Virtualised USB Fuzzing
Breaking USB for Fun and Profit

Tobias Mueller
School of Computing
Dublin City University

4. September 2010
1 USB
   - Trivia
   - Architecture
   - Fuzzing
   - Scapy Model

2 The Fuzzing
   - QEMU
   - Virtual USB Device
   - Obtaining Valid USB communication

3 Results
   - Stack Stress Test
   - USB Fingerprinting
   - Driver Flaws
About me

Tobi(as) Mueller

<table>
<thead>
<tr>
<th>Mail</th>
<th><a href="mailto:muellet2@computing.dcu.ie">muellet2@computing.dcu.ie</a></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>974C F452 FDA0 99D8 CB7E</td>
</tr>
<tr>
<td></td>
<td>54F7 DC03 BAA3 D349 2A2A</td>
</tr>
</tbody>
</table>

🔗 Talk ~ 30 mins
🔗 Ask immediately
🔗 Demo afterwards
Motivation

What's the problem?

- USB drivers in kernel space
- USB supported by every major OS
- USB widely deployed
- Not easy to assess security
  - Development board?
  - Inject messages into kernel?
Digital Voting Pen
Yes, it uses USB. hehe
In-Flight entertainment
Based on Linux or VxWorks
<1990: IO and IRQs

>1990: USB *yay*
cheap to build
hotplug
auto config

<5 metres
device → device
Architecture

- USB/IP
- Wireless USB

- Host initiated communication
- packet-based
  - SETUP
  - IN
  - OUT
- interrupt, bulk, isochronous, control
- descriptors → Driver
Abbildung: Key components of a system with USB support
Device Descriptor

QemuUSB
pipe_direction
pid
devaddr
devep
length

USBIn
Descriptor
length
type

DeviceDescriptor
bcdUSB
bDeviceClass
bDeviceSubClass
bDeviceProtocol
bMaxPacketSize
idVendor
idProduct
bcdDevice
iManufacturer
iProduct
iSerialNumber
bNumConfigurations
Configuration Descriptor

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>QemuUSB</td>
<td></td>
</tr>
<tr>
<td>pipe_direction</td>
<td></td>
</tr>
<tr>
<td>pid</td>
<td></td>
</tr>
<tr>
<td>devaddr</td>
<td></td>
</tr>
<tr>
<td>devep</td>
<td></td>
</tr>
<tr>
<td>length</td>
<td></td>
</tr>
<tr>
<td>Configuration</td>
<td></td>
</tr>
<tr>
<td>wTotalLength</td>
<td>39</td>
</tr>
<tr>
<td>bNumInterfaces</td>
<td>1</td>
</tr>
<tr>
<td>bConfigurationValue</td>
<td>1</td>
</tr>
<tr>
<td>iConfiguration</td>
<td>0</td>
</tr>
<tr>
<td>Reserved1</td>
<td>Correct</td>
</tr>
<tr>
<td>selfpowered</td>
<td>False</td>
</tr>
<tr>
<td>remotewakeup</td>
<td>False</td>
</tr>
<tr>
<td>Reserved0</td>
<td>Correct</td>
</tr>
<tr>
<td>bMaxPower</td>
<td>40</td>
</tr>
</tbody>
</table>
| descriptors      | [œ[0m<œ[0mœ[31mœ[1[...] ]

pipe direction 'D>H' (device to h[...]
69 00 00 00
pid IN
00
devaddr 0
00
devep 0
27 00 00 00
length 39
USBIn Descriptor
27 00 01 01 00 00 00 03 ff 06 ff 00
07 05 01 02 00 02 01 07 05 82 02 00 02 01 07 05 83 03 40 00 08
descriptors [œ[0m <œ[0mœ[31mœ[1[...]}
Endpoint Descriptor

**Descriptor**
- length: 7
- type: Endpoint

**InterfaceDescriptor**
- endpoint_direction: Out
- endpoint_reserved0: Correct
- endpoint_number: 1L
- reserved0: 0L
- usage_mode: Data
- sync_type: None
- transfer_type: Bulk
- wMaxPacketSize: 512
- bInterval: 1
Fuzzing

- Dumb Fuzzing
  - coined in late 80’s
  - feed program with random(?) data
  - got a lot attention in 2004

- Smart Fuzzing
  - Patent encumbered?
  - Modify existing valid structured data
  - Checksums
  - Cover more code

- Scapy
  - Awesome (!) framework
  - sniff, manipulate, craft, send (Ethernet) packets
  - models packets in Python
USB Stack stress testing
How many devices can you handle?

```python
class USBInDeviceDescriptor(Packet):
    name = 'DeviceDescriptor'

    fields_desc = [
        LEShortField('bcdUSB', 0x0200),
        ByteEnumField('bDeviceClass', 0, CLASS_ENUMS),
        ByteEnumField('bDeviceSubClass', 0, SUBCLASS_ENUMS),
        ByteEnumField('bDeviceProtocol', 0, PROTOCOL_ENUMS),
        ByteField('bMaxPacketSize', 64),
        LEXShortEnumField('idVendor', 0xffffff, VENDOR_ENUMS),
        LEXShortField('idProduct', 0x1337),
        LEShortField('bcdDevice', 0x2342),
        ByteField('iManufacturer', 0),
        ByteField('iProduct', 0),
        ByteField('iSerialNumber', 0),
        ByteField('bNumConfigurations', 0),
    ]
```
What the Fuzz?

Goal

Framework to **automatically** fuzz-test OS’s

- 🐾 Virtualised Guest
- 🐆 Attach virtual external USB devices
- 🐍 Generate USB packets in software
- 🐿 Monitor guest OS
**QEMU**

- Full virtualisation (not Xen, OpenVZ, UML, etc...)  
- Free (as in speech) Virtualisation (not VMWare)  
- Existing Virtual USB Drivers (not VirtualBox)  
- Supports management via QMP  
- had to patch in USB and other stuff
Virtual USB Device

- Take simple existing MSD or Serial driver
- Write out / Read in USB packets
- Implement desired behaviour externally
- cat and echo
- Or enhancing Scapy to read/write from pipes
- → Automaton class
Obtaining Valid USB communication

- Read specs :-(

  mount none -t debugfs /sys/kernel/debug
  mount none -t usbmon
  see Documentation/usb/usbmon.txt
  :-(

- Using QEMU: Implement filter to pipe out communication
def run_simple_test(qemu, timeout=4, delete=False):
    qemu.usb_add('mouse')
    time.sleep(timeout)
    cmd = list('dmesg') + ['space'] \
            + ['minus'] + ['c'] + ['enter']
    qemu.sendkeys(cmd)
    usb_devices = qemu.usb_info()
    if delete:
        for device in usb_devices['usbdevices']:
            qemu.usb_del('%d.%d' %
                         (device['busnr'], device['devaddr'])))

print qemu.cpu_info()
USB Fingerprinting

Targeted attacks

<table>
<thead>
<tr>
<th>OS</th>
<th>Packet Sequence</th>
<th>Retries</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>SET, IN, OUT</td>
<td>3</td>
<td>IN length: 64</td>
</tr>
<tr>
<td>Linux 2.6.33</td>
<td>SET (9x), RESET</td>
<td>4+2</td>
<td>4 get descriptor then 2 s</td>
</tr>
<tr>
<td>OpenBSD 4.7</td>
<td>SET, IN, OUT</td>
<td>7</td>
<td>IN length: 8</td>
</tr>
<tr>
<td>FreeBSD 8.0</td>
<td>SET, IN, OUT</td>
<td>6</td>
<td>tries to set address right</td>
</tr>
</tbody>
</table>

*Tabelle:* USB Stack Fingerprints of various operating systems
Demo: Who’s got Linux?
Future Work
What’s next?

- USB-3? (SuperSpeed, Device Initiated Communication)
- Making it work with GadgetFS
- Make that work on N900
- Get more fingerprints
- Exploit more drivers
- Run shellcode
Thanks!

Questions?!