Risky USBusiness

Say ”what the fuzz.”... If you can’t say it, you can’t do it.

Jordan BOUYAT
jbouyat@quarkslab.com
@la_F0uin3

Fernand LONE-SANG
flonesang@quarkslab.com

Hack.lu, October 22, 2014

QUARKSLAB
INNOVATIVE SECURITY

Observation

USB ubiquity

- Workstations;
- Interactive machines;
- Printers;
- Embedded systems;
- Etc.

Massively used, but internals are not well known.

Interest

Possible attacks

USB devices are attack vectors:

- Physical access in limited time;
- Device deliberately left behind;
- Attacks on isolated networks.

Summary

1. USB basics
2. Fuzzing approaches
3. Our tool
4. Results
5. Conclusion

Table of contents

1. USB basics
2. Fuzzing approaches
3. Our tool
4. Results
5. Conclusion

Hierarchy

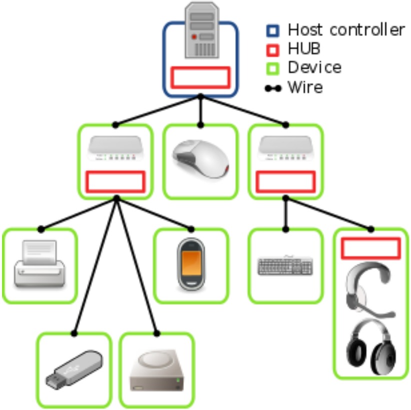
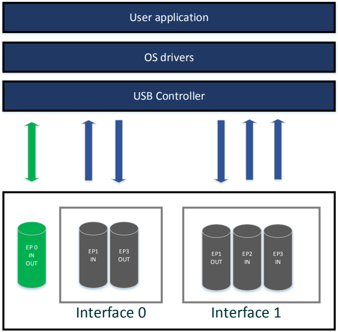


Figure: USB topology

- An ordered topology
- 1 host controller: 127 devices
- One hub can be connected to another
- Connections and transfers are initiated by a host only (except OTG)

Device logical view

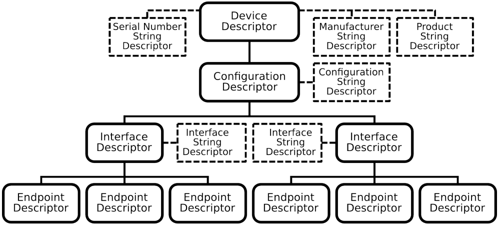


- An interface provides a function
- It contains endpoints
- Endpoints are logical links between the device and the host drivers
- They are unidirectional. Four kinds of transfer are available:
  - Control
  - Interrupt
  - Bulk
  - Isochronous

Descriptors

Data structures that describe the device:

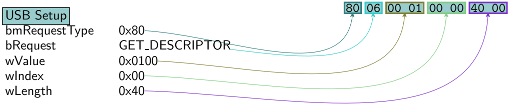
1. Its characteristics (USB version, VID, PID...);
2. Its interfaces (type, endpoint numbers...);
3. Its endpoints (direction, transfert type...).



A configuration descriptor corresponds to different associations of configuration.

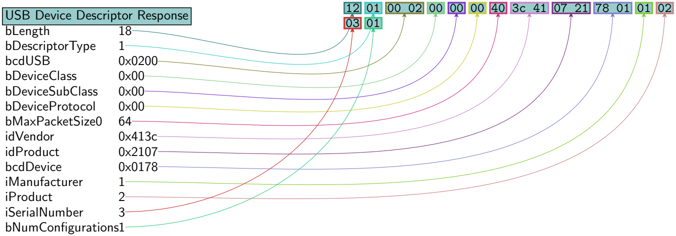
Standard requests

Descriptors are retrieved during the **enumeration** process.



80 06 00 01 00 00 40 00

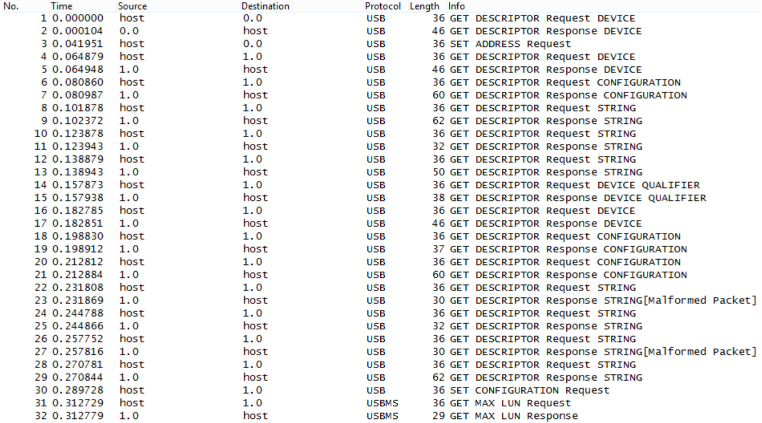
USB Setup	
bmRequestType	0x80
bRequest	GET_DESCRIPTOR
wValue	0x0100
wIndex	0x00
wLength	0x40



12 01 00 02 00 00 00 40 3c 41 07 21 78 01 01 02
03 01

USB Device Descriptor Response	
bLength	18
bDescriptorType	1
bcdUSB	0x0200
bDeviceClass	0x00
bDeviceSubClass	0x00
bDeviceProtocol	0x00
bMaxPacketSize0	64
idVendor	0x413c
idProduct	0x2107
bcdDevice	0x0178
iManufacturer	1
iProduct	2
iSerialNumber	3
bNumConfigurations	1

Enumeration



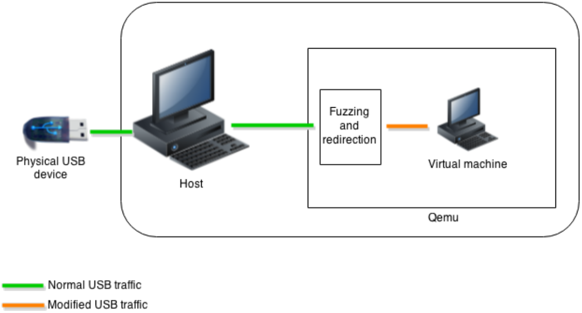
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	host	0.0	USB	36	GET DESCRIPTOR Request DEVICE
2	0.000104	0.0	host	USB	46	GET DESCRIPTOR Response DEVICE
3	0.041951	host	0.0	USB	36	SET ADDRESS Request
4	0.064879	host	1.0	USB	36	GET DESCRIPTOR Request DEVICE
5	0.064948	1.0	host	USB	46	GET DESCRIPTOR Response DEVICE
6	0.080860	host	1.0	USB	36	GET DESCRIPTOR Request CONFIGURATION
7	0.080987	1.0	host	USB	60	GET DESCRIPTOR Response CONFIGURATION
8	0.101878	host	1.0	USB	36	GET DESCRIPTOR Request STRING
9	0.102372	1.0	host	USB	62	GET DESCRIPTOR Response STRING
10	0.123878	host	1.0	USB	36	GET DESCRIPTOR Request STRING
11	0.123943	1.0	host	USB	32	GET DESCRIPTOR Response STRING
12	0.138879	host	1.0	USB	36	GET DESCRIPTOR Request STRING
13	0.138943	1.0	host	USB	50	GET DESCRIPTOR Response STRING
14	0.157873	host	1.0	USB	36	GET DESCRIPTOR Request DEVICE QUALIFIER
15	0.157938	1.0	host	USB	38	GET DESCRIPTOR Response DEVICE QUALIFIER
16	0.182785	host	1.0	USB	36	GET DESCRIPTOR Request DEVICE
17	0.182851	1.0	host	USB	46	GET DESCRIPTOR Response DEVICE
18	0.198830	host	1.0	USB	36	GET DESCRIPTOR Request CONFIGURATION
19	0.198912	1.0	host	USB	37	GET DESCRIPTOR Response CONFIGURATION
20	0.212812	host	1.0	USB	36	GET DESCRIPTOR Request CONFIGURATION
21	0.212884	1.0	host	USB	60	GET DESCRIPTOR Response CONFIGURATION
22	0.231808	host	1.0	USB	36	GET DESCRIPTOR Request STRING
23	0.231869	1.0	host	USB	30	GET DESCRIPTOR Response STRING[Malformed Packet]
24	0.244788	host	1.0	USB	36	GET DESCRIPTOR Request STRING
25	0.244866	1.0	host	USB	32	GET DESCRIPTOR Response STRING
26	0.257752	host	1.0	USB	36	GET DESCRIPTOR Request STRING
27	0.257816	1.0	host	USB	30	GET DESCRIPTOR Response STRING[Malformed Packet]
28	0.270781	host	1.0	USB	36	GET DESCRIPTOR Request STRING
29	0.270844	1.0	host	USB	62	GET DESCRIPTOR Response STRING
30	0.289728	host	1.0	USB	36	SET CONFIGURATION Request
31	0.312729	host	1.0	USBMS	36	GET MAX LUN Request
32	0.312779	1.0	host	USBMS	29	GET MAX LUN Response

Table of contents

1. USB basics
2. Fuzzing approaches
3. Our tool
4. Results
5. Conclusion

Qemu: configuration 1

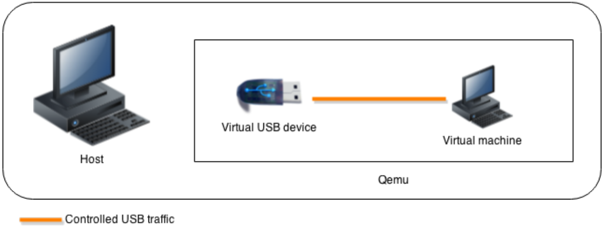
Dumb fuzzer: fuzzing the forwarded traffic between a virtual machine and a physical device.



Experimented by: Fabien Perigaud

Qemu: configuration 2

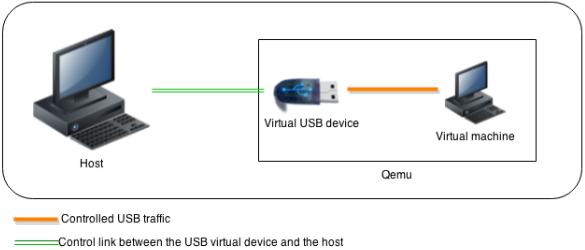
A virtual fuzzer device



Experimented by: MWR Labs

Qemu: configuration 3

USB traffic is forwarded to the host userland by the virtual device. Then it's fuzzed and re-injected.



Experimented by: Tobias Mueller and Sergej Schumilo (vUSBf)

Feedbacks

Pros:

- Restoration of the system to a healthy state using snapshots;
- Better instrumentation and monitoring;
- Easy to parallelize;
- No special hardware needed.

Cons:

- Not all OS can be virtualized;
- Possible bugs in USB implementation in the hypervisor.

Possibilities

Dedicated hardware

Pros: Low level capture/replay, scripting language
Cons: Expensive, inflexible API
Example: Totalphase Beagle USB*

Microcontrollers and FPGAs

Pro: Cheap
Con: You need to re-flash each time you make a modification of the code
Examples: PIC, AVR (like Teensy with LUFA library), Daisho for the FPGA

A compromise: the Facedancer?

Facedancer

Introduction

- Developped by Travis Goodspeed
- Contains a serial/USB adapter, a MSP430 microcontroller and a USB controller
- Allows USB device emulation by controlling it with Python scripts running on a remote machine

Figure: http://int3.cc/

Limitations

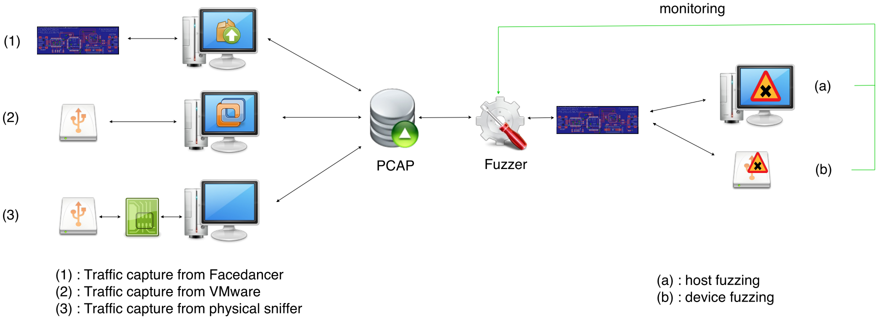
- Only 3 endpoints
- No isochronous transfer support
- Low data rate because of the serial connection over USB
- No USB3 support

However, the Facedancer is enough to begin to fuzz.

Table of contents

1. USB basics
2. Fuzzing approaches
3. Our tool
4. Results
5. Conclusion

Architecture

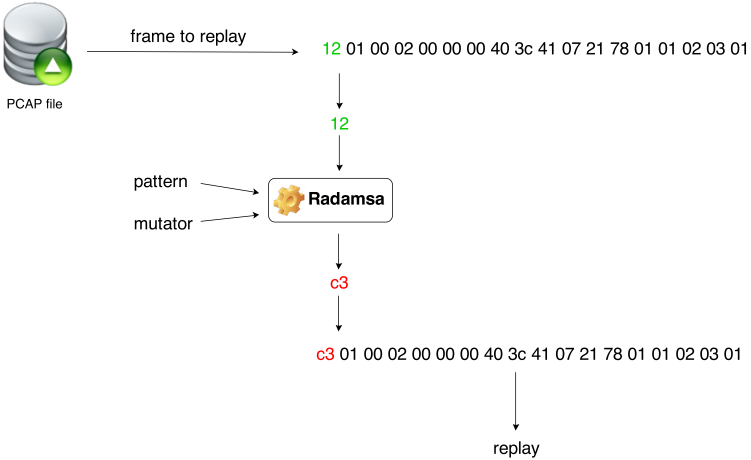


(1) : Traffic capture from Facedancer
(2) : Traffic capture from VMware
(3) : Traffic capture from physical sniffer

(a) : host fuzzing
(b) : device fuzzing

Figure: USB fuzzing architecture

Usage



PCAP file

frame to replay

12 01 00 02 00 00 00 40 3c 41 07 21 78 01 01 02 03 01

12

pattern

mutator

Radamsa

c3

c3 01 00 02 00 00 00 40 3c 41 07 21 78 01 01 02 03 01

replay

Technical details

Base

- Based on the open source tool Umap developed by Andy Davis
- Umap is based on Travis Goodspeed's code

Contribution

Modifications

- PCAP capture and replay
- Mutation of replayed data with Radamsa
- Frame choice, bytes and fuzzing patterns to apply
- Fuzzing monitor with crash report
- Step by step debug mode

Table of contents

1. USB basics
2. Fuzzing approaches
3. Our tool
4. **Results**
5. Conclusion

Results on Windows 8.1

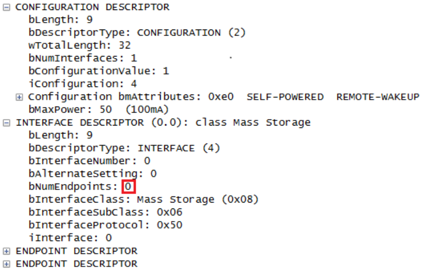
HID parsing

Other bytes values which trigger the same crash of Andy Davis:
Not exploitable

Mass storage device

Wrong control of endpoints number in USBSTOR.sys:
Not exploitable

Mutated descriptor

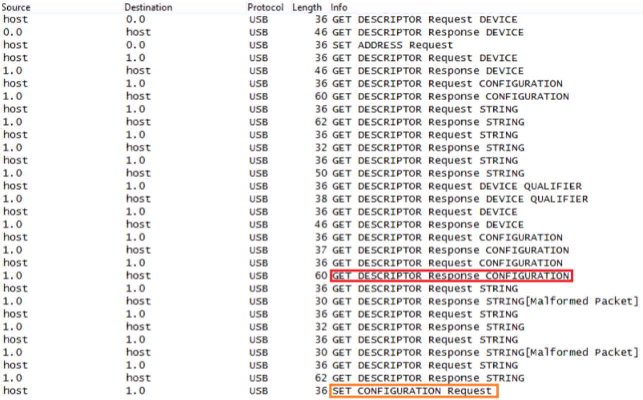


```
⊟ CONFIGURATION DESCRIPTOR
    bLength: 9
    bDescriptorType: CONFIGURATION (2)
    wTotalLength: 32
    bNumInterfaces: 1
    bConfigurationValue: 1
    iConfiguration: 4
  ⊞ Configuration bmAttributes: 0xe0  SELF-POWERED  REMOTE-WAKEUP
    bMaxPower: 50  (100mA)
⊟ INTERFACE DESCRIPTOR (0.0): class Mass Storage
    bLength: 9
    bDescriptorType: INTERFACE (4)
    bInterfaceNumber: 0
    bAlternateSetting: 0
    bNumEndpoints: 0
    bInterfaceClass: Mass Storage (0x08)
    bInterfaceSubClass: 0x06
    bInterfaceProtocol: 0x50
    iInterface: 0
⊞ ENDPOINT DESCRIPTOR
⊞ ENDPOINT DESCRIPTOR
```

Craft of a configuration descriptor providing an interface that contains 0 endpoint.

Result: crash

Enumeration



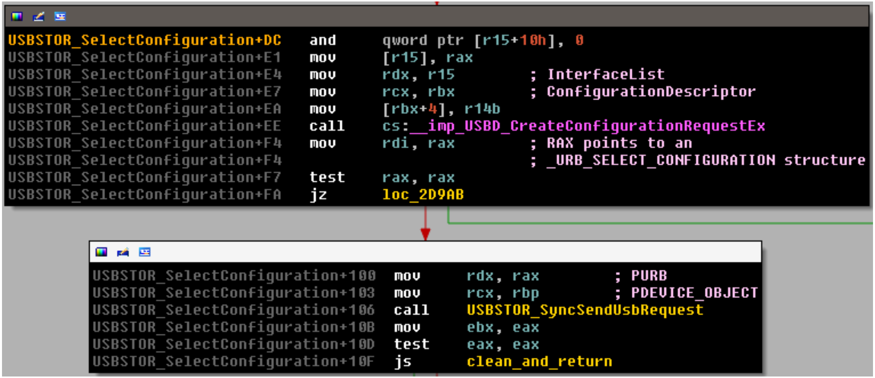
Source	Destination	Protocol	Length	Info
host	0.0	USB	36	GET DESCRIPTOR Request DEVICE
0.0	host	USB	46	GET DESCRIPTOR Response DEVICE
host	0.0	USB	36	SET ADDRESS Request
host	1.0	USB	36	GET DESCRIPTOR Request DEVICE
1.0	host	USB	46	GET DESCRIPTOR Response DEVICE
host	1.0	USB	36	GET DESCRIPTOR Request CONFIGURATION
1.0	host	USB	60	GET DESCRIPTOR Response CONFIGURATION
host	1.0	USB	36	GET DESCRIPTOR Request STRING
1.0	host	USB	62	GET DESCRIPTOR Response STRING
host	1.0	USB	36	GET DESCRIPTOR Request STRING
1.0	host	USB	32	GET DESCRIPTOR Response STRING
host	1.0	USB	36	GET DESCRIPTOR Request STRING
1.0	host	USB	50	GET DESCRIPTOR Response STRING
host	1.0	USB	36	GET DESCRIPTOR Request DEVICE QUALIFIER
1.0	host	USB	38	GET DESCRIPTOR Response DEVICE QUALIFIER
host	1.0	USB	36	GET DESCRIPTOR Request DEVICE
1.0	host	USB	46	GET DESCRIPTOR Response DEVICE
host	1.0	USB	36	GET DESCRIPTOR Request CONFIGURATION
1.0	host	USB	37	GET DESCRIPTOR Response CONFIGURATION
host	1.0	USB	36	GET DESCRIPTOR Request CONFIGURATION
1.0	host	USB	60	GET DESCRIPTOR Response CONFIGURATION
host	1.0	USB	36	GET DESCRIPTOR Request STRING
1.0	host	USB	30	GET DESCRIPTOR Response STRING[Malformed Packet]
host	1.0	USB	36	GET DESCRIPTOR Request STRING
1.0	host	USB	32	GET DESCRIPTOR Response STRING
host	1.0	USB	36	GET DESCRIPTOR Request STRING
1.0	host	USB	30	GET DESCRIPTOR Response STRING[Malformed Packet]
host	1.0	USB	36	GET DESCRIPTOR Request STRING
1.0	host	USB	62	GET DESCRIPTOR Response STRING
host	1.0	USB	36	SET CONFIGURATION Request

Controllers and OS drivers

USBSTOR.sys

Crash analysis

We move in `USBSTOR_SelectConfiguration`.

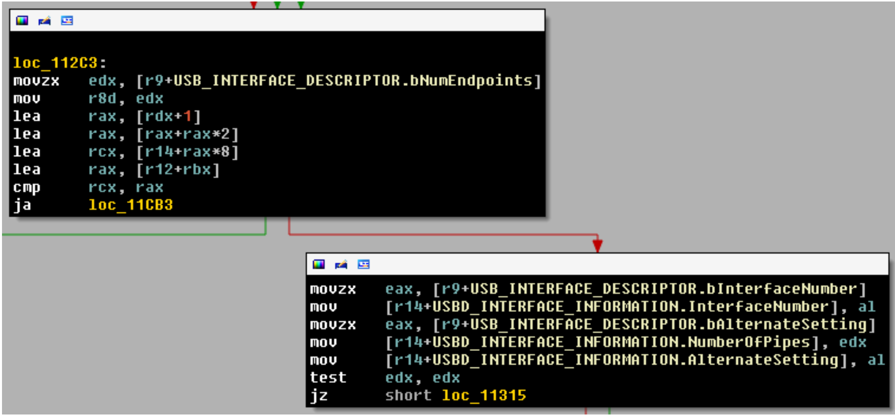


```
USBSTOR_SelectConfiguration+DC    and     qword ptr [r15+10h], 0
USBSTOR_SelectConfiguration+E1    mov     [r15], rax
USBSTOR_SelectConfiguration+E4    mov     rdx, r15         ; InterfaceList
USBSTOR_SelectConfiguration+E7    mov     rcx, rbx         ; ConfigurationDescriptor
USBSTOR_SelectConfiguration+EA    mov     [rbx+4], r14b
USBSTOR_SelectConfiguration+EE    call    cs:__imp_USBD_CreateConfigurationRequestEx
USBSTOR_SelectConfiguration+F4    mov     rdi, rax         ; RAX points to an
USBSTOR_SelectConfiguration+F4                             ; _URB_SELECT_CONFIGURATION structure
USBSTOR_SelectConfiguration+F7    test    rax, rax
USBSTOR_SelectConfiguration+FA    jz      loc_2D9AB
```

```
USBSTOR_SelectConfiguration+100   mov     rdx, rax         ; PURB
USBSTOR_SelectConfiguration+103   mov     rcx, rbp         ; PDEVICE_OBJECT
USBSTOR_SelectConfiguration+106   call    USBSTOR_SyncSendUsbRequest
USBSTOR_SelectConfiguration+10B   mov     ebx, eax
USBSTOR_SelectConfiguration+10D   test    eax, eax
USBSTOR_SelectConfiguration+10F   js      clean_and_return
```

Figure: USBSTOR.sys : `USBSTOR_SelectConfiguration+EE`

Crash analysis



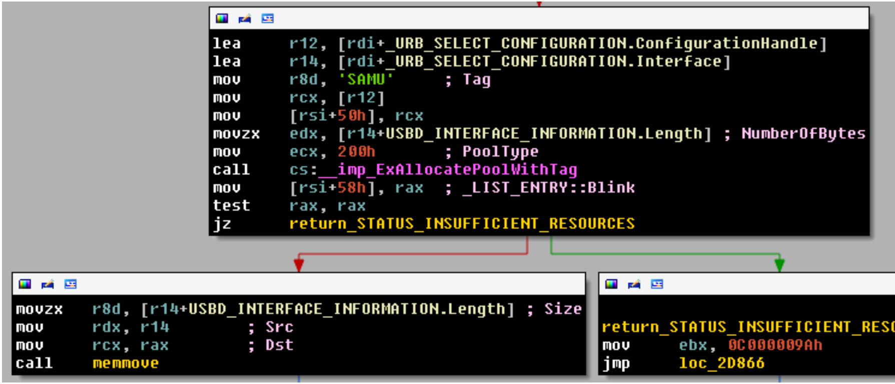
```
loc_112C3:
movzx   edx, [r9+USB_INTERFACE_DESCRIPTOR.bNumEndpoints]
mov     r8d, edx
lea     rax, [rdx+1]
lea     rax, [rax+rax*2]
lea     rcx, [r14+rax*8]
lea     rax, [r12+rbx]
cmp     rcx, rax
ja      loc_11CB3
```

```
movzx   eax, [r9+USB_INTERFACE_DESCRIPTOR.bInterfaceNumber]
mov     [r14+USBD_INTERFACE_INFORMATION.InterfaceNumber], al
movzx   eax, [r9+USB_INTERFACE_DESCRIPTOR.bAlternateSetting]
mov     [r14+USBD_INTERFACE_INFORMATION.NumberOfPipes], edx
mov     [r14+USBD_INTERFACE_INFORMATION.AlternateSetting], al
test    edx, edx
jz      short loc_11315
```

Figure: usbd.sys : USBD_CreateConfigurationRequestEx+113

Duplication of the USB_INTERFACE_DESCRIPTOR.bNumEndpoints field.

Crash analysis



```
lea     r12, [rdi+_URB_SELECT_CONFIGURATION.ConfigurationHandle]
lea     r14, [rdi+_URB_SELECT_CONFIGURATION.Interface]
mov     r8d, 'SAMU'       ; Tag
mov     rcx, [r12]
mov     [rsi+50h], rcx
movzx   edx, [r14+USBD_INTERFACE_INFORMATION.Length] ; NumberOfBytes
mov     ecx, 200h         ; PoolType
call    cs:__imp_ExAllocatePoolWithTag
mov     [rsi+58h], rax    ; _LIST_ENTRY::Blink
test    rax, rax
jz      return_STATUS_INSUFFICIENT_RESOURCES
```

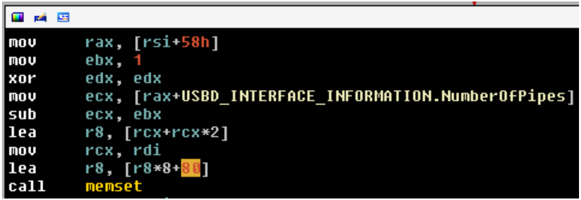
```
movzx   r8d, [r14+USBD_INTERFACE_INFORMATION.Length] ; Size
mov     rdx, r14          ; Src
mov     rcx, rax          ; Dst
call    memmove
```

```
return_STATUS_INSUFFICIENT_RESO
mov     ebx, 0C000009Ah
jmp     loc_2D866
```

Figure: USBSTOR.sys : `USBSTOR_SelectConfiguration+11`

Duplication of `USBD_INTERFACE_INFORMATION` structure.

Crash origin in x64



```
mov     rax, [rsi+58h]
mov     ebx, 1
xor     edx, edx
mov     ecx, [rax+USBD_INTERFACE_INFORMATION.NumberOfPipes]
sub     ecx, ebx
lea     r8, [rcx+rcx*2]
mov     rcx, rdi
lea     r8, [r8*8+80]
call    memset
```

$ECX \longleftarrow$ endpoint number

$ECX \longleftarrow ECX - 1$

$R8 \longleftarrow 3 * RCX$

$R8 \longleftarrow R8 * 8 + 80$

`memset(@dest, 0x0, R8)`

If endpoint number is 0 :

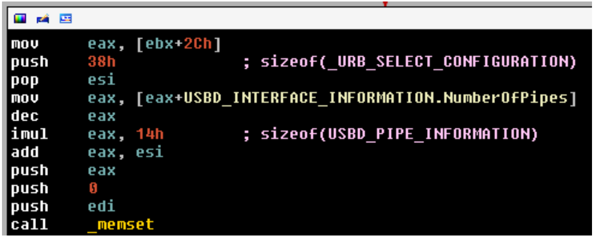
$ECX \longleftarrow 0 - 1 = 0xffffffff$

$R8 \longleftarrow 0xffffffff * 3 = 0x0002fffffffd$

$R8 \longleftarrow 0x0002fffffffd * 8 + 80 = 0x1800000038$

`memset(@dest, 0x0, 0x1800000038)`

x86 problem



```
mov     eax, [ebx+2Ch]
push    38h               ; sizeof(_URB_SELECT_CONFIGURATION)
pop     esi
mov     eax, [eax+USBD_INTERFACE_INFORMATION.NumberOfPipes]
dec     eax
imul    eax, 14h          ; sizeof(USBD_PIPE_INFORMATION)
add     eax, esi
push    eax
push    0
push    edi
call    _memset
```

$EAX \longleftarrow$ endpoint number
$EAX \longleftarrow ECX - 1$
$EAX \longleftarrow EAX * 0x14 + 0x38$
`memset(@dest, 0x0, EAX)`

If endpoint number is 0 :
$EAX \longleftarrow 0 - 1 = 0xffffffff$
$EAX \longleftarrow 0xffffffff * 0x14 + 0x38 = 0x24$
`memset(@dest, 0x0, 0x24)`

The last 20 bytes of the `_URB_SELECT_CONFIGURATION` structure are not initialized.

Table of contents

1. USB basics
2. Fuzzing approaches
3. Our tool
4. Results
5. Conclusion

Conclusion and prospects

Currently

- Functional capture sources: Facedancer and VMware
- Host fuzzing is working

To do

- Improve performances:
  - FPGA
  - ARM board with OTG port for capture/replay using USBGadget
- Implement device fuzzing
- Add other capture sources
- Add USB3 support

Questions?

Thanks to all the QuarksLab team and particularly Fernand Lone-Sang, Kevin Szkudlapski and Damien Aumaître.

QUARKSLAB
INNOVATIVE SECURITY

www.quarkslab.com

contact@quarkslab.com | @quarkslab.com