Revealing Embedded Fingerprints:
Deriving intelligence from USB stack interactions

Andy Davis, Research Director NCC Group
UK Offices
Manchester - Head Office
Cheltenham
Edinburgh
Leatherhead
London
Thame

European Offices
Amsterdam - Netherlands
Munich - Germany
Zurich - Switzerland

North American Offices
San Francisco
Atlanta
New York
Seattle

Australian Offices
Sydney
Agenda

Part One:
• Overview of the USB enumeration phase
• Different USB stack implementations
• USB testing platform
• Installed drivers and supported devices
• Fingerprinting USB stacks and OS versions

Part Two:
• The Windows 8 RNDIS kernel pool overflow
• Challenges faced when exploiting USB bugs
• Conclusions
Part One: Information gathering

• Why do we care?

• If you connect to a device surely you already know the platform?

• Embedded devices are mostly based on Linux anyway aren't they?

• May provide information useful for other attacks
USB Background stuff

UNIVERSAL SERIAL BUS

Image from: blog.brickhousesecurity.com
Overview of the USB enumeration phase

• What is enumeration for?
  • Assign an address
  • Speed of communication
  • Power requirements
  • Configuration options
  • Device descriptions
  • Class drivers

• Lots of information exchange – implemented in many different ways
# The USB enumeration phase

<table>
<thead>
<tr>
<th>LS</th>
<th>Control Transfer</th>
<th>Addr</th>
<th>Endp</th>
<th>Data (18 bytes)</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Get Device Descriptor 0x00</td>
<td>0x0</td>
<td>0x0</td>
<td>12 01 10 01 00 00 00 00 08...</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>Set Address (0x01)</td>
<td>0x00</td>
<td>0x0</td>
<td></td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>Get Device Descriptor 0x01</td>
<td>0x0</td>
<td>0x0</td>
<td>12 01 10 01 00 00 00 00 08...</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>Get Configuration Descriptor 0x01</td>
<td>0x0</td>
<td>0x0</td>
<td>09 02 22 00 01 01 00 A0...</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>Get String Descriptor 0</td>
<td>0x01</td>
<td>0x0</td>
<td>04 03 09 04</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>Get String Descriptor 2</td>
<td>0x01</td>
<td>0x0</td>
<td>30 03 44 00 65 00 6C 00...</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>Get Device Descriptor 0x01</td>
<td>0x0</td>
<td>0x0</td>
<td>12 01 10 01 00 00 00 00 08...</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>Get Configuration Descriptor 0x01</td>
<td>0x0</td>
<td>0x0</td>
<td>09 02 22 00 01 01 00 A0...</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>Get Configuration Descriptor 0x01</td>
<td>0x0</td>
<td>0x0</td>
<td>09 02 22 00 01 01 00 A0...</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>Set Configuration (0x01)</td>
<td>0x01</td>
<td>0x0</td>
<td></td>
<td>OK</td>
</tr>
</tbody>
</table>
Enumeration phase peculiarities

• Why is the device descriptor initially requested twice?

• Why are there multiple requests for other descriptors?

• Class-specific descriptors:

```
Control Transfer  Addr  Endp  Data (64 bytes)     Status
Get HID Report Descriptor  0x01  0x0  05 01 09 06 A1 01 05 07
```

```
Control Transfer  Addr  Endp  Data (1 byte)     Status
Set Report (HID)   0x01  0x0  00              OK
```
Different USB stack implementations

• Typical components of a USB stack

• Windows USB driver stack

• Linux USB stack

• Embedded Access USB stack
Typical components of a USB stack

- Host Controller hardware

- USB System software:
  - Host Controller Driver – Hardware Abstraction Layer
  - USB Driver

- Class drivers

- Application software
Windows USB driver stack

Legend
- Client or Class Driver
- Microsoft-Provided Helper Libraries
- Microsoft-Provided Inbox drivers
- Hardware

User Mode
Kernel Mode

USB Client Driver Layer
Winusb.dll
Client Driver

USB Driver Stack Layer
Winusb.sys
Usbccgp.sys
Client Driver
Usbd.sys

Data Flow
URB, IOCTL

USB 3.0 Driver Stack
Usbhci.sys
Usbhub3.sys
Ucx01000.sys
Usbxchci.sys

USB 2.0 Driver Stack
Usbhci.sys
Usbhub.sys
Usbport.sys

UHCI Host Controller and Root Hub

Image from: msdn.microsoft.com
Linux USB stack

(user-space) → (Class drivers) → (vendor-specific)

mp3 → hub → HID → webcam

URBs → or func parms

usbdevfs/procfs → usbcors services

URB → usb_operations: URB or dev*

OHCI HCD → UHCI HCD

callbacks(urb)
Embedded Access USB stack
Interacting with USB
USB interaction requirements

• Need to capture and replay USB traffic

• Full control of generated traffic

• Class decoders extremely useful

• Support for Low/High/Full speed required

• USB 3.0 a bonus
USB testing – gold-plated solution

* Commercial test equipment
USB testing – the cheaper approach

• Facedancer (http://goodfet.sourceforge.net/hardware/facedancer21)
Best solution: A combination of both

- Device data can be carefully crafted
- Host response data can be captured
- Microsecond timing is also recorded
- All class-specific data is decoded
Information enumeration
Target list

• Windows 8
• Ubuntu Linux 12.04 LTS
• Apple OS X Lion
• FreeBSD 5.3
• Chrome OS
• Linux-based TV STB
Installed drivers and supported devices

• Enumerating supported class types – standard USB drivers

• Enumerating all installed drivers

• Other devices already connected
Enumerating supported class types

Where is USB class information stored?

### Device Descriptor

<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>0x0110</td>
<td>Spec Version</td>
</tr>
<tr>
<td>bDeviceClass</td>
<td>0x00</td>
<td>Class Information in Interface Descriptor</td>
</tr>
<tr>
<td>bDeviceSubClass</td>
<td>0x00</td>
<td>Class Information in Interface Descriptor</td>
</tr>
</tbody>
</table>

### Interface Descriptor

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bLength</td>
<td>9</td>
<td>Valid length</td>
</tr>
<tr>
<td>bDescriptorType</td>
<td>4</td>
<td>INTERFACE</td>
</tr>
<tr>
<td>bInterfaceNumber</td>
<td>0</td>
<td>Zero-based Number of this Interface.</td>
</tr>
<tr>
<td>bAlternateSetting</td>
<td>0</td>
<td>Value used to select this alternative setting for the interface identified in the prior field</td>
</tr>
<tr>
<td>bNumEndpoints</td>
<td>1</td>
<td>Number of endpoints used by this interface (excluding endpoint zero).</td>
</tr>
<tr>
<td>bInterfaceClass</td>
<td>0x03</td>
<td>HID</td>
</tr>
<tr>
<td>bInterfaceSubClass</td>
<td>0x01</td>
<td>Boot Interface</td>
</tr>
</tbody>
</table>
Installed drivers and supported devices

• Drivers are referenced by class (Device and Interface descriptors)
• Also, by VID and PID:

<table>
<thead>
<tr>
<th>idVendor</th>
<th>0x090C</th>
<th>Silicon Motion, Inc. - Taiwan</th>
</tr>
</thead>
<tbody>
<tr>
<td>idProduct</td>
<td>0x1000</td>
<td>Memory Bar</td>
</tr>
</tbody>
</table>

• For each device class VID and PID values can be brute-forced (can easily be scripted using Facedancer)
• Valid PIDs and VIDs are available (http://www.linux-usb.org/usb.ids)
Enumerating installed drivers

### Not installed:
- ![Not installed example](image)

### Installed:
- ![Installed example](image)

All communication stops after "Set Configuration"
Sniffing the bus - Other connected devices

- Data from other devices will be displayed on other addresses
Fingerprinting USB stacks and OS versions

• Descriptor request patterns
• Timing information
• Descriptor types requested
• Responses to invalid data
• Order of Descriptor requests
Matching req. patterns to known stacks

### Linux-based TV STB

<table>
<thead>
<tr>
<th>Control Transfer</th>
<th>Addr</th>
<th>Endp</th>
<th>Data (1 byte)</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Max LUN</td>
<td>0x21</td>
<td>0x02</td>
<td>00</td>
<td>Ok</td>
</tr>
<tr>
<td>CBW: INQUIRY</td>
<td>0x21</td>
<td>0x2</td>
<td>55 53 42 43 01 00 00 00...</td>
<td>Ok</td>
</tr>
<tr>
<td>Bulk Transfer</td>
<td>0x21</td>
<td>0x1</td>
<td>55 53 42 53 20 00 00 00...</td>
<td>Ok</td>
</tr>
<tr>
<td>MSC Data In</td>
<td>0x21</td>
<td>0x1</td>
<td>00 80 04 02 1F 73 6D 69...</td>
<td>Ok</td>
</tr>
<tr>
<td>CSW - Status:Passed</td>
<td>0x21</td>
<td>0x1</td>
<td>55 53 42 53 02 00 00 00...</td>
<td>Ok</td>
</tr>
<tr>
<td>CBW: TEST UNIT READY</td>
<td>0x21</td>
<td>0x2</td>
<td>55 53 42 43 02 00 00 00...</td>
<td>Ok</td>
</tr>
<tr>
<td>Bulk Transfer</td>
<td>0x21</td>
<td>0x1</td>
<td>55 53 42 53 02 00 00 00...</td>
<td>Ok</td>
</tr>
<tr>
<td>CSW - Status:Passed</td>
<td>0x21</td>
<td>0x1</td>
<td>55 53 42 53 03 00 00 00...</td>
<td>Ok</td>
</tr>
<tr>
<td>CBW: READ CAPACITY(10)</td>
<td>0x21</td>
<td>0x2</td>
<td>55 53 42 43 03 00 00 00...</td>
<td>Ok</td>
</tr>
<tr>
<td>Bulk Transfer</td>
<td>0x21</td>
<td>0x1</td>
<td>55 53 42 53 00 00 00 00...</td>
<td>Ok</td>
</tr>
<tr>
<td>CSW - Status:Passed</td>
<td>0x21</td>
<td>0x1</td>
<td>55 53 42 53 00 00 00 00...</td>
<td>Ok</td>
</tr>
<tr>
<td>CBW: MODE SENSE(6)</td>
<td>0x21</td>
<td>0x2</td>
<td>55 53 42 43 04 00 00 00...</td>
<td>Ok</td>
</tr>
</tbody>
</table>

### Windows 8

<table>
<thead>
<tr>
<th>Control Transfer</th>
<th>Addr</th>
<th>Endp</th>
<th>Data (1 byte)</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Max LUN</td>
<td>0x02</td>
<td>0x02</td>
<td>00</td>
<td>Ok</td>
</tr>
<tr>
<td>CBW: INQUIRY</td>
<td>0x02</td>
<td>0x2</td>
<td>55 53 42 43 28 1E A6 83...</td>
<td>Ok</td>
</tr>
<tr>
<td>Bulk Transfer</td>
<td>0x02</td>
<td>0x1</td>
<td>00 80 04 02 1F 73 6D 69...</td>
<td>Ok</td>
</tr>
<tr>
<td>MSC Data In</td>
<td>0x02</td>
<td>0x1</td>
<td>00 80 04 02 1F 73 6D 69...</td>
<td>Ok</td>
</tr>
<tr>
<td>CSW - Status:Passed</td>
<td>0x02</td>
<td>0x1</td>
<td>55 53 42 53 28 1E A6 83...</td>
<td>Ok</td>
</tr>
<tr>
<td>CBW: TEST UNIT READY</td>
<td>0x02</td>
<td>0x2</td>
<td>55 53 42 43 80 3C A8 83...</td>
<td>Ok</td>
</tr>
<tr>
<td>Bulk Transfer</td>
<td>0x02</td>
<td>0x1</td>
<td>55 53 42 53 28 1E A6 83...</td>
<td>Ok</td>
</tr>
<tr>
<td>CSW - Status:Passed</td>
<td>0x02</td>
<td>0x1</td>
<td>55 53 42 53 80 3C A8 83...</td>
<td>Ok</td>
</tr>
<tr>
<td>CBW: READ FORMAT CAPACITIES (or VS)</td>
<td>0x02</td>
<td>0x2</td>
<td>55 53 42 43 28 8E B2 83...</td>
<td>Ok</td>
</tr>
<tr>
<td>Bulk Transfer</td>
<td>0x02</td>
<td>0x1</td>
<td>55 53 42 53 80 3C A8 83...</td>
<td>Ok</td>
</tr>
<tr>
<td>CSW - Status:Passed</td>
<td>0x02</td>
<td>0x1</td>
<td>55 53 42 53 80 3C A8 83...</td>
<td>Ok</td>
</tr>
</tbody>
</table>
Request patterns unique elements?

- Windows 8 (HID) – Three Get Configuration descriptor requests (others have two)
- Apple OS X Lion (HID) – Set Feature request right after Set Configuration
- FreeBSD 5.3 (HID) – Get Status request right before Set Configuration
- Linux-based TV STB (Mass Storage) – Order of class-specific requests
### Timing information

<table>
<thead>
<tr>
<th>Capture13</th>
<th>Capture14</th>
<th>Capture15</th>
<th>Capture16</th>
<th>Capture17*</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>#4.18</strong></td>
<td><strong>#4.18</strong></td>
<td><strong>#4.18</strong></td>
<td><strong>#4.18</strong></td>
<td><strong>#4.18</strong></td>
</tr>
<tr>
<td>3.035,975 s</td>
<td>3.150,387 s</td>
<td>2.846,082 s</td>
<td>3.858,666 s</td>
<td>3.235,349 s</td>
</tr>
<tr>
<td><strong>#20.25</strong></td>
<td><strong>#20.25</strong></td>
<td><strong>#20.25</strong></td>
<td><strong>#20.25</strong></td>
<td><strong>#20.25</strong></td>
</tr>
<tr>
<td>3.085,990 s</td>
<td>3.211,403 s</td>
<td>2.893,073 s</td>
<td>3.908,665 s</td>
<td>3.282,360 s</td>
</tr>
<tr>
<td>3.104,993 s</td>
<td>3.238,408 s</td>
<td>2.924,081 s</td>
<td>3.939,672 s</td>
<td>3.301,365 s</td>
</tr>
<tr>
<td><strong>#41.61</strong></td>
<td><strong>#41.61</strong></td>
<td><strong>#41.61</strong></td>
<td><strong>#41.61</strong></td>
<td><strong>#41.61</strong></td>
</tr>
<tr>
<td>3.110,995 s</td>
<td>3.238,408 s</td>
<td>2.930,084 s</td>
<td>3.945,674 s</td>
<td>3.307,366 s</td>
</tr>
<tr>
<td><strong>#62.70</strong></td>
<td><strong>#62.70</strong></td>
<td><strong>#62.70</strong></td>
<td><strong>#62.70</strong></td>
<td><strong>#62.70</strong></td>
</tr>
<tr>
<td>3.122,997 s</td>
<td>3.249,410 s</td>
<td>2.941,083 s</td>
<td>3.957,678 s</td>
<td>3.319,369 s</td>
</tr>
<tr>
<td><strong>#71.97</strong></td>
<td><strong>#71.97</strong></td>
<td><strong>#71.97</strong></td>
<td><strong>#71.97</strong></td>
<td><strong>#71.97</strong></td>
</tr>
<tr>
<td>3.127,999 s</td>
<td>3.254,413 s</td>
<td>2.946,086 s</td>
<td>3.962,677 s</td>
<td>3.324,370 s</td>
</tr>
<tr>
<td><strong>#98.112</strong></td>
<td><strong>#98.112</strong></td>
<td><strong>#98.112</strong></td>
<td><strong>#98.112</strong></td>
<td><strong>#98.112</strong></td>
</tr>
<tr>
<td>3.537,094 s</td>
<td>3.333,430 s</td>
<td>3.026,103 s</td>
<td>4.006,685 s</td>
<td>3.761,470 s</td>
</tr>
<tr>
<td><strong>#113.124</strong></td>
<td><strong>#113.124</strong></td>
<td><strong>#113.124</strong></td>
<td><strong>#113.124</strong></td>
<td><strong>#113.124</strong></td>
</tr>
<tr>
<td>3.543,095 s</td>
<td>3.333,430 s</td>
<td>3.032,104 s</td>
<td>4.006,685 s</td>
<td>3.761,470 s</td>
</tr>
<tr>
<td><strong>#125.145</strong></td>
<td><strong>#125.145</strong></td>
<td><strong>#125.145</strong></td>
<td><strong>#125.145</strong></td>
<td><strong>#125.145</strong></td>
</tr>
<tr>
<td>3.548,096 s</td>
<td>3.344,435 s</td>
<td>3.037,106 s</td>
<td>4.011,685 s</td>
<td>3.766,475 s</td>
</tr>
<tr>
<td><strong>#146.151</strong></td>
<td><strong>#146.151</strong></td>
<td><strong>#146.151</strong></td>
<td><strong>#146.151</strong></td>
<td><strong>#146.151</strong></td>
</tr>
<tr>
<td>3.556,098 s</td>
<td>3.352,435 s</td>
<td>3.046,107 s</td>
<td>4.019,689 s</td>
<td>3.774,475 s</td>
</tr>
<tr>
<td><strong>#152.157</strong></td>
<td><strong>#152.157</strong></td>
<td><strong>#152.157</strong></td>
<td><strong>#152.157</strong></td>
<td><strong>#152.157</strong></td>
</tr>
<tr>
<td>3.559,099 s</td>
<td>3.355,437 s</td>
<td>3.048,109 s</td>
<td>4.022,690 s</td>
<td>3.777,476 s</td>
</tr>
<tr>
<td><strong>#158.190</strong></td>
<td><strong>#158.190</strong></td>
<td><strong>#158.190</strong></td>
<td><strong>#158.190</strong></td>
<td><strong>#158.190</strong></td>
</tr>
<tr>
<td>3.562,101 s</td>
<td>3.358,436 s</td>
<td>3.051,109 s</td>
<td>4.025,691 s</td>
<td>3.780,477 s</td>
</tr>
<tr>
<td><strong>#191.199</strong></td>
<td><strong>#191.199</strong></td>
<td><strong>#191.199</strong></td>
<td><strong>#191.199</strong></td>
<td><strong>#191.199</strong></td>
</tr>
<tr>
<td>3.612,112 s</td>
<td>3.403,447 s</td>
<td>3.089,118 s</td>
<td>4.055,698 s</td>
<td>3.834,489 s</td>
</tr>
</tbody>
</table>

---

--- End of Capture ---
### Timing information

<table>
<thead>
<tr>
<th>Capture13</th>
<th>Capture14</th>
<th>Capture15</th>
<th>Capture16</th>
<th>Capture17*</th>
</tr>
</thead>
<tbody>
<tr>
<td>#4..18</td>
<td>#4..18</td>
<td>#4..18</td>
<td>#4..18</td>
<td>#4..18</td>
</tr>
<tr>
<td>3.035.975s</td>
<td>3.150.387s</td>
<td>2.946.062s</td>
<td>3.856.666s</td>
<td>3.235.349s</td>
</tr>
<tr>
<td>#20..25</td>
<td>#20..25</td>
<td>#20..25</td>
<td>#20..25</td>
<td>#20..25</td>
</tr>
<tr>
<td>3.085.990s</td>
<td>3.211.403s</td>
<td>3.293.073s</td>
<td>3.908.665s</td>
<td>3.282.360s</td>
</tr>
<tr>
<td>#26..40</td>
<td>#26..40</td>
<td>#26..40</td>
<td>#26..40</td>
<td>#26..40</td>
</tr>
<tr>
<td>3.104.993s</td>
<td>3.232.408s</td>
<td>2.924.081s</td>
<td>3.939.672s</td>
<td>3.301.365s</td>
</tr>
<tr>
<td>#41..61</td>
<td>#41..61</td>
<td>#41..61</td>
<td>#41..61</td>
<td>#41..61</td>
</tr>
<tr>
<td>3.110.995s</td>
<td>3.238.409s</td>
<td>2.930.084s</td>
<td>3.946.674s</td>
<td>3.307.366s</td>
</tr>
<tr>
<td>#62..70</td>
<td>#62..70</td>
<td>#62..70</td>
<td>#62..70</td>
<td>#62..70</td>
</tr>
<tr>
<td>3.122.997s</td>
<td>3.249.410s</td>
<td>2.941.083s</td>
<td>3.957.678s</td>
<td>3.319.369s</td>
</tr>
<tr>
<td>#71..97</td>
<td>#71..97</td>
<td>#71..97</td>
<td>#71..97</td>
<td>#71..97</td>
</tr>
<tr>
<td>3.127.999s</td>
<td>3.254.413s</td>
<td>2.946.086s</td>
<td>3.962.677s</td>
<td>3.324.370s</td>
</tr>
<tr>
<td>#98..112</td>
<td>#98..112</td>
<td>#98..112</td>
<td>#98..112</td>
<td>#98..112</td>
</tr>
<tr>
<td>3.637.094s</td>
<td>3.333.430s</td>
<td>3.026.103s</td>
<td>4.000.685s</td>
<td>3.765.470s</td>
</tr>
<tr>
<td>#113..124</td>
<td>#113..124</td>
<td>#113..124</td>
<td>#113..124</td>
<td>#113..124</td>
</tr>
<tr>
<td>3.543.095s</td>
<td>3.339.431s</td>
<td>3.032.104s</td>
<td>4.006.685s</td>
<td>3.767.472s</td>
</tr>
<tr>
<td>#125..145</td>
<td>#125..145</td>
<td>#125..145</td>
<td>#125..145</td>
<td>#125..145</td>
</tr>
<tr>
<td>3.548.096s</td>
<td>3.344.430s</td>
<td>3.037.108s</td>
<td>4.011.687s</td>
<td>3.786.473s</td>
</tr>
<tr>
<td>#146..151</td>
<td>#146..151</td>
<td>#146..151</td>
<td>#146..151</td>
<td>#146..151</td>
</tr>
<tr>
<td>3.556.098s</td>
<td>3.354.430s</td>
<td>3.038.116s</td>
<td>4.056.698s</td>
<td>3.834.489s</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>LS</th>
<th>Control Transfer</th>
<th>Addr</th>
<th>Endp</th>
<th>Data (18 bytes)</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>LS</td>
<td>Get Device Descriptors 0x00</td>
<td>0x00</td>
<td>12</td>
<td>01 01 00 00 00 00 00 08</td>
<td>OK</td>
</tr>
<tr>
<td>LS</td>
<td>Get Address (0x01)</td>
<td>0x00</td>
<td>0x0</td>
<td>00</td>
<td>OK</td>
</tr>
<tr>
<td>LS</td>
<td>Get Device Descriptors 0x01</td>
<td>0x00</td>
<td>12</td>
<td>01 01 01 00 00 00 00 00 00 08</td>
<td>OK</td>
</tr>
<tr>
<td>LS</td>
<td>Get Device Descriptors 0x02</td>
<td>0x00</td>
<td>12</td>
<td>01 01 01 00 00 00 00 00 00 00 00 00 00</td>
<td>OK</td>
</tr>
<tr>
<td>LS</td>
<td>Get Configuration Descriptors 0x01</td>
<td>0x00</td>
<td>0x0</td>
<td>09 02 22 00 01 00 00 0A</td>
<td>OK</td>
</tr>
<tr>
<td>LS</td>
<td>Get String Descriptor 0x01</td>
<td>0x00</td>
<td>0x0</td>
<td>04 03 09 04</td>
<td>OK</td>
</tr>
<tr>
<td>LS</td>
<td>Get Device Descriptors 0x03</td>
<td>0x00</td>
<td>12</td>
<td>01 01 01 00 00 00 00 00 00 00 00 00 00</td>
<td>OK</td>
</tr>
<tr>
<td>LS</td>
<td>Get Configuration Descriptors 0x01</td>
<td>0x00</td>
<td>0x0</td>
<td>09 02 22 00 01 00 00 0A</td>
<td>OK</td>
</tr>
<tr>
<td>LS</td>
<td>Get Configuration Descriptors 0x02</td>
<td>0x00</td>
<td>0x0</td>
<td>09 02 22 00 01 00 00 0A</td>
<td>OK</td>
</tr>
<tr>
<td>LS</td>
<td>Get Configuration Descriptors 0x03</td>
<td>0x00</td>
<td>0x0</td>
<td>09 02 22 00 01 00 00 0A</td>
<td>OK</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>LS</th>
<th>Control Transfer</th>
<th>Addr</th>
<th>Endp</th>
<th>Data (6 bytes)</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>LS</td>
<td>Get HID Report Descriptors 0x00</td>
<td>0x00</td>
<td>0x0</td>
<td>01 05 01 00 01 05 07</td>
<td>OK</td>
</tr>
<tr>
<td>LS</td>
<td>Get Report (HID)</td>
<td>0x00</td>
<td>0x0</td>
<td>00</td>
<td>OK</td>
</tr>
</tbody>
</table>

---

Note: The "Capture17*" section appears to have data that is marked as "OK" for all entries, suggesting successful completion or validation of the captured data.
Using timing for fingerprinting?

• Large amount of variance over entire enumeration phase:
  • 4.055s, 3.834s, 3.612s, 3.403s, 3.089s

• Much greater accuracy between specific requests:
  • Between String Descriptor #0 and #2 requests - 5002us, 5003us, 5003us, 4999us, 5001us

• If we know the OS we can potentially determine the processor speed
Descriptor types requested

• Microsoft OS Descriptors (MOD)
• Used for “unusual” devices classes
• Devices that support Microsoft OS Descriptors must store a special USB string descriptor in firmware at the fixed string index of 0xEE. The request is:

<table>
<thead>
<tr>
<th>bmRequestType</th>
<th>bRequest</th>
<th>wValue</th>
<th>wIndex</th>
<th>wLength</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000 0000B</td>
<td>GET_DESCRIPTOR</td>
<td>0x03EE</td>
<td>0x0000</td>
<td>0x12</td>
<td>Returned String</td>
</tr>
</tbody>
</table>

• If a device does not contain a valid string descriptor at index 0xEE, it must respond with a stall packet. If the device does not respond with a stall packet, the system will issue a single-ended zero reset packet to the device, to help it recover from its stalled state (Windows XP only).
Responses to invalid data

- Different USB stacks respond to invalid data in different ways
- Maximum and minimum values
- Logically incorrect values
- Missing data
Invalid data unique elements?

Windows 8 (all versions)

If you send a specific, logically incorrect HID Report descriptor this happens:
Invalid data unique elements?

Windows 8 (all versions)

If you send a specific, logically incorrect HID Report descriptor this happens:
Order of Descriptor requests

• Some USB stacks request data from devices in a different order

• Different drivers may request different descriptors multiple times

• Sometimes Device descriptors are re-requested after enumeration is complete
Part Two: Potentially exploitable USB bugs
The Windows 8 RNDIS kernel pool overflow

- MS13-027
- usb8023x.sys - default (Microsoft-signed) Windows Remote NDIS driver that provides network connectivity over USB.
- When a USB device that uses this driver is inserted into a Windows host, during the enumeration phase the USB Configuration descriptor is requested and parsed.
- When the following USB descriptor field is manipulated a Bug check occurs indicating a kernel pool overwrite:
  - Configuration descriptor --> bNumInterfaces field > actual number of USB interfaces

The field is “bNumInterfaces” in Table A2: USB Configuration Descriptor (http://msdn.microsoft.com/en-us/windows/hardware/gg463298)
The Bug Check

BAD_POOL_HEADER (19)
The pool is already corrupt at the time of the current request.

<Truncated for brevity>

Arguments:

Arg1: 00000020, a pool block header size is corrupt.
Arg2: 83e38610, The pool entry we were looking for within the page.
Arg3: 83e38690, The next pool entry.
Arg4: 08100008, (reserved)

<Truncated for brevity>

WARNING: SystemResourcesList->Flink chain invalid. Resource may be corrupted, or already deleted.

WARNING: SystemResourcesList->Blink chain invalid. Resource may be corrupted, or already deleted.

SYMBOL_NAME: usb8023x!SelectConfiguration+1bd
The SelectConfiguration() function

```assembly
SelectConfiguration(x)
SelectConfiguration(x)+2
SelectConfiguration(x)+3
SelectConfiguration(x)+5
SelectConfiguration(x)+8
SelectConfiguration(x)+9
SelectConfiguration(x)+A
SelectConfiguration(x)+D
SelectConfiguration(x)+E
SelectConfiguration(x)+11
SelectConfiguration(x)+14
SelectConfiguration(x)+16
SelectConfiguration(x)+1C
SelectConfiguration(x)+1F
SelectConfiguration(x)+26
SelectConfiguration(x)+27
SelectConfiguration(x)+2C
SelectConfiguration(x)+2F
SelectConfiguration(x)+31
SelectConfiguration(x)+37
SelectConfiguration(x)+39
SelectConfiguration(x)+3C
SelectConfiguration(x)+3E
mov    edi, edi
push   ebp
mov    ebp, esp
sub    esp, 10h
push   ebx
push   esi
mov    esi, [ebp+ptr_Pool_U802]
push   edi
mov    edi, [esi+1Ch]   ; points to start of configuration descriptor
mov    al, [edi+4]     ; al = bNumInterfaces
cmp    al, 2          ; compares with 2 (what it should be)
jb     loc_11877      ; no jump
movzx  eax, al
lea    eax, ds:8[eax*8]   ; multiply bNumInterfaces by 8 then add 8 = 24
push   eax
call   _AllocPool@4    ; AllocPool(x)
mov    [ebp+ptr_Pool_U802_24_bytes], eax
test   eax, eax
jz     loc_11877       ; no jump (AllocPool was successful)
xor    ebx, ebx
cmp    [edi+4], bl    ; compares bNumInterfaces with 0
jbe    short loc_1171F ; no jump
mov    esi, eax
```
The crash point

```c
SelectConfiguration(x)+9B
SelectConfiguration(x)+9D
SelectConfiguration(x)+9F
SelectConfiguration(x)+A1
SelectConfiguration(x)+A3
SelectConfiguration(x)+A4
SelectConfiguration(x)+A5
SelectConfiguration(x)+A6
SelectConfiguration(x)+AC
SelectConfiguration(x)+B0
SelectConfiguration(x)+B3
SelectConfiguration(x)+B6
SelectConfiguration(x)+B8
SelectConfiguration(x)+B8
SelectConfiguration(x)+BC
SelectConfiguration(x)+BF
SelectConfiguration(x)+C2
SelectConfiguration(x)+C4
SelectConfiguration(x)+C8
SelectConfiguration(x)+C9
SelectConfiguration(x)+CC
SelectConfiguration(x)+CE

yet_more_interfaces_to_parse:

    push 0FFFFFFFh
    push 0FFFFFFFh
    push 0
    push ecx
    push edi
    call ds:imp_USB_ParseConfigurationDescriptorEx028
    test eax, eax
    jz short loc_11770
    mov al, [eax+5]
    mov [esi+4], al
    jmp short loc_11774

loc_11770:

    mov byte ptr [esi+4], 0 ; writes one null byte over the first byte of the next pool header
    ; this is where the corruption occurs

loc_11774:

    movzx eax, word ptr [esi]
    mov ecx, [ebp+ptr_Pool_U802_24_bytes]
    add esi, eax
    movzx eax, byte ptr [edi+4]
    inc ecx
    mov [ebp+ptr_Pool_U802_24_bytes], ecx
    cmp ecx, eax
    jb short yet_more_interfaces_to_parse
```
Analysis #1

When bNumInterfaces = 3 (one more than it should be) and bNumEndpoints = 2 (valid value)

Next kernel pool:

849c3b28 10 00 0a 04 56 61 64 6c-6b 8f 94 85 28 8c 90 85 ....Vadlk...(...

becomes:

849c3b28 00 00 0a 04 56 61 64 6c-6b 8f 94 85 28 8c 90 85 ....Vadlk...(...

So we’re overwriting "PreviousSize" in the next nt!_POOL_HEADER - this is what triggered the original Bug Check when ExFreePool() is called
Analysis #2

When bNumInterfaces = 3 (one more than it should be) and bNumEndpoints = 5 (three more than it should be)

Next kernel pool:

84064740 17 00 03 00 46 72 65 65-48 2d 09 84 30 a8 17 84 .....FreeH-...0...

becomes:

84064740 17 00 03 00 00 72 65 65-48 2d 09 84 30 a8 17 84 .....reeH-...0...

So we’re now overwriting "PoolTag" in the next nt!_POOL_HEADER
What’s going on?

kd> dt nt!_POOL_HEADER
- +0x000 PreviousSize : Pos 0, 8 Bits
- +0x000 PoolIndex : Pos 8, 8 Bits
- +0x000 BlockSize : Pos 16, 8 Bits
- +0x000 PoolType : Pos 24, 8 Bits
- +0x004 PoolTag : Uint4B
- +0x008 ProcessBilled : Ptr64 _EPROCESS

By manipulating bNumInterfaces and bNumEndpoints in a USB Configuration descriptor we appear to have a degree of control over where in the next adjacent kernel memory pool I can overwrite a single byte with a null (the null write occurs four bytes after the end of the pool I control and I can also control its size and some elements of its contents so could also potentially overwrite the next pool header with something useful)
for (i=0; i<something->count; i++)
{
    list[i].descriptor = USBD_ParseConfigurationDescriptorEx (...);
    ...
    if (!list[i].descriptor)
        break;
}

list[i].descriptor = NULL;

newthing = USB_CreateConfigurationRequestEx(thing, list);

if (newthing)
{
    ptr = &newthing->somemember;
    for (i=0; i<something->count; i++)
    {
        descriptor = USBD_ParseConfigurationDescriptorEx (...);
        ...
        if (descriptor)
        {
            ptr->someothermember = descriptor->whatever;
        }
        else
        {
            ptr->someothermember = 0;  // this is where I believe the corruption happens
        }
        ptr = ptr + ptr->Length;
    }
}
Challenges faced when exploiting USB bugs

- Lack of feedback channel
- The bug is often in kernel code
- Descriptors are generally very size-constrained
- Typical impact of USB exploitation typically restricted to privilege escalation
- What about USB over RDP?
Conclusions

• The USB enumeration phase reveals useful information for fingerprinting

• Class-specific communication is potentially even more revealing

• Even vendors with mature SDL processes have USB bugs

• USB bugs can potentially be exploited, to provide privilege escalation

• …but it is extremely difficult to achieve reliably
Questions?

Andy Davis, Research Director NCC Group
andy.davis ‘at’ nccgroup ‘dot’ com