Virtualised USB Fuzzing using QEMU and Scapy
Breaking USB for Fun and Profit

Tobias Mueller
(c) 2015, CC-BY-SA 3.0

2015-10-01
1 **Intro**
   - Motivation
   - USB Architecture

2 **Fuzzing**
   - Obtaining valid USB communication
   - QEMU
   - Virtual USB Device

3 **Results**
   - Stack Stress Test
   - USB Fingerprinting
   - Driver Flaws
Motivation

What’s the problem?

- USB supported by every major OS
- USB widely deployed
- USB drivers in kernel space
- 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
Architecture

- Host initiated communication
  - → polling
  - Yes, even with keyboards or mice
- packet-based
  - SETUP
  - IN
  - OUT
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

'D>H' (device to h[...]
69 00 00 00
pid IN
00
devaddr 0
00
devep 0
12 00 00 00

00 02 00 00 00 40 07 13 63 01 00 01 01 02 03 01
Known USB issues

The Playstation 3 Hack

Configuration Descriptor overflow...
Known USB issues (cont.)

**Solaris FAIL**
Configuration Descriptor overflow by Andy Davies (CVE-2011-2295)

**BadUSB**
Put several classes onto one device
Physical Access?

- Often argued that it’s not in the OS’s threat model
- except, it is…

- Not necessarily needed due to:
  - USB/IP
  - Wireless USB
Fuzzing

- **Dumb Fuzzing**
  - coined in late 80's
  - feed program with random(?) data
  - received a lot of attention ~ 2004

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

- **Scapy**
  - Awesome (!) framework
  - sniff, manipulate, craft, send (Ethernet) packets
  - models packets in Python
Obtaining Valid USB communication

.readString :-(

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

Using QEMU: Implement filter to pipe out communication (originally done by Moritz Jodeit)
Obtaining valid USB communication

QEMU
Virtual USB Device
QEMU

- Full virtualisation (not Xen, OpenVZ, UML, etc...)
- Free (as in speech) Virtualisation (not VMware)
- Existing Virtual USB Drivers
- (Unusable) Existing infrastructure for USB indirection
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
QEMU
Virtual USB Device

Host

Virtual Machine (QEMU)  Virtual USB Software Device
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
Targetted attacks

<table>
<thead>
<tr>
<th>OS</th>
<th>Packet Sequence</th>
<th>Retries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>SETUP, IN, OUT</td>
<td>3</td>
</tr>
<tr>
<td>Linux 2.6.33</td>
<td>SETUP (9x), RESET</td>
<td>4+2</td>
</tr>
<tr>
<td>OpenBSD 4.7</td>
<td>SETUP, IN, OUT</td>
<td>7</td>
</tr>
<tr>
<td>FreeBSD 8.0</td>
<td>SETUP, IN, OUT</td>
<td>6</td>
</tr>
</tbody>
</table>

_Tabelle:_ USB Stack Fingerprints of various operating systems
Future Work

What’s next?

- USB-3? (SuperSpeed, Device Initiated Communication)
- Making it work with GadgetFS
- Make that work on phones
- Get more OS fingerprints
- Exploit more drivers
- Run shellcode
- USB Firewall
Muchas Gracias!

Tobi(as) Mueller

Mail 4tmuelle@informatik.uni-hamburg.de
FF52 DA33 C025 B1E0 B910
92FC 1C34 19BF 1BF9 8D6D