>bMaxPower</td>
<td>2mA</td>
<td>Maximum current drawn by device in this configuration</td>
</tr>
</tbody>
</table>
What's behind a USB port?

USB port
controller (eg, EHCI)
USB bus
PCI bus
system root bus (virtual)
A lot hangs on these wires
System programmer view

Filesystems

CAM

SCSI ATA umass

UHCI OHCI EHCI XHCI

IO Syscall

CAM_action callback

Translates from CCB to command protocol, run state machine for wire protocols, sets up bus Xfers

Handles Xfers

DMA, interrupts
Port-side view

- All kinds of subsystems and drivers are reachable from USB
- “Sanity checks” are haphazard; data is trusted
- “Go anywhere in the kernel”
Through the port, down the rabbit hole

Kernel, view from the outside ↑
Kernel, view from the inside →
Are you firewalling this?

- More targets
- Richer data structures
- Looser code
- Higher privilege (Kernel/Ring0 until recent userland USB stacks)
“I see dead drivers”

- 1999, conforms to no standards
- Ubuntu includes drivers
- “Works great with Windows ME!”
“APT”
“APT”
“APT”
“APT”
Why aren’t we firewalling that, again?

* Payload delivered over USB can pick any target in the kernel - it will **pick** & **choose** the **loosest** code

* “Sloppy webcam 0.1” driver?

* How easy it is to firewall all the “bad” commands across SCSI, ATAPI, ...?

* **s/Application Firewalls/Driver Firewalls/g**

* ... 

* Profit!
<table>
<thead>
<tr>
<th>USB</th>
<th>Ethernet</th>
<th>Assumption</th>
<th>Violation</th>
<th>Attack Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transfer</td>
<td>One round-trip, maybe NAK-ed</td>
<td><strong>Intended</strong> device will reply to the transfer</td>
<td>Non-compliant controller</td>
<td>Hijack session, change state under the feet of the host</td>
</tr>
<tr>
<td>Transaction</td>
<td>One set of transfers, all but the last NAK-ed</td>
<td>Host controller complies with the USB spec on transactions</td>
<td>Hijack session on disconnect</td>
<td>Use of trusted session context</td>
</tr>
<tr>
<td>Packet</td>
<td>Packet Fragment</td>
<td><strong>Implicit</strong> length of concatenated frames will match <strong>explicit</strong> length of transaction</td>
<td>Non-compliant device</td>
<td>Memory corruption, info leak</td>
</tr>
<tr>
<td>Controller</td>
<td>Ethernet Card</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Bus</td>
<td>D+/D- Pair</td>
<td>Electrically legal signals, but in reality those <strong>widely outside</strong> of spec are accepted</td>
<td>Non-compliant controller</td>
<td>Damage frames for session hijack, jamming</td>
</tr>
</tbody>
</table>
Same-day prototype:
Custom PCB

TravisGoodspeed<travis@tnbelt.com>
+1.865.471.8519
+49.152.23.90.76.92

HOST

U2

U1

MSP430F2618

12MHz

MAX3420E

Facedancer10
http://goodfet.sf.net/

VICTIM
Let’s network them!
The Router/Injector/Facedancer

FROM HOST, RAW PACKET (IN PYTHON)

SPI BUS

USB TO VICTIM

“SEND BUFFER NOW”

Figure 8. The MAX3420E SPI command byte.
Maxim MAX3420E
<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>183</td>
<td>19.444980</td>
<td>host</td>
<td>5.1</td>
<td>USB</td>
<td>179</td>
<td>URB_BULK out</td>
</tr>
<tr>
<td>184</td>
<td>19.445184</td>
<td>5.1</td>
<td></td>
<td>USB</td>
<td>64</td>
<td>URB_BULK out</td>
</tr>
<tr>
<td>185</td>
<td>19.917713</td>
<td>host</td>
<td>5.1</td>
<td>USB</td>
<td>378</td>
<td>URB_BULK out</td>
</tr>
<tr>
<td>186</td>
<td>19.917940</td>
<td>host</td>
<td>5.1</td>
<td>USB</td>
<td>64</td>
<td>URB_BULK out</td>
</tr>
<tr>
<td>187</td>
<td>20.221662</td>
<td>host</td>
<td>5.1</td>
<td>USB</td>
<td>159</td>
<td>URB_BULK out</td>
</tr>
<tr>
<td>188</td>
<td>20.221768</td>
<td>host</td>
<td>5.1</td>
<td>USB</td>
<td>64</td>
<td>URB_BULK out</td>
</tr>
<tr>
<td>189</td>
<td>20.556802</td>
<td>host</td>
<td>5.1</td>
<td>USB</td>
<td>410</td>
<td>URB_BULK out</td>
</tr>
</tbody>
</table>

Device Setup Request: Not Relevant

Data: present (0)
URB sec: 1339611306
URB usec: 912260
URB status: Operation now in progress (-EINPROGRESS) (-115)
URB length [bytes]: 346
Data length [bytes]: 346

[Response in: 190]
[bInterfaceClass: Unknown (0xffff)]
USB glossary

- Ports are called **Endpoints**. EP0 or the SETUP endpoint is for auto-configuration (think a “broadcast address” for setup)

- Unconfigured devices respond to “broadcasts”, send their **Descriptors**

- This setup exchange is called **Enumeration**

- Host assigns device number (~address on the bus)
On the wire with MAX3420

- USB host acquires device descriptors (tables)
- Looks up driver by device/vendor numbers
- Sets up kernel “routing” through the stack layers

Thursday, April 25, 13
NAKS, DEVICE MUST SEND WHILE WORKING ON REPLY TO HOST, OR ELSE HOST DISCONNECTS; LUCKILY, SENT BY MAX 3420 AUTOMAGICALLY
USB devices, in Python

- **Class types** are standardized. (HID, Mass Storage)
- **Vendor types** are not (e.g., FTDI, Wi-Fi).
- Descriptors have structs unique to each device class
- Fairly complex: nested lengths, offsets
  => parser bugs

- Be the host’s worst driver nightmare - in Python: http://goodfet.sf.net/
Facedancer

“If you can write a webserver, you can write a disk”

http://goodfet.sf.net/
Descriptor structs are unique to each device class: **Nested lengths, in-struct offsets = trouble**
Exploiting enumeration

- Host requests the first few bytes of the descriptor.
- Host mallocs that many bytes.
- Host reads the entire descriptor into a temporary buffer.
- Host memcpy() the descriptor into the malloced buffer.

*PSGroove* exploits this on the Playstation 3!
Exploit Dev Cycle
Before & After

1. Change your code.
2. Plug the dongle into your workstation.
3. Reflash it.
4. Move the dongle to your target.
5. Try it.
HID Emulation

- python goodfet.maxusbhid
- Easiest to implement.
- Lots of prior examples,
  - Social Engineering Toolkit
  - **Teensy**, AVR USB Key, vendor examples
- Embarrassing bugs remain!
HID Format String

- Ubuntu 12.04, Xorg

- Manufacturer String:
  "%n%s%n%s%n%s"

- Device String:
  "%n%s%n%s%n%s"

- Thanks to the ChromeOS team!
Skype crashes too

Thursday, April 25, 13
Host Mode Emulation

- Roundtrip time becomes an issue. (Only on OS X)
- Code is already in SVN, hardware coming in FD20.
- Firmware security is even worse than in drivers!
- Most exploits can use **libusb** instead of a Facedancer.
Device Bugs

- Memory exposed by reads past the end of the Strings table.
- Integer overflows, stack smashing, etc.
- Never any ASLR; any DEP is accidental.
Device Firmware Update (DFU)

- Device Firmware Update Protocol
- iPhone, iPod, and other MP3 players.
- Handy attack target.
- Facedancer supported.
The DFU emulator is now running. Any firmware which is downloaded to the virtual device will be locked to this console, beginning with the block device.

Starting a DFU device as FFFF:0004

Defaulting to idle state.
Dear Mr. Goodspeed,

It has come to my attention that you have created a "hacking tool" that may be used to intercept firmware intended for deployment to USB devices and that you have used this tool to capture firmware for my product, Ubonto0h One.

I demand that you cease and desist reverse engineering and publication of technical information relating to Ubonto0h One. The Ubonto0h firmware is open source and may be downloaded freely! I insist that you instead turn your attention to a proprietary technology that is less widely available and understood.

Very sincerely,

Michael Ossmann
Great Scott Gadgets
Mass Storage

- TOCTTOU Exploits
  - See Collin Mulliner’s at WOOT ’12.
- Active Antiforensics
  - Disk erases itself if forensically analyzed.
sudodiff - if=/dev/sdb count=1 bs=512 hd
00000000 e9 86 00 0a 47 6f 6f 64 44 69 73 6b 20 30 2e 30
00000010 31 0a 0d 62 79 20 54 72 65 66 64 0a 0a 0d 00 59
00000020 6f 75 20 68 6f 66 20 72 75 6e 65 6e 6e 0a 0d 00
00000030 62 61 6e 64 69 76 69 73 20 47 6f 6f 20 66 6f
00000040 72 6d 20 6f 66 66 20 73 6f 75 72 69 6e 67 6e
00000050 65 6e 63 65 6e 69 67 68 74 73 0a 0d 00 31 29
00000060 20 52 65 73 70 6c 61 79 73 65 6e 63 65 6e 69
00000070 76 69 73 20 73 74 72 69 6e 67 6e 65 6e 63 65
00000080 6e 69 67 68 74 73 20 73 74 72 69 6e 67 6e 65
00000090 63 65 6e 69 67 68 74 73 0a 0d 00 0e 00 0e 00
000000a0 0e 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00
000000b0 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00
000000c0 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00
000000d0 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00
000000e0 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00
000000f0 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00
00000100 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00
00000110 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00
00000120 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00
00000130 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00
00000140 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00
00000150 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00
00000160 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00
00000170 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00
00000180 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00
00000190 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00
000001a0 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00
000001b0 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00
000001c0 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00
000001d0 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00
000001e0 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00
000001f0 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00 0f 00
1+0 records in
1+0 records out
512 bytes (512 B) copied, 4.8327 s, 0.1 kB/s
pro%

Thursday, April 25, 13
USB Serial Emulation

pro% cat /dev/ttyUSB1

dfET emulates FTDI properly, if you can read this!
GoodFET emulates FTDI properly, if you can read this!
GoodFET emulates FTDI properly, if you can read this!
GoodFET emulates FTDI properly, if you can read this!
GoodFET emulates FTDI properly, if you can read this!
GoodFET emulates FTDI properly, if you can read this!
GoodFET emulates FTDI properly, if you can read this!
GoodFET emulates FTDI properly, if you can read this!
GoodFET emulates FTDI properly, if you can read this!
GoodFET emulates FTDI properly, if you can read this!

pro%
USB Serial Emulation

- All sorts of things appear as a serial port.
  - Uninterruptible Power Supplies
  - Modems, Phones, Radios
  - Facedancer!
Targets in Windows

- Unmaintained drivers are gold.
- Auto-installation approximates Linux variety.
  - Variety, but not speed.
- Windows 8 disables misbehaving USB ports.
Targets in Linux

- All drivers by default!
- No loading delays!
- Massive attack surface.
Targets in Mac

- Holy crap the stack’s performance is bad.
- Can’t emulate HID on localhost!
- Lack of driver variety can limit attack surface.
Targets in FreeBSD

- Complex drivers not included by default.
  - Wifi, etc.
- Pay attention to **usbpf**.
- See our paper from **WESS 2012**.
  - Instrumentation with dtrace.
Conclusions

- USB opens a massive attack surface to inputs.
- Network stack exploration methods also work for USB stacks – similar “routing” structure to be exploited.
- We’ve begun to build tools to exploit this structure.
- “Magical” abstractions lead to unrealistic validity assumptions ⇒ bugs, likely exploitable.
- Other buses: you are next!
  (If Daisho doesn’t beat us to it)
“Layers of abstraction become boundaries of competence”

← “Fast path”, cross-layer design

WTF 1.0, reference implementation →