

USB Device Drivers

A Stepping Stone into your Kernel

Moritz Jodeit, Martin Johns



Agenda

- USB intro
- Motivation
- Attack surface
- Vulnerability identification
 - Hardware-aided approach
 - Emulated environment
- Crash analysis
- Some findings
- Conclusion

Who we are?

- Martin Johns <martin.johns@sap.com>
 - Senior Security Researcher at SAP
- Moritz Jodeit <moritz.jodeit@nruns.com>
 - Bug hunter / security researcher
 - Security Consultant at n.runs AG

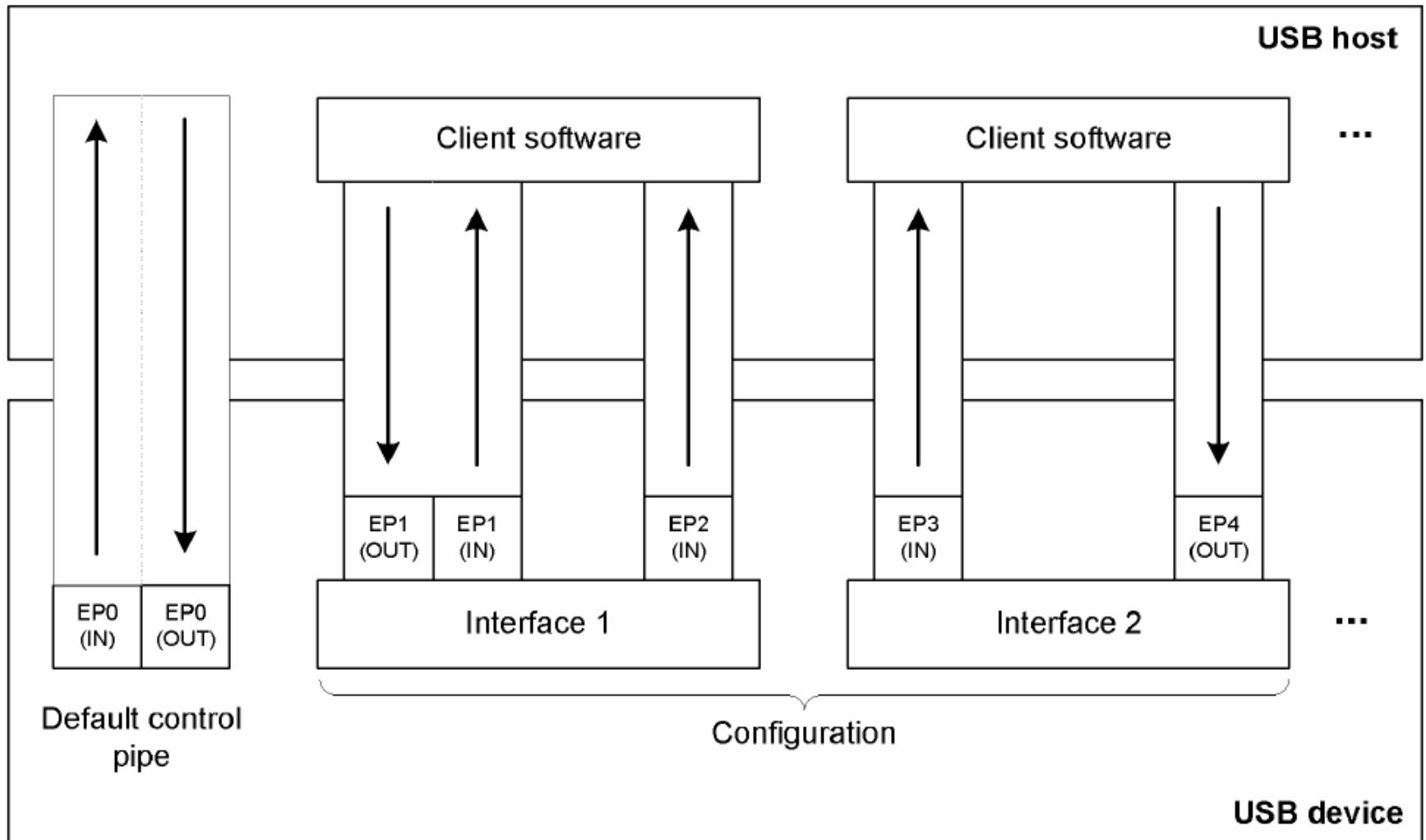
USB intro



USB concepts

- Host / device
- Enumeration
- Descriptors
- USB lingo
 - Endpoints
 - Pipes
 - Interfaces
 - Configurations

USB overview



Motivation

- Social engineering attacks
- Gain access to locked workstations
 - USB device enumeration starts even while workstation is locked!
- Digital voting pen
- Wireless USB (CWUSB)
- Unprotected USB ports...

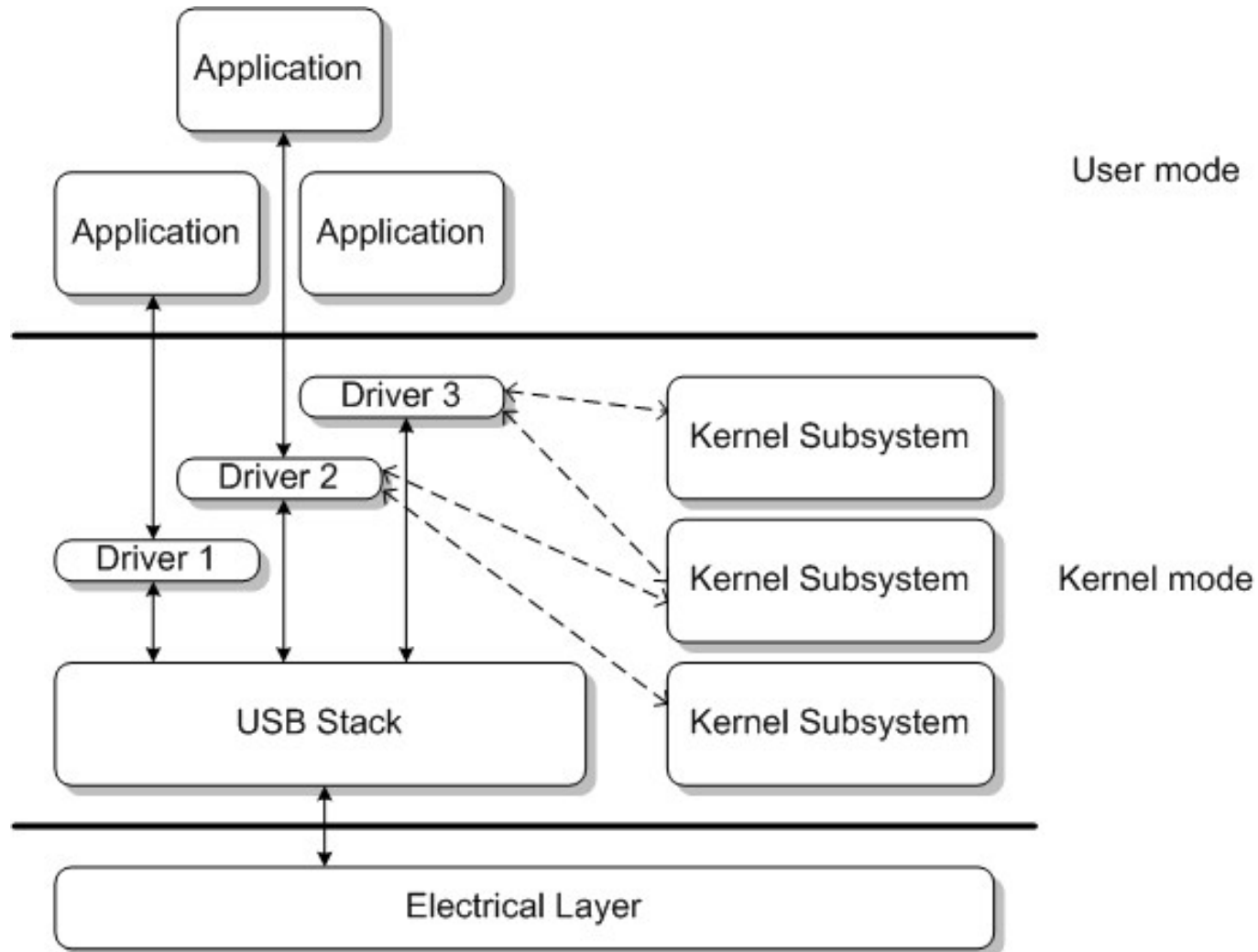
Motivation



Attacks

- Data leakage
- AutoRun malware
 - U3 flash drives
- Malicious USB mouse/keyboard
- OS attacks
 - LNK vulnerability
- Bugs in USB stacks and device drivers
 - PSJailbreak

Attack surface



Vulnerability identification

- Hardware fuzzer
- USB over IP
- Emulated environments

Hardware fuzzer

- Direct connection to target
 - No middle layer which could influence results
 - Embedded devices can be fuzzed
- Disadvantages
 - Fuzzing target might stop responding
 - Fuzzing EP0 on Windows XP (SP2)
 - Inflexible during development

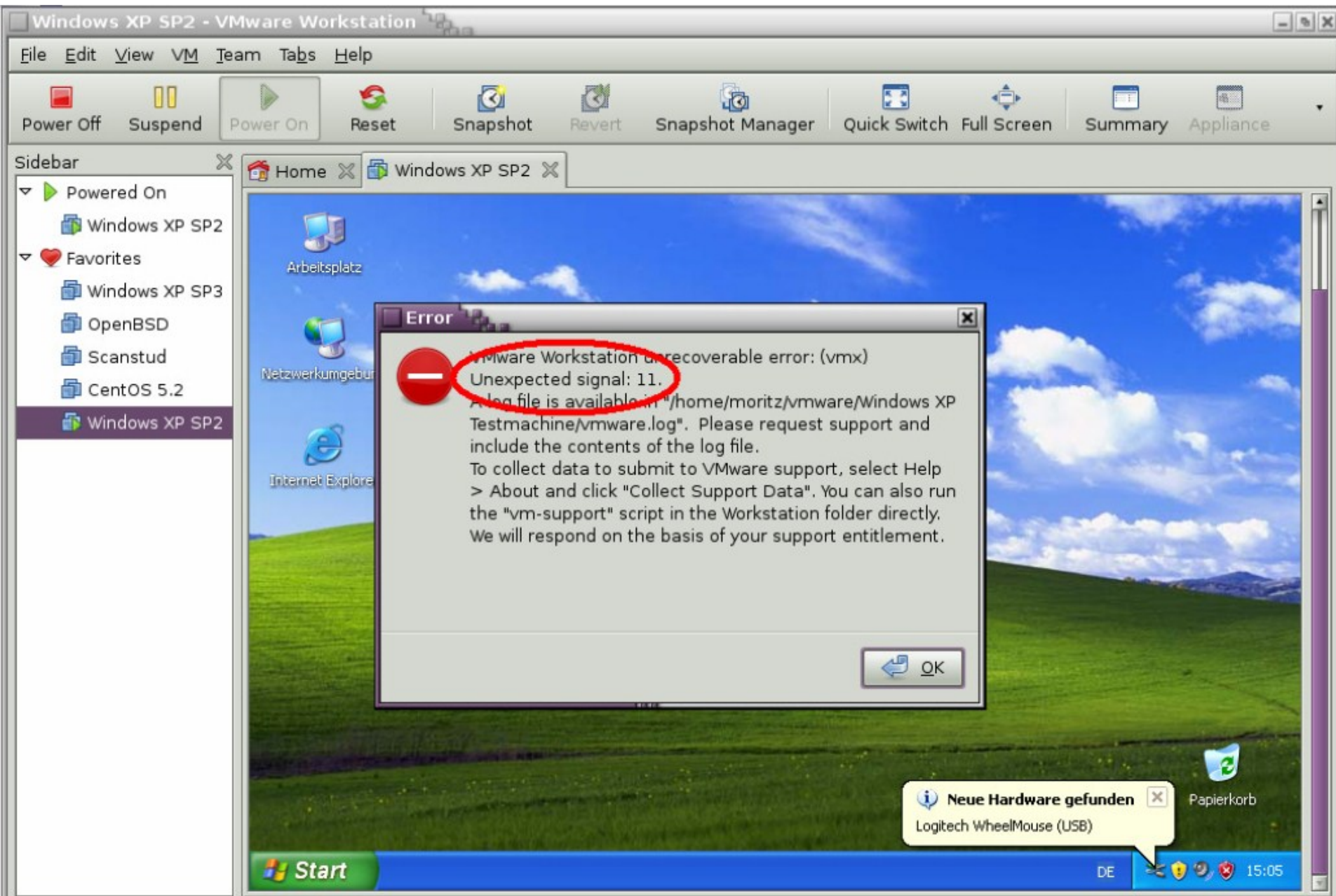
USB over IP

- Use of USB over IP bridge
- Easy access to raw USB packets
 - Existing fuzzers / fuzzing frameworks can be used
 - USB hardware sniffer
- All bridges we know of require software on the host :(
- Hardware USB-IP-USB bridge anyone?

Emulated environments

- Good target monitoring capabilities
- Virtual machine snapshots
 - Quickly recover non-responding target
 - Easy way to reproduce crashes
- Use of high level languages
- (Interesting) side effects...

...bugs in virtualisation software

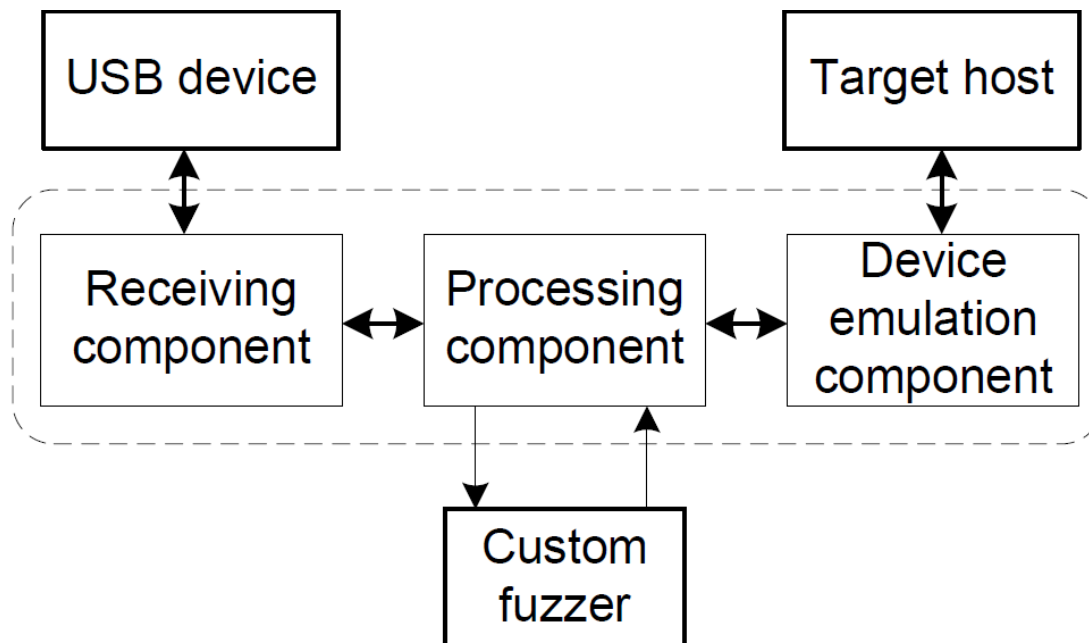


Fuzzing

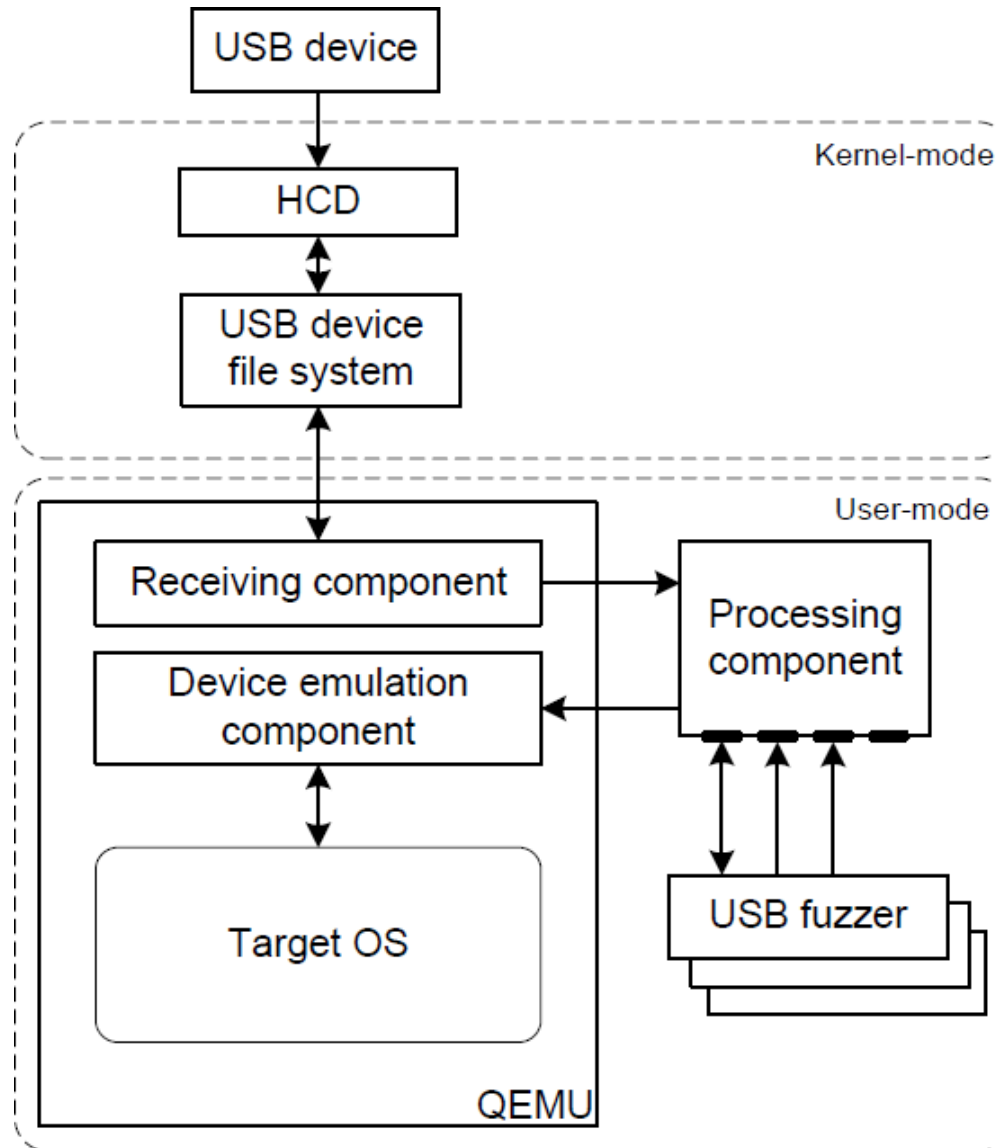
- Generation-based fuzzing
 - Time consuming
 - New device firmware
 - Good code coverage
- Mutation-based fuzzing
 - Good for first quick results
 - USB man-in-the-middle fuzzing

Fuzzing in emulated environments

- First approach
 - Implemented as a patch to Qemu
 - Complete fuzzing logic implemented in python
 - Easy development of custom fuzzers



Fuzzing in emulated environments



Fuzzing in emulated environments

- Current framework centered around Qemu
 - Nothing restricts us to Qemu
- We also implemented a more generic prototype
 - Based on the same three components

Universal man-in-the-middle fuzzer

- Based on USB device file system
- All USB communication passes through usbfs (/proc/bus/usb)
- Syscall interception (ptrace)
 - Fuzz data before it is passed to the virtualisation software
- Universal solution (Qemu, Vmware, ...)
 - No modifications needed

Universal man-in-the-middle fuzzer

- Atomic device attachment/detachment
 - Qemu
 - `usb_add host:0123:4567`
 - `usb_del host:0123:4567`
 - Vmware
 - No VIX API available (AFAIK)
 - Re-attachment can be triggered by starting/stopping the VM

```
=> IOCTL_USB_REAPURBNDELAY (urb.actual_length=9,urb.buffer_length=17)
```

```
[*] Fuzzing URB data in packet 8
```

```
80 06 00 02 00 00 09 00 09 9d 7e cc 03 d2 00 80 .....~.....
```

```
97
```

```
=> IOCTL_USB_RESET
```

```
=> IOCTL_USB_REAPURBNDELAY
```

```
=> IOCTL_USB_SUBMITURB urb[type=2,
```

```
=> IOCTL_USB_REAPURBNDELAY (urb.ac
```

```
[*] Fuzzing URB data in packet 9
```

```
c2 06 00 02 0c 00 09 dd 09 85 7e 01
```

```
fa
```

```
=> IOCTL_USB_REAPURBNDELAY
```

```
=> IOCTL_USB_SUBMITURB urb[type=2,
```

```
=> IOCTL_USB_REAPURBNDELAY (urb.ac
```

```
[*] Received URB packet 10
```

```
=> IOCTL_USB_RESET
```

```
=> IOCTL_USB_REAPURBNDELAY
```

```
=> IOCTL_USB_SETCONFIG 1
```

```
=> IOCTL_USB_GETDRIVER
```

```
=> IOCTL_USB_SUBMITURB urb[type=2,
```

```
=> IOCTL_USB_REAPURBNDELAY (urb.ac
```

```
[*] Fuzzing URB data in packet 11
```

```
80 06 01 03 09 ec 04 00 16 c2 c3 00
```

```
=> IOCTL_USB_REAPURBNDELAY
```

```
=> IOCTL_USB_REAPURBNDELAY
```

```
=> IOCTL_USB_RELEASEINTF 0
```

```
=> IOCTL_USB_IOCTL
```

```
=> IOCTL_USB_RELEASEINTF 1
```

```
=> IOCTL_USB_IOCTL
```

```
=> IOCTL_USB_RELEASEINTF 2
```

```
=> IOCTL_USB_IOCTL
```

```
[*] Closing usbfs descriptor 345
```

```
[*] Opening /proc/bus/usb/002/003 (345)
```

```
[*] Reading 9 bytes from usbfs descripto
```

```
09 00 00 00 09 ec 04 00 16
```

```
=> IOCTL_USB_CLAIMINTF 0
```

```
=> IOCTL_USB_CLAIMINTF 1
```

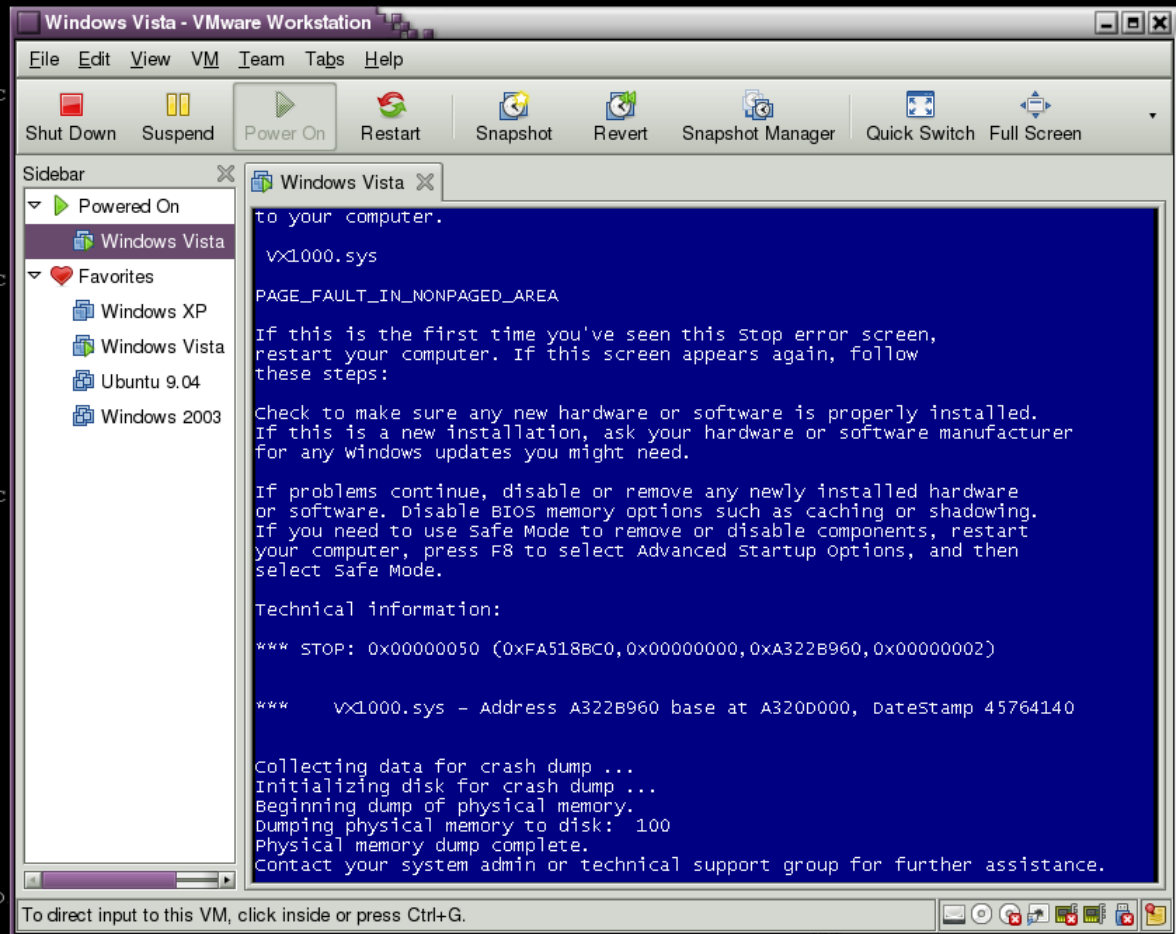
```
=> IOCTL_USB_CLAIMINTF 2
```

```
[*] Process 4449 detached
```

```
[*] Check if guest is still alive
```

```
[*] Guest does not respond... We crashed it :)
```

```
moritz@kindergarten ~/share/projects/usbfsmitm/utils $
```



Crash analysis

- Reproducing a triggered crash
 - Re-apply the same modifications
 - Based on packet number received from host
 - Works best for crashes in enum phase
 - Doesn't really work for crashes after hundreds of packets being exchanged...
 - Replaying the whole communication
 - Works with easy protocols (e.g. HID)
 - Breaks with mass storage devices

Evaluation



- Apple iPod Shuffle
 - Mass storage device (talking SCSI)
 - Lots of communication after attachment
 - User-mode software (iTunes, iPodService.exe)
 - Software available on many systems

Apple iPod Shuffle

- Connected to Windows XP (SP2)
- Double-free of kernel pool memory in usbstor.sys

```
Probably caused by : USBSTOR.SYS ( USBSTOR+dfb )
Followup: MachineOwner
-----
0: kd> !analyze -v
*****
*                                     *
*                               Bugcheck Analysis                               *
*                                     *
*****

BAD_POOL_CALLER (c2)
The current thread is making a bad pool request. Typically this is at a bad IRQL
Arguments:
Arg1: 00000007, Attempt to free pool which was already freed
Arg2: 00000cd4, (reserved)
Arg3: 940082a5, Memory contents of the pool block
Arg4: 86e70008, Address of the block of pool being deallocated
```

- Kernel pool memory corruption in disk.sys
 - While reading the partition table
- Two unclassified bug checks in usbuhci.sys
- Crash in iTunes iPodService.exe
 - NULL pointer deref



Various other devices



- Microsoft LifeCam VX-1000
 - Kernel oops on Ubuntu 9.04
 - NULL pointer deref in SN9C102 driver
 - NULL pointer deref on Windows Vista (SP2)
 - Inside vx1000.sys driver
- Various USB flash drives
 - NULL pointer deref on Windows Vista (SP2)
 - Inside the usbhub.sys driver
 - Call to NULL function pointer

Conclusion

- Fuzzing in emulated environment seems like the right approach
- Reproduction of crashes can be hard sometimes
- Potential for more vulns to be discovered
 - More intelligent fuzzing
 - 3rd party drivers?

Questions?



- Code will be published when ready...
 - Drop us a line, if you want to be notified (moritz.jodeit@nruns.com)