Implementing an USB Host Driver Fuzzer

Daniel Mende – dmende@ernw.de
Disclaimer

- This is an implementation talk.

- We (still) haven’t finished testing.

- No exploits were given that day.
Agenda

- USB Basics
- Facedancer Hardware
- dizzy Fuzzing Toolkit
- dizzy USB Additions
- Practical USB Fuzzing
- Results
USB BASICS
USB History

- Development of the first specification started in 1994.
  - USB version 1.0 released in 1996
  - USB version 1.1 released in 1998
  - USB version 2.0 released in 2001
  - USB version 3.0 released in 2008
USB History

- Developed by a group of companies: Compaq, DEC, IBM, Intel, Microsoft, NEC, and Nortel

- Standardized by the USB Implementers Forum (USBIF)
USB Versions

- Three different versions defined:
  - USB 1.1 with 1.5Mb/s
  - USB 2.0 with 480Mb/s
  - USB 3.0 with 4Gb/s

- Only USB 1.1 and USB 2.0 are addressed with this talk.
USB Descriptor

- Device descriptor identifies the USB device.

- Followed by more descriptors:
  - Configuration Descriptor
  - Interface Descriptor
  - Endpoint Descriptor
  - String Descriptor

- Signals the Host which driver to use.
Device Descriptor

- DeviceDescriptor
  - bNumConfigurations
    - Configuration Descriptor
      - bNumInterfaces
        - Interface Descriptor
          - bNumEndpoints
            - Endpoint Descriptor
          - Endpoint Descriptor
        - Interface Descriptor
          - bNumEndpoints
            - Endpoint Descriptor
          - Endpoint Descriptor
      - Interface Descriptor
        - bNumEndpoints
          - Endpoint Descriptor
        - Endpoint Descriptor
    - Configuration Descriptor
      - bNumInterfaces
        - Interface Descriptor
          - bNumEndpoints
            - Endpoint Descriptor
          - Endpoint Descriptor
USB Enumeration

- Device descriptor is requested.
- Selected configuration descriptor is requested.
- String descriptors referenced in device and configuration descriptor are requested (Manufacturer, Product, eg.).
USB Enumeration

<table>
<thead>
<tr>
<th>Host Address</th>
<th>USB Version</th>
<th>USB Type</th>
<th>Request Type</th>
<th>Request Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.000000</td>
<td>1.0</td>
<td>USB</td>
<td>36 GET DESCRIPTOR Request DEVICE</td>
<td></td>
</tr>
<tr>
<td>2 0.000001.0</td>
<td></td>
<td>host</td>
<td>46 GET DESCRIPTOR Response DEVICE</td>
<td></td>
</tr>
<tr>
<td>3 0.000007</td>
<td>1.0</td>
<td>USB</td>
<td>36 GET DESCRIPTOR Request CONFIGURATION</td>
<td></td>
</tr>
<tr>
<td>4 0.000001.0</td>
<td></td>
<td>host</td>
<td>37 GET DESCRIPTOR Response CONFIGURATION</td>
<td></td>
</tr>
<tr>
<td>5 0.000007</td>
<td>1.0</td>
<td>USB</td>
<td>36 GET DESCRIPTOR Request CONFIGURATION</td>
<td></td>
</tr>
<tr>
<td>6 0.000001.0</td>
<td></td>
<td>host</td>
<td>163 GET DESCRIPTOR Response CONFIGURATION</td>
<td></td>
</tr>
<tr>
<td>7 0.000007</td>
<td>1.0</td>
<td>USB</td>
<td>36 GET DESCRIPTOR Request STRING</td>
<td></td>
</tr>
<tr>
<td>8 0.000001.0</td>
<td></td>
<td>host</td>
<td>66 GET DESCRIPTOR Response STRING</td>
<td></td>
</tr>
<tr>
<td>9 0.015601.0</td>
<td>1.0</td>
<td>USB</td>
<td>36 GET DESCRIPTOR Request STRING</td>
<td></td>
</tr>
<tr>
<td>10 0.015601.0</td>
<td></td>
<td>host</td>
<td>50 GET DESCRIPTOR Response STRING</td>
<td></td>
</tr>
<tr>
<td>11 15.6624</td>
<td>1.0</td>
<td>USB</td>
<td>36 GET DESCRIPTOR Request STRING</td>
<td></td>
</tr>
<tr>
<td>12 15.6624.1</td>
<td></td>
<td>host</td>
<td>66 GET DESCRIPTOR Response STRING</td>
<td></td>
</tr>
</tbody>
</table>
Well, how do matters stand with security?
USB Vulnerabilities

- Windows USB Descriptor Vulnerability
  - CVE-2013-1285
  - CVE-2013-1286
  - CVE-2013-1287

- Linux Kernel caiaq USB Drivers Buffer Overflow
  - CVE-2011-0712
USB Vulnerabilities (cont.)

- Solaris USB configuration descriptor kernel stack overflow
  - CVE-2011-2295
- usbmuxd 1.0.7 Buffer Overflow Vulnerability
  - CVE-2012-0065
FACEDANCER HARDWARE
Facedancer

- External USB testing hardware.
- Allows to send raw USB PDUs.
- Consists of:
  - FT232R USB to serial UART (Host connect)
  - MSP430F2618 16bit µc
  - MAX3421E USB controller (Target connect)
Facedancer
Facedancer
Facedancer

- Controlled from the host PC, using a python library.
- Implements basic USB protocol layer as well.

http://goodfet.sourceforge.net/hardware/facedancer21/
DIZZY FUZZING TOOLKIT
Fuzzing or Fuzz Testing is a software negative testing technique. It uses the fault injection approach, which basically injects wrong data into the tested program. Fuzzing can be used to test parser of any kind like protocol parser, file format parser, language parser, etc."

B. Miller, L. Fredriksen, and B. So. An empirical study of the reliability of unix utilities.
Dizzy

- Python based fuzzing toolkit.
- First release in 2011.
- Simple packet description syntax.
- Does state less and state full fuzzing.

http://c0decafe.de/tools/dizzy-0.8.2.tar.bz2
- Defined in Python syntax.

```python
name = "empty"
objects = []
functions = []
```
Dizzy packet description

- The *object* array contains the fields of the packet.
- The *functions* array contains operations performed on the packet before its sent to the target.
Dizzy packet description

- Available objects are:
  - field, list, rand, link, fill and padding.

- Available functions are:
  - time, time_no_fracs, length, lambda_length, csum, lambda_csum

• See dizzy README for details.
.dizz file example

name = "example"

objects = [
    field("type", 8, "\x00", "full"),
    field("length", 8, "\x06", "full"),
    field("value", None, "fuzz", "std"),
]

functions = [
    length("length", "type", "value"),
]
arp.dizz

name = "arp"

objects = [
    field("hw_type", 16, "\x00\x01", "full"),
    field("proto_type", 16, "\x08\x00", "full"),
    field("hw_size", 8, "\x06", "full"),
    field("proto_size", 8, "\x04", "full"),
    field("opcode", 16, "\x00\x01", "full"),
    field("mac_src", 48, "\x01\x02\x03\x04\x05\x06", "none"),
    field("ip_src", 32, "\xc0\xa8\x5f\xb5", "none"),
    field("mac_dst", 48, "\x00\x00\x00\x00\x00\x00", "none"),
    field("ip_dst", 32, "\xc0\xa8\x5f\xb6", "none"),
] functions = []
more objects

objects = [
    rand("nonce", 8*16),
    list("list1", "default", "~/list1.txt"),
    field("val1", None, "fuzz", "std"),
    link("nonce_again", "nonce"),
]
padding example

```python
field("val", 8, "\x01", "none"),
field("val2", None, "\x02", "std"),
padding("pad", "val", "val2", 8*8, "\x00"),
```
Interaction concept

- State full fuzzing is represented as a series of packets (.dizz files).
- Packets are sent out in order and an answer is read between.
- Parts of an answer can be extracted and used in further transmissions.
Dizzy interaction description

name = "testact"

objects = [
    dizz("first", "dizzes/first.dizz"),
    dizz("second", "dizzes/second.dizz"),
]

functions = []
Dizzy interaction description

name = "act1"

objects = [
    dizz("first", "dizzes/first.dizz"),
    dizz("auth", "dizzes/auth.dizz"),
    dizz("command", "dizzes/command.dizz"),
]

functions = [
    copy(2, "auth", 0x10, 0x20),
]
DIZZY USB ADDITIONS
Dizzy Output Type

dizzy.py

- Two new output types:
  - usb-desc for USB descriptor fuzzing.
  - usb-endp for USB endpoint fuzzing.
- Added in *dizz_session* class.
Dizzy – Facedancer glue

- `usb.py`

- Implements the USBDevice and USBInterface classes required by the Facedancer USB lib.
- `dizzUSB` class is used to start/stop/control the USB thread and descriptor payload.
PRACTICAL USB FUZZING
There are different targets on USB:
- Device Descriptor
- Configuration Descriptor
- Endpoints
USB fuzzing

- Descriptor fuzzing targets the host OS USB stack.
- Endpoint fuzzing targets the USB device driver (either from the OS vendor or third party).
Were to get the descriptors

- We are setting up an USB device/configuration descriptor collection at [usbdescriptors.com](http://usbdescriptors.com).
- You can contribute to the collection by submitting:
  - The output of `lsusb -v`
  - The raw descriptors
What you need

- **Hardware:**
  - Host PC
  - Facedancer Board
  - Two USB cables
  - Target PC
What you need

- **Software:**
  - Python and dizzy on the host.
  - USB device file.
  - Dizzy packet description file.
Step I

Create the USB descriptor file from the device you want to emulate:

```
lsusb -s 1:1 -v | perl parse_lsusb.pl > DEVICENAME.usb
```
Step II

- Create / Modify your .dizz file:

```bash
vim dizzes/usb/configuration_descriptor.dizz
```
Step III

- Attach the Facedancer board to the host PC and to the target PC.
Step IV

- Start dizzy:

dizzy.py -o usb-desc -d DEVICENAME.usb -e CD configuration_descriptor.dizz
Monitor USB traffic via wireshark.
  - On Windows USBPcap is needed: http://desowin.org/usbpcap/
RESULTS
Results

- We didn’t perform wide scale fuzzing.
- Still, some unexpected behavior appeared during functional testing of the code ;
Results

- Host (USB Stack) crash
  - Win7 (x64), Linux 3.10 (x64)

- Software crashes on target:
  - Skype, VMware, etc...
Results

Unable to connect the USB device "(null)" to the virtual machine (error code 13). Contact VMware, Inc. technical support for assistance.
Results

- We didn’t investigate those crashes, as getting the implementation done was the main prio.
- We will investigate them in the future, no worries (;
Conclusions

- In the past expensive hardware was needed to perform USB testing.
- With low cost hardware available more people can test USB implementations.
- Expect a bunch of new USB vulnerabilities to show up.
There’s never enough time...

THANK YOU...

...for yours!
Q&A
You have Questions
We have Answers