USB for All!!1
You should be looking at USB.
Yes, you.
Introduction

• Who We Are
  • Jesse Michael
  • Mickey Shkatov
• What We Do
  • Break things
  • Cry about the current level of security research focused on USB

DISCLAIMER: The views and opinions expressed in this presentation are those of the authors and not their employer.
Purpose of this talk

YOU GON LEARN TODAY!
Purpose of this talk

- We want to demonstrate to attendees how easy it is to get started at performing their own USB security research and help them understand why they should undertake this challenge.
Why care about USB?

- We believe that even though USB is a pervasive technology in modern computing platforms, current security research has still only scratched its surface.
- USB has some interesting capabilities and is currently being used in a wide array of lesser-known usage models that can result in security problems.
USB Basics

- USB versions
  - 1.0, 1.1, 2.0, 3.0, 3.1

- Speeds
  - Low Speed, Full Speed, High Speed, Super Speed Gen1 and Gen2

- Device classes
  - HID, Mass Storage, Image, Video, Audio, Communications, Vendor Defined

- Physical connections
  - 1.x/2.0 standard
    - Vcc (5V), Data+, Data-, Ground
  - 1.x/2.0 mini/micro
    - Added USB OTG ID Pin
  - 3.0+
    - Added SSTx+, SSTx-, SSRx+, SSRx-
Getting started

Figure 3-4. Enhanced SuperSpeed Bus Communications Layers and Power Management Elements

http://www.usb.org/developers/docs/

At over 600 pages, the USB specification can be a little intimidating...
In most USB devices, the physical, link, and protocol layers are handled in hardware...
Getting started

These areas are the easiest place to get started and find vulnerabilities so we’ll focus here and on some bigger picture views of how USB devices are used in modern platforms.
Do I need to touch it?

Most people think about USB like this...
Do I need to touch it?

...but almost all modern laptops have internal USB devices.

Which often contain their own processors with firmware and are separate from the host CPU and operating system.

What could go wrong?
There’s firmware in my USB?

Even “simple” USB devices can have interesting complexity

As an example, here’s a sync cable for an older phone...
There’s firmware in my USB?

That contains a USB to UART bridge chip that looks like this internally...

- 8052 processor
- 10K Boot ROM
- 16K RAM
- 2K SRAM
- Loads firmware from I²C

![Figure 1-2. USB-to-Serial (Single Channel) Controller Block Diagram]
There’s firmware in my USB?

And the datasheet describes how to run your own code in it...

11.8 Built-In Vendor Specific USB Requests

The bootcode supports several vendor specific USB requests. These requests are primarily for internal testing only. These functions should not be used in normal operation.

11.8.1 Reboot

The reboot command forces the bootcode to execute.

<table>
<thead>
<tr>
<th>bmRequestType</th>
<th>USB_REQ_TYPE_DEVICE</th>
<th>USB_REQ_TYPE_VENDOR</th>
<th>USB_REQ_TYPE_OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bRequest</td>
<td>BTC_REBOOT</td>
<td>0x85</td>
<td></td>
</tr>
<tr>
<td>wValue</td>
<td>None</td>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td>wIndex</td>
<td>None</td>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td>wLength</td>
<td>None</td>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

11.8.2 Force Execute Firmware

The force execute firmware command requests the bootcode to execute the downloaded firmware unconditionally.

<table>
<thead>
<tr>
<th>bmRequestType</th>
<th>USB_REQ_TYPE_DEVICE</th>
<th>USB_REQ_TYPE_VENDOR</th>
<th>USB_REQ_TYPE_OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bRequest</td>
<td>BTC_FORCE_EXECUTE_FIRMWARE</td>
<td>0x83</td>
<td></td>
</tr>
<tr>
<td>wValue</td>
<td>None</td>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td>wIndex</td>
<td>None</td>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td>wLength</td>
<td>None</td>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Arbitrary code execution inside your phone sync cable? Really?
There's firmware in my USB?

And the datasheet describes how to run your own code in it...

11.8 Built-In Vendor Specific USB Requests

The bootcode supports several vendor specific USB requests. These requests are primarily for internal testing only. These functions should not be used in normal operation.

The reboot command forces the bootcode to execute.

<table>
<thead>
<tr>
<th>bmRequestType</th>
<th>USB_REQ_TYPE_DEVICE</th>
<th>USB_REQ_TYPE_VENDOR</th>
<th>USB_REQ_TYPE_OUT</th>
<th>01000000b</th>
</tr>
</thead>
<tbody>
<tr>
<td>bRequest</td>
<td>BTC_REBOOT</td>
<td>0x85</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wValue</td>
<td>None</td>
<td>0x0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wIndex</td>
<td>None</td>
<td>0x0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wLength</td>
<td>None</td>
<td>0x0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>None</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

11.8.2 Force Execute Firmware

The force execute firmware command requests the bootcode to execute the downloaded firmware unconditionally.

<table>
<thead>
<tr>
<th>bmRequestType</th>
<th>USB_REQ_TYPE_DEVICE</th>
<th>USB_REQ_TYPE_VENDOR</th>
<th>USB_REQ_TYPE_OUT</th>
<th>01000000b</th>
</tr>
</thead>
<tbody>
<tr>
<td>bRequest</td>
<td>BTC_FORCE_EXECUTE_FIRMWARE</td>
<td>0x8F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wValue</td>
<td>None</td>
<td>0x0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wIndex</td>
<td>None</td>
<td>0x0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wLength</td>
<td>None</td>
<td>0x0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>None</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Arbitrary code execution inside your phone sync cable? Really?
There's firmware in my USB?

The datasheet also describes how to read and write to the I²C EEPROM it executes code from...

### 11.8.5 I²C Memory Read

The bootloader returns the content of the specified address in I²C EEPROM.

In the wValue field, the I²C device number is from 0x00 to 0x07 in the high byte. The memory type is from 0x01 to 0x03 for CAT I to CAT II devices. If bit 7 of bValueL is set, then the bus speed is 400 kHz. This request is also used to set the device number and speed before the I²C write request.

<table>
<thead>
<tr>
<th>bmRequestType</th>
<th>USB_REQ_TYPE_DEVICE</th>
<th>USB_REQ_TYPE_VENDOR</th>
<th>USB_REQ_TYPE_IN</th>
</tr>
</thead>
<tbody>
<tr>
<td>bRequest</td>
<td>BTC_I2C_MEMORY_READ</td>
<td>0x92</td>
<td></td>
</tr>
<tr>
<td>wValue</td>
<td>Hi: I²C device number</td>
<td>LO: Memory type bit[1:0]</td>
<td>Speed bit[7]</td>
</tr>
<tr>
<td>wIndex</td>
<td>Data address</td>
<td>0xNNNN (From 0x0000 to 0xFFFF)</td>
<td></td>
</tr>
<tr>
<td>wLength</td>
<td>1 byte</td>
<td>0x0001</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>Byte in the specified address</td>
<td>0xNN</td>
<td></td>
</tr>
</tbody>
</table>

### 11.8.6 I²C Memory Write

The I²C memory write command tells the bootloader to write data to the specified address.

<table>
<thead>
<tr>
<th>bmRequestType</th>
<th>USB_REQ_TYPE_DEVICE</th>
<th>USB_REQ_TYPE_VENDOR</th>
<th>USB_REQ_TYPE_OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bRequest</td>
<td>BTC_I2C_MEMORY_WRITE</td>
<td>0x93</td>
<td></td>
</tr>
<tr>
<td>wValue</td>
<td>Hi: should be zero</td>
<td>LO: Data</td>
<td></td>
</tr>
<tr>
<td>wIndex</td>
<td>Data address</td>
<td>0xNNNN (From 0x0000 to 0xFFFF)</td>
<td></td>
</tr>
<tr>
<td>wLength</td>
<td>None</td>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Device Firmware Upgrade

http://www.usb.org/developers/devclass_docs/DFU_1.1.pdf

There’s actually a specification for how to create USB devices with upgradable firmware.

It doesn’t mention security at all. And most devices that implement this capability don’t bother to do any validation of the firmware image other than basic checksums which are easy to bypass.

DFU and similar custom device upgrade methods are a good way to easily get arbitrary code execution within a USB device.

What can we do with that?
Attack Surfaces

So instead of looking at it like this...

Host

USB cable

Device
Attack Surfaces

There’s actually a lot more going on...
Attack Surfaces

All of this is probably happening inside your laptop right now.
Attack Surfaces

Some USB devices even have radio interfaces...
Attack Surfaces

If you can get arbitrary code execution within the USB device...
Attack Surfaces

It can be used to attack components within the host.
Attack Surfaces

Even with attacks originating from the host, these can cross privilege boundaries.
Debug Capability

• Allows low-level debug over USB

• Now required to gain Windows Logo certification
  • “If the XHCI controller in the SUT has any user-accessible ports, the controller must have debug capability.”
Media Agnostic USB

Figure 1—Building blocks of an MA USB host and relationship to existing USB infrastructure

http://www.usb.org/developers/devclass_docs/Media_Agnostic_USB_v1.0.zip
Media Agnostic USB

**Figure 9—MA USB protocol hierarchy**
Tools!
Total Phase Beagle 5000

http://www.totalphase.com/protocols/usb/

Supports USB 3.0 SuperSpeed, but very expensive. Can only be used for observation and not injection.
Total Phase Beagle 480

http://www.totalphase.com/protocols/usb/

Less expensive than Beagle 5000, but only supports USB 2.0. Can only be used for observation and not injection.
ITIC 1480A USB 2.0 Protocol Analyzer


HW less expensive than Beagle 480, but some SW modules sold separately. Can only be used for observation and not injection.
Facedancer

http://goodfet.sourceforge.net/

Open source, cheap and easy to build, allows arbitrary emulation of USB endpoints, but can be very slow
Daisho

https://github.com/mossmann/daisho

Open source, intended to support full USB 3.0 SuperSpeed monitoring and injection, but still in development
USBProxy

https://github.com/dominicgs/USBProxy

Open source project to create a USB 2.0 MitM device using the BeagleBone Black, still in early stages, but can already do some cool stuff
libusb

- [http://www.libusb.org/](http://www.libusb.org/)

- Good way to get started with writing tools to access USB devices
Peach Publishers

- Data Publisher
- Configuration Publisher

- This uses libUsbDotNet which hasn’t been maintained in a while, so it has its bugs, but has been useful for finding issues.

- Available on the DEFCON CD.
- Submitted to Peach upstream
Phison PS2303 framework

- [https://bitbucket.org/flowswitch/phison/](https://bitbucket.org/flowswitch/phison/)

- Phison PS2303 is a USB 3.0 NAND controller used in many flash drives

- 8051-compatible core
- 256KiB ram
- USB, NAND, and DMA controllers
- Stores firmware in NAND

- Goal of the project is to use PS2303-based flash drives as a cheap USB3.0 development platform
Kautilya toolkit

- [https://github.com/samratashok/Kautilya](https://github.com/samratashok/Kautilya)

- Collection of 40+ USB HID payloads for penetration testing
  - Dump Process Memory
  - Dump Windows Vault Credentials
  - Download and Execute
  - Connect to Hotspot and Execute code
  - Code Execution using Powershell
  - Code Execution using DNS TXT queries
  - Add an admin user
  - Add a user and Enable RDP
  - Add a user and Enable Powershell Remoting
  - ... and more

- Designed to be used with Arduino devices like the Teensy, but easily adaptable to other devices like the Phison chipset
Demos!
Summary

- USB is a pervasive technology in modern computing devices.
- Not just external ports which require physical access to attack
- Devices connected over USB run upgradable firmware
- Debug capability required for Windows certification
- Interesting attack scenarios with internal devices

USB provides a rich set of capabilities and is being used in a variety of configurations that could result in security vulnerabilities and it’s easier than people think to get started looking at this stuff.
Questions?

jesse at lonelyrhinoceros.com
laplinker at gmail.com