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 am I?

• Moritz Jodeit <moritz@jodeit.org>
  – Bug hunter / security researcher
  – Penetration tester at n.runs AG
  – Living in Hamburg, Germany
USB intro
USB concepts

• Host / device
• Enumeration
• Descriptors
• USB lingo
  – Endpoints
  – Pipes
  – Interfaces
  – Configurations
USB overview

USB host

Client software

Interface 1

Client software

Interface 2

Default control pipe

Configuration

USB device
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
• Bugs in USB stacks and device drivers
Attack surface
Vulnerability identification

- Hardware fuzzer
- Hardware-aided software fuzzer
- Emulated environments
- USB over IP
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
Hardware-aided software solution

• Linux-USB Gadget API Framework
  – Peripheral controller drivers
  – Gadget drivers
    • Ethernet
    • Mass storage
    • Serial
    • MIDI
    • GadgetFS

• Peripheral controller
  – Netchip NET2280
  – PCI evaluation board
Hardware-aided software solution
Hardware-aided software solution

• Linux-USB Gadget API Framework
  – Disadvantages
    • Encountered various dead locks on fuzzing host
    • Main focus doesn’t seem to be fuzzing ;-)
    • Still bad target control
  – Can be used to build the final exploit
    • No firmware writing required
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
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 :(
- Currently planning our own USB-IP-USB bridge
  - Work in progress
Fuzzing

• Generation-based fuzzing
  – Time consuming
    • New device firmware
    • New Linux gadget driver
  – 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

• Disadvantages of first approach
  – Restricted to Qemu
  – Maintaining patches is no fun

• We can do better...
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

- Automic 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/stoping the VM
[*] Fuzzing URB data in packet 8
  80 06 00 02 00 00 00 09 00 09 dc 76 cc 03 d2 00 80 ............
  97
  => IOCTL_USB_RESET
  => IOCTL_USB_REAPUBNDELY
  => IOCTL_USB_SUBMITURB urb[type=2, urb.ac
  => IOCTL_USB_REAPUBNDELY
[*] Fuzzing URB data in packet 9
  c2 06 00 02 0c 00 09 dd 09 85 7e 01
  fa
  => IOCTL_USB_REAPUBNDELY
  => IOCTL_USB_SUBMITURB urb[type=2, urb.ac
  => IOCTL_USB_REAPUBNDELY
[*] Received URB packet 10
  => IOCTL_USB_RESET
  => IOCTL_USB_REAPUBNDELY
  => IOCTL_USB_SUBMITURB urb[type=2, urb.ac
  => IOCTL_USB_REAPUBNDELY
[*] Fuzzing URB data in packet 11
  80 06 01 03 09 ac 04 00 16 c2 c3 00
  => IOCTL_USB_RESET
  => IOCTL_USB_REAPUBNDELY
  => IOCTL_USB_SUBMITURB urb[type=2, urb.ac
  => IOCTL_USB_REAPUBNDELY
[*] Closing usbfs descriptor 345
[*] Opening /proc/bus/usb/002/003 (345)
[*] Reading 9 bytes from usbfs descriptor
  09 00 00 00 09 ac 04 00 16
  => IOCTL_USB_CLAIMINF 0
  => IOCTL_USB_CLAIMINF 1
  => IOCTL_USB_CLAIMINF 2
[*] Process 4449 detached
[*] Check if guest is still alive
[*] Guest does not respond... We crashed it :)

moritz@kindergarten:~$
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

• 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 *
* *
*******************************************************************************

EAP_POOL_CALER [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: 00000000. (reserved)
Arg3: 9e008ea5. Memory contents of the pool block
Arg4: 8e70008. Address of the block of pool being deallocated

• Kernel pool memory corruption in disk.sys
  – While reading the partition table
• Crash in iTunes iPodService.exe
  – NULL pointer deref
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 mass storage devices

- NULL pointer deref on Windows Vista (SP2)
  - Inside the usbhub.sys driver
- Function pointer set to NULL
  - call 0x00000000
  - Not reproducible using current approach :(
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?

• Fuzzer will be published when ready...
  – Drop me a line, if you want to be notified
    (moritz@jodeit.org)