USB arsenal for masses

Krzysztof Opasiak

Samsung R&D Institute Poland

Agenda

USB protocol intro

USB sniffing & modification

USB security testing

Summary

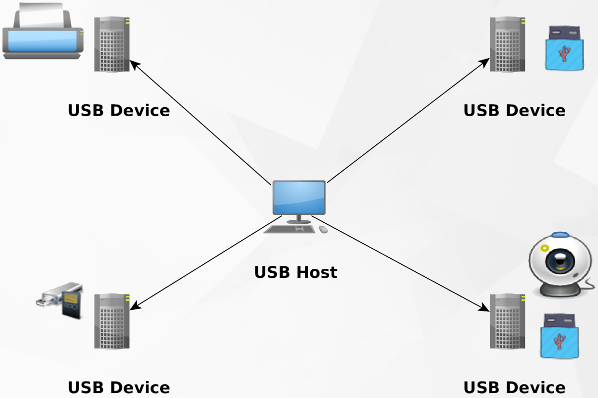
Q & A

USB protocol intro

What USB is about?

It's about providing services!

- Storage
- Printing
- Ethernet
- Camera
- Any other



Endpoints...

- Device may have up to 31 endpoints (including ep0)
- Each of them gets a unique endpoint address
- Endpoint 0 may transfer data in both directions
- All other endpoints may transfer data in one direction:

  IN Transfer data from device to host

  OUT Transfer data from host to device

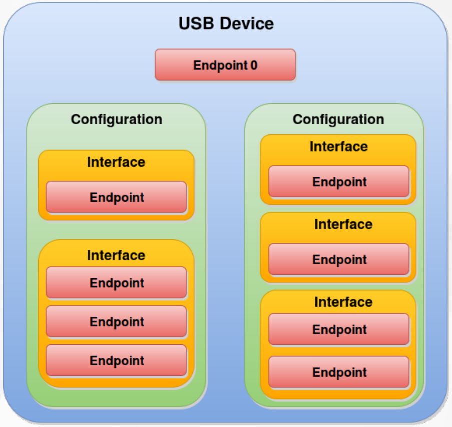
Endpoint types

- **Control**
  - Bi-directional endpoint
  - Used for enumeration
  - Can be used for application
- **Bulk**
  - Used for large data transfers
  - Used for large, time-insensitive data (Network packets, Mass Storage, etc).
  - Does not reserve bandwidth on bus, uses whatever time is left over

Endpoint types

- **Interrupt**
  - Transfers a small amount of low-latency data
  - Reserves bandwidth on the bus
  - Used for time-sensitive data (HID)
- **Isochronous**
  - Transfers a large amount of time-sensitive data
  - Delivery is not guaranteed (no ACKs are sent)
  - Used for Audio and Video streams
  - Late data is as good as no data
  - Better to drop a frame than to delay and force a re-transmission

USB device


USB Device
Endpoint 0
Configuration
Interface
Endpoint
Interface
Endpoint
Endpoint
Endpoint
Configuration
Interface
Endpoint
Interface
Endpoint
Interface
Endpoint
Endpoint

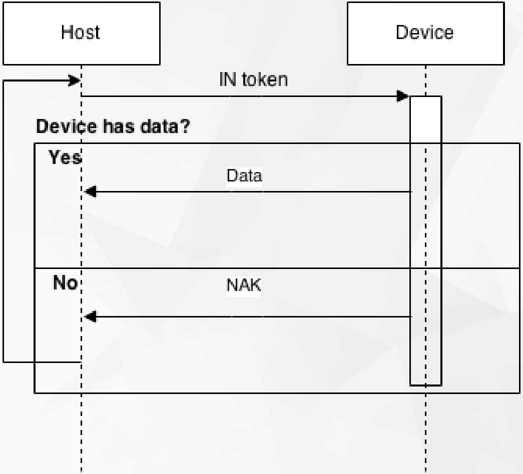
USB bus

- USB is a Host-controlled bus
- Nothing on the bus happens without the host first initiating it.
- Devices cannot initiate any communication.
- The USB is a Polled Bus.
- The Host polls each device, requesting data or sending data.

USB transport (Link Layer)

IN

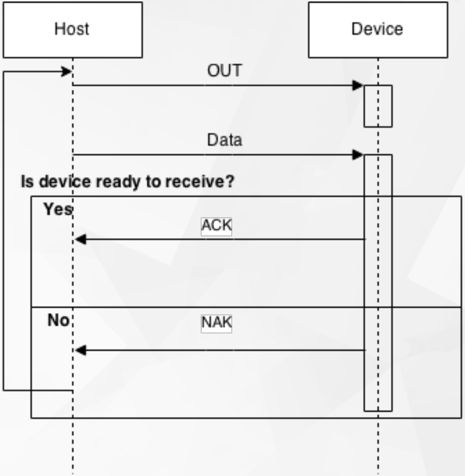
- **Host sends an IN token**
- **If the device has data:**
  - Device sends data
  - Host sends ACK
- **else**
  - Device sends NAK
  - Host will retry until timeout



USB transport (Link Layer)

OUT

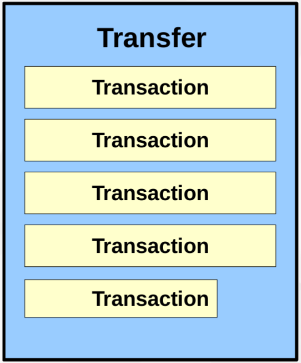
- **Host sends an OUT token**
- **Host sends the data (one packet)**
- **If device accepts data transfer:**
  - Device sends an ACK
- **else**
  - Device sends an NAK
  - Host will retry until success or timeout



∗ PING, NYET - bandwidth savers

USB transfer vs transaction

- **Transaction**
  - Delivery of data to endpoint
  - Limited by wMaxPacketSize
- **Transfer**
  - One or more transactions
  - May be large or small
  - Completion conditions



Source: [10]

USB Request Block

- **Kernel provides hardware independent API for drivers**
- **URB is a kind of envelope for data**
- **This API is asynchronous**
  - *usb_alloc_urb()*
  - *usb_free_urb()*
  - *usb_submit_urb()*
  - *usb_unlink_urb()*
  - *usb_kill_urb()*

```
struct urb {
    struct list_head urb_list;

    struct usb_device *dev;
    unsigned int pipe;

    int status;
    unsigned int transfer_flags;
    void *transfer_buffer;
    u32 transfer_buffer_length;
    u32 actual_length;

    unsigned char *setup_packet;

    void *context;
    usb_complete_t complete;
};
```

USB sniffing & modification

SAMSUNG

USBMon

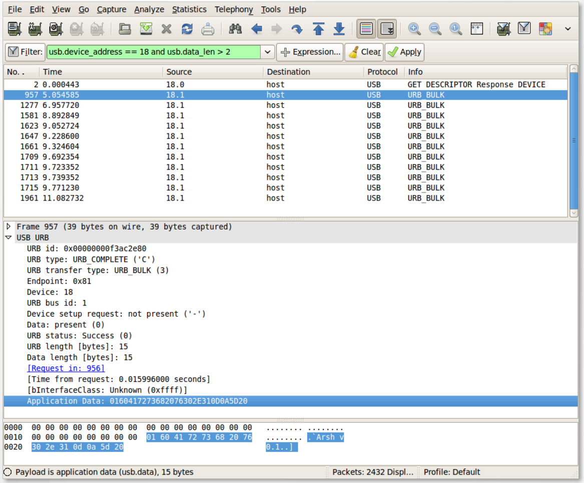
- **Kind of logger for URB related events:**
  - submit()
  - complete()
  - submit_error()
- **So it's not going to show you link layer USB tokens!**
- **Text interface**
- **Binary Interface**
- **One instance for each USB bus**

Data validity

- **Data in URB buffer may is not always valid**
- **Validity depends on transfer results**
- **And on endpoint direction:**

	IN	**OUT**
submit()	NO	YES
complete()	YES	NO

Good old friend Wireshark - DEMO


File Edit View Go Capture Analyze Statistics Telephony Tools Help
Expression... Clear Apply

Filter: usb.device_address == 18 and usb.data_len > 2

```
No. .  Time         Source      Destination  Protocol  Info
    2 0.000443      18.0        host         USB       GET DESCRIPTOR Response DEVICE
  957 5.054585      18.1        host         USB       URB_BULK
 1277 6.957720      18.1        host         USB       URB_BULK
 1581 8.892849      18.1        host         USB       URB_BULK
 1623 9.052724      18.1        host         USB       URB_BULK
 1647 9.228600      18.1        host         USB       URB_BULK
 1661 9.324604      18.1        host         USB       URB_BULK
 1709 9.692354      18.1        host         USB       URB_BULK
 1711 9.723352      18.1        host         USB       URB_BULK
 1713 9.739352      18.1        host         USB       URB_BULK
 1715 9.771230      18.1        host         USB       URB_BULK
 1961 11.082732     18.1        host         USB       URB_BULK
```

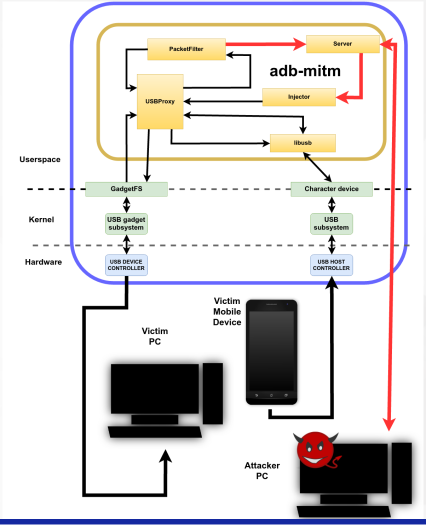
```
▷ Frame 957 (39 bytes on wire, 39 bytes captured)
▽ USB URB
    URB id: 0x00000000f3ac2e80
    URB type: URB_COMPLETE ('C')
    URB transfer type: URB_BULK (3)
    Endpoint: 0x81
    Device: 18
    URB bus id: 1
    Device setup request: not present ('-')
    Data: present (0)
    URB status: Success (0)
    URB length [bytes]: 15
    Data length [bytes]: 15
    [Request in: 956]
    [Time from request: 0.015996000 seconds]
    [bInterfaceClass: Unknown (0xffff)]
    Application Data: 0160417273682076302E310D0A5D20
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0010  00 00 00 00 00 00 00 00  01 60 41 72 73 68 20 76   ........ .`Arsh v
0020  30 2e 31 0d 0a 5d 20                               0.1..]
```

○ Payload is application data (usb.data), 15 bytes | Packets: 2432 Displ... | Profile: Default

USBProxy[1]

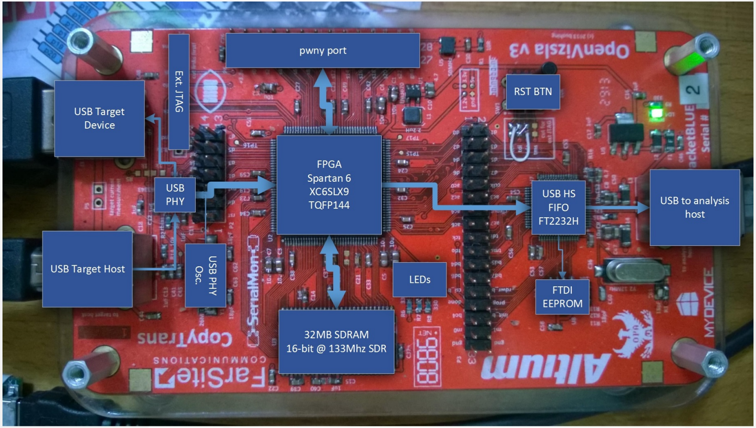
- Framework for USB MITM
- In theory, works on any SBC with UDC and HCD
- In practice, works only on BBB with custom kernel image
- Uses libusb & GadgetFS
- Can intercept only one device
- Still needs some love...



Just a logic analyzer...

- **For Full or Low Speed devices definitely yes!**
- **High speed bus signaling is 480 Mbit/s**
- **So you would need to probe with 1GHz frequency**

OpenVizsla[8]



Source: [9]

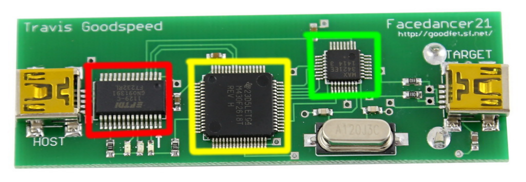
OpenVizsla host tools - DEMO

- **ovctl.py**
- **ViewSB**
- **Wireshark!**

USB security testing

FaceDancer[3]

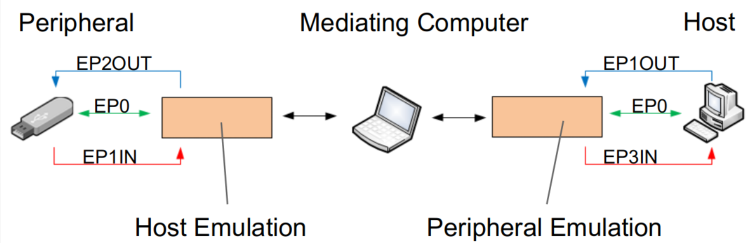
- **Hardware**



Source: [2]

- **Software**
    - Python framework for emulating USB devices

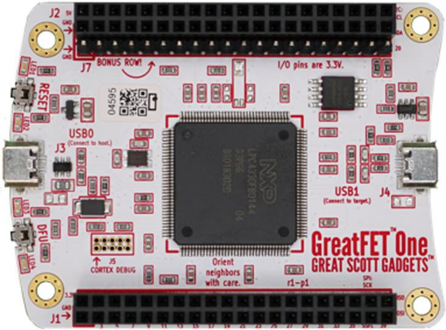
BTW 2x Facedancer MITM



Source: [12]

GreatFET[4]

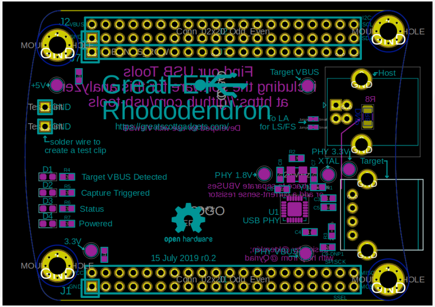
- Hacking platform
- Initially created for Radio Hacking
- NXP LPC4330 MCU
- 1x HS USB
- 1x FS USB
- Compatible with Facedancer software



Source: [6]

GreatFET Rhodadendron[5]

- **GreatFET neighbor with USB3343 for sniffing**
- **Unfortunately GreatFET does not have any external RAM memory...**



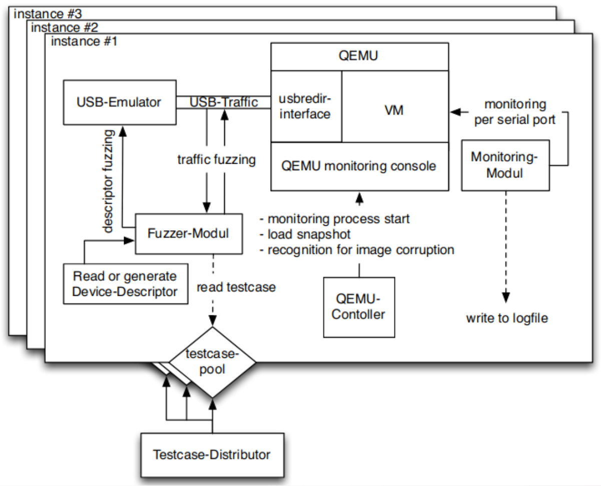
Source: [5]

umap2[13]

- **umap2scan**
- **umap2emulate**
- **umap2stages**
- **umap2fuzz (kitty-based)**
- **Supported backends:**
  - Facedancer (and GreatFET)
  - Raspdancer
  - GadgetFS (partially supported)

vUSBf[11] & friends

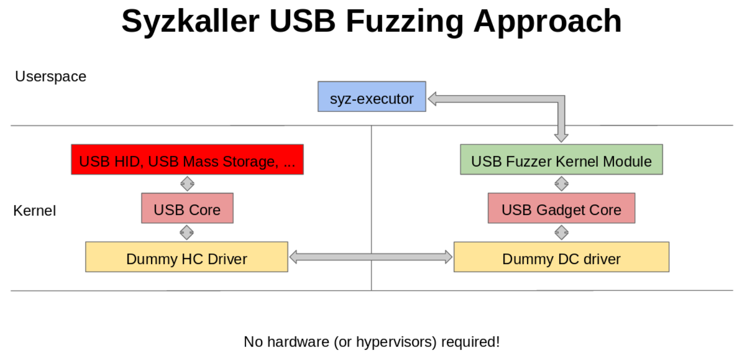
- **VM-based fuzzing**
- **Hypervisor specific**
- **Limited by hypervisor implementation**
- **Scapy-based fuzzing**



Source: [11]

syzcaller-based architecture[7]

- DummyHCD-based
- GadgetFS/ Custom module
- Use syzcaller to generate USB traffic
- Require "description" files



Syzkaller USB Fuzzing Approach

Userspace

syz-executor

Kernel

USB HID, USB Mass Storage, ...

USB Core

Dummy HC Driver

USB Fuzzer Kernel Module

USB Gadget Core

Dummy DC driver

No hardware (or hypervisors) required!

Source: [7]

Summary

Summary

- **You don't need to spend a lot money to sniff USB traffic**
- **There is a number of Open Source and Open Hardware USB tools**
- **There is no perfect architecture for testing USB security**

Q & A

Thank you!

Krzysztof Opasiak

Samsung R&D Institute Poland

+48 605 125 174
k.opasiak@samsung.com

References I

[1] *A proxy for USB devices, libUSB and GadgetFS*. Oct. 2017. URL: https://github.com/dominicgs/USBProxy.

[2] *FaceDancer21 in Hackerware House*. URL: https://hackerwarehouse.com/product/facedancer21/.

[3] *FaceDancer21 (USB Emulator/USB Fuzzer)*. URL: https://int3.cc/products/facedancer21.

[4] *GreatFET github repo*. URL: https://github.com/greatscottgadgets/greatfet.

[5] *GreatFET Rhododendron*. URL: https://github.com/ktemkin/greatfet-rhododendron.

References II

[6] *GreatScottGadgets GreatFET*. URL: https://greatscottgadgets.com/greatfet/.

[7] Andrey Konovalov. "Coverage-Guided USB Fuzzing with Syzkaller". In: *OffensiveCon*. Berlin, DE, 2019. URL: https://www.youtube.com/watch?v=1MD5JV6LfxA.

[8] *OpenVizsla USB Analyzer*. URL: https://github.com/openvizsla/ov_ftdi.

[9] *OpenVizsla USB Analyzer - fail0ver article*. URL: https://fail0verflow.com/blog/2014/ov3-hardware/.

[10] Alan Ott. "USB and the Real World". In: *Embedded Linux Conference*. San Jose, CA, USA, 2014. URL: http://elinux.org/images/6/66/Elc\%5F2014\%5Fusb.pdf.

References III

[11] Ralf Spenneberg Sergej Schumilo and Hendrik Schwartke. “Don’t trust your USB! How to find bugs in USB device drivers”. In: *Black Hat Europe*. Amsterdam, NL, 2014. URL: https://www.blackhat.com/docs/eu-14/materials/eu-14-Schumilo-Dont-Trust-Your-USB-How-To-Find-Bugs-In-USB-Device-Drivers-wp.pdf.

[12] Rijnard van Tonder and Herman Engelbrecht. “Lowering the USB Fuzzing Barrier by Transparent Two-Way Emulation”. In: *8th USENIX Workshop on Offensive Technologies (WOOT 14)*. San Diego, CA: USENIX Association, 2014. URL: https://www.usenix.org/conference/woot14/workshop-program/presentation/van-tonder.

[13] *umap2: The second revision of NCC Group’s python based USB host security assessment tool*. URL: https://github.com/nccgroup/umap2.