Emulation de périphérique USB-ETH pour l'audit IoT/Automotive

Philippe AZALBERT - *@Phil_BARR3TT*

Quarkslab

Étapes typique d’une évaluation d’IoT

- Extraction **mémoire**
- Recherche d’accès de **debug**
- Analyse des communications vers le **backend**
- …

Illustration : [lien]

Exemple #1 : accès aux logs

```
#
# there are multiple types of loggers with different characteristics
#
# the Logger which is a usb device using CDC-EEM
# that comes up on 192.168.186.1 with an ftp server
findLogger () {
    monitorLoggerName=""
    # find the usb device vendor and product id
    USBID=$(lsusbv)
    if [ "${USBID}" == "f00d:b33f" ]; then
      # logger board
      monitorLoggerName="common-logger"
      loggerInterface="usb0"
      LOGGER_IP=192.168.186.1
      DAEMONOPTS=" ${LOGGER_IP}"
    fi
}
```

Exemple #1 : accès aux logs

```
#
# there are multiple types of loggers with different characteristics
#
# the Logger which is a usb device using CDC-EEM
# that comes up on 192.168.186.1 with an ftp server
findLogger () {
    monitorLoggerName=""
    # find the usb device vendor and product id
    USBID=$(lsusbv)
    if [ "${USBID}" == "f00d:b33f" ]; then
      # logger board
      monitorLoggerName="common-logger"
      loggerInterface="usb0"
      LOGGER_IP=192.168.186.1
      DAEMONOPTS=" ${LOGGER_IP}"
    fi
}
```

Exemple #2 : ouverture de ports

```
manage_ports() {
    local ifaces=$@

    for iface in ${ifaces}; do
        if [[ "${iface}" == "usb0" ]]; then
            _add_rule -A ${INPUT_CHAIN_BLOCK_NEW} "-p tcp -i ${TCU_NETWORK_IFACE} --dport 1337 -j ACCEPT"
            _add_rule -A ${INPUT_CHAIN_BLOCK_NEW} "-p udp -i ${TCU_NETWORK_IFACE} --dport 1337 -j ACCEPT"
        fi
    done
}
```

Exemple #2 : ouverture de ports

```
manage_ports() {
    local ifaces=$@

    for iface in ${ifaces}; do
        if [[ "${iface}" == "usb0" ]]; then
            _add_rule -A ${INPUT_CHAIN_BLOCK_NEW} "-p tcp -i ${TCU_NETWORK_IFACE} --dport 1337 -j ACCEPT"
            _add_rule -A ${INPUT_CHAIN_BLOCK_NEW} "-p udp -i ${TCU_NETWORK_IFACE} --dport 1337 -j ACCEPT"
        fi
    done
}
```

Point commun entre ces deux IoTs

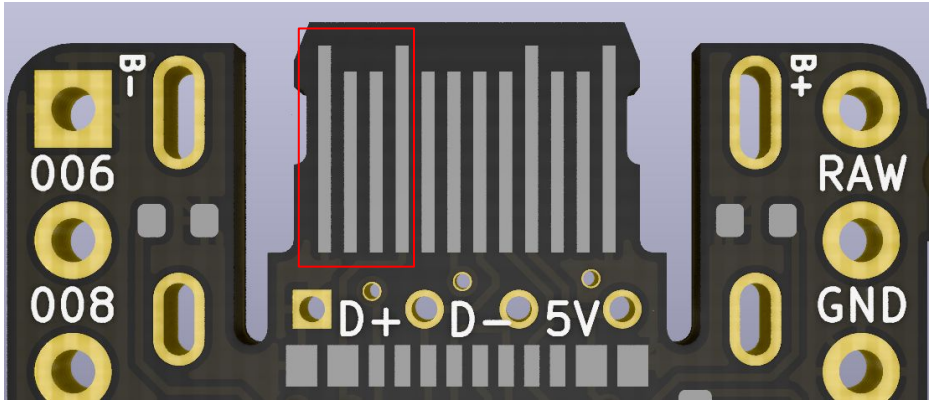


Illustration : [lien]

Un équipement **Ethernet-over-USB** spécifique débloque l'accès aux logs ou modifie les iptables

L’automobile, un IoT de tout les jours

- Réglementation **UE 2015/758 “E-Call”** : **connectivité data obligatoire** dans tout nouveau modèle de véhicule routier

- Le calculateur **télématique (TCU)** est en charge de cette connectivité

- Outre les appels d’urgence, il sert également pour la connectivité d’autres ECUs
  - Console **info-divertissement (IVI)**
  - Remontée de logs de la **Gateway**
  - **Mises à jour**
  - …

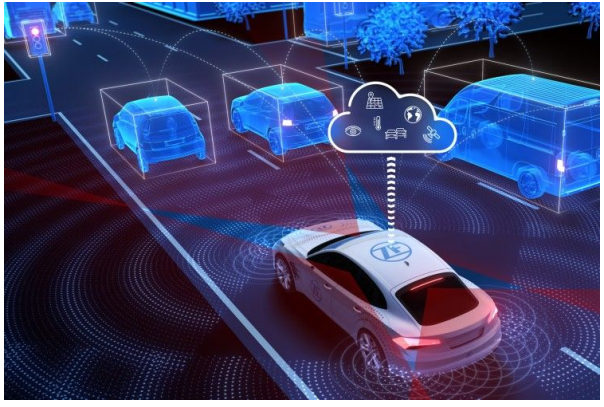
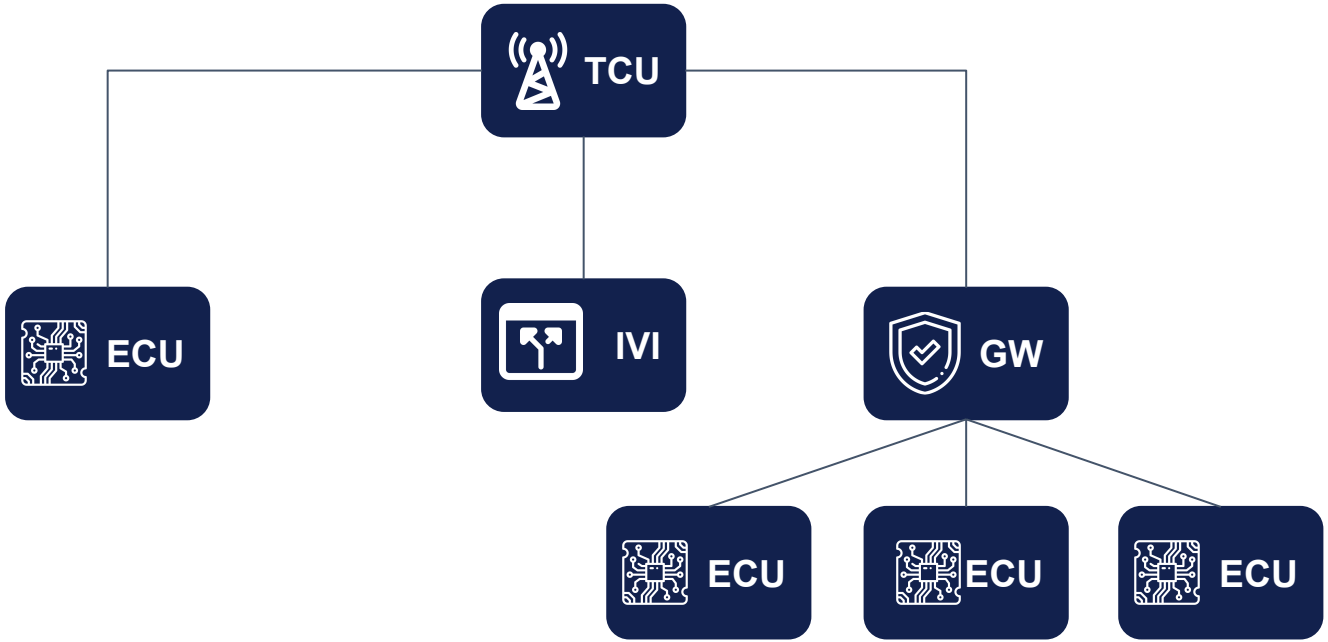


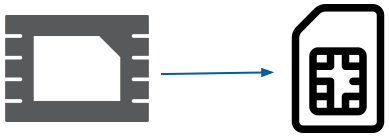
Illustration : [lien]

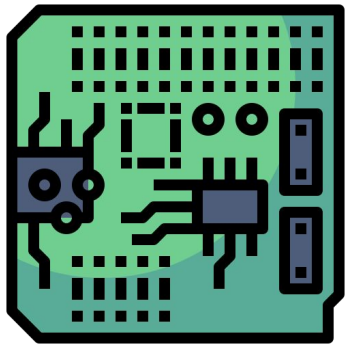
Topologie d’un véhicule connecté



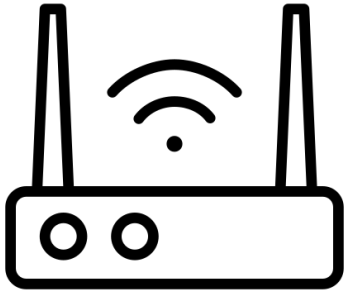
Illustrations : [lien]

Audit des communications d’un modem 4G/5G





TCU
eSIM -> SIM

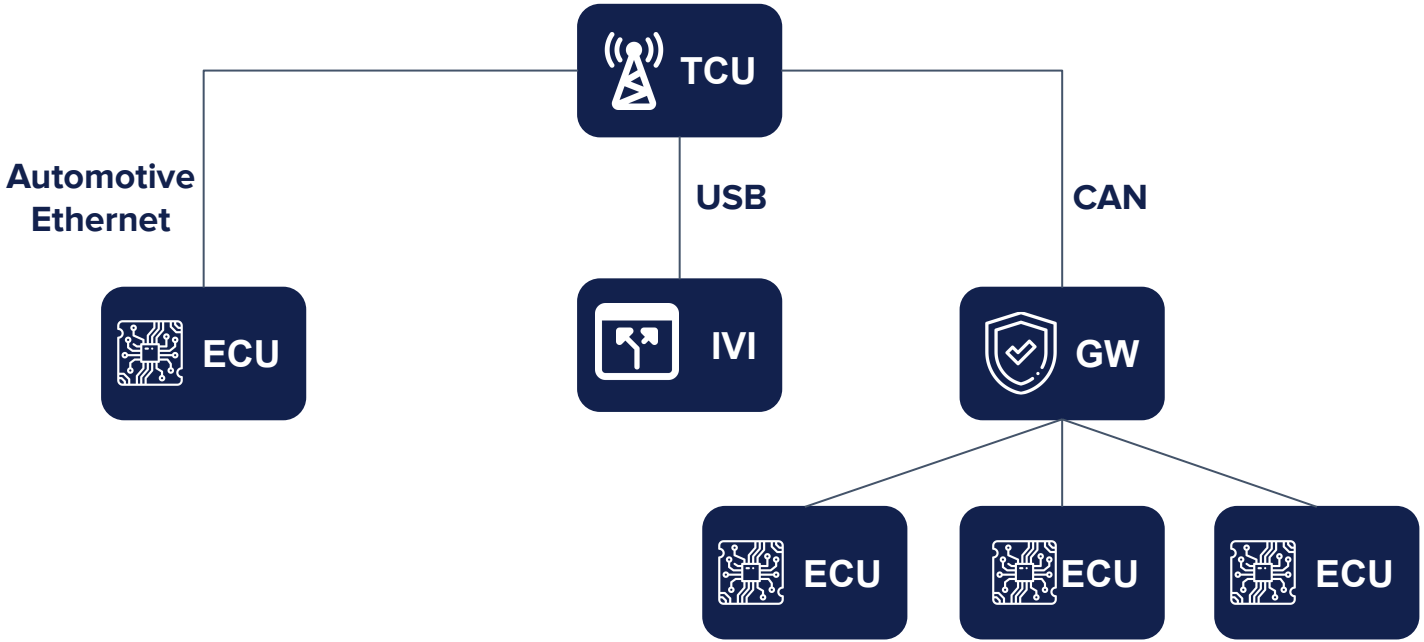


SDR
Réseau de test 4G / 5G

Cage Faraday

Illustrations : [lien]

Topologie d’un véhicule connecté



Et pourquoi pas un adaptateur ASIX ?

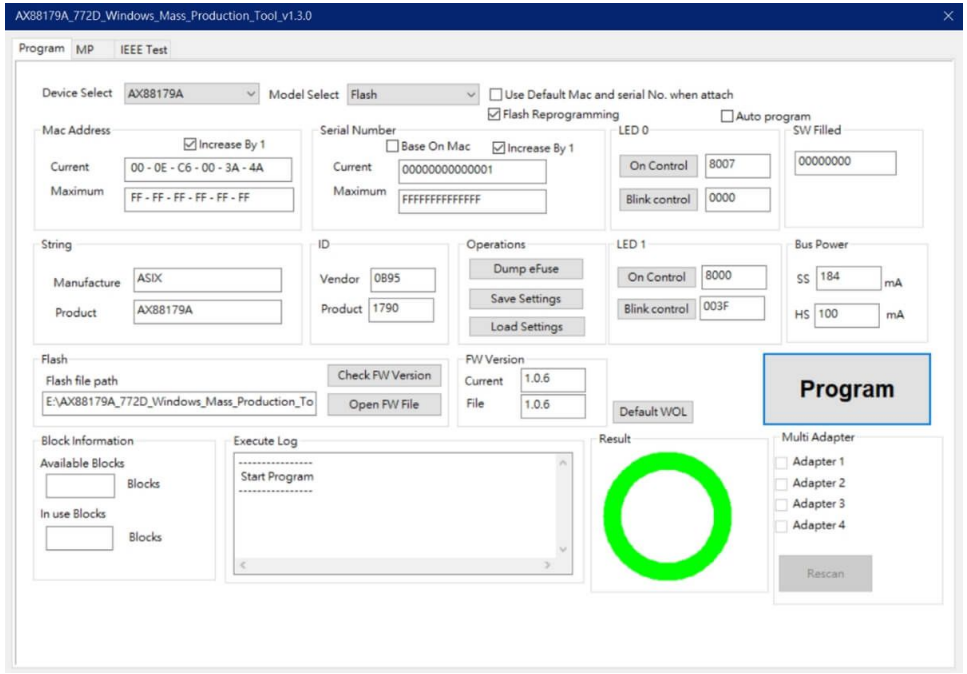
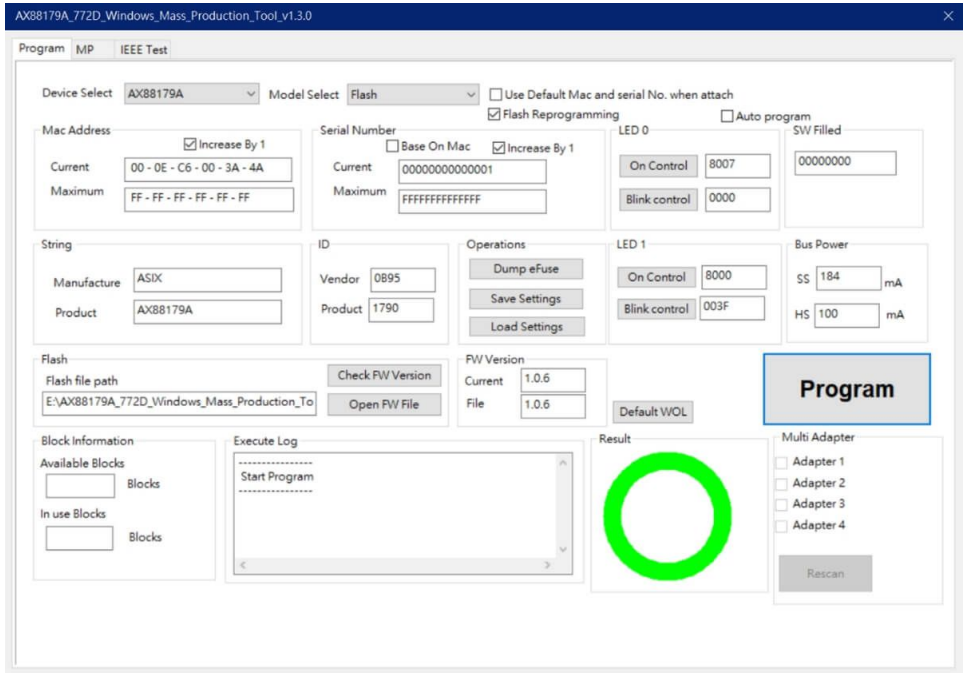


Illustration : [lien]

Et pourquoi pas un adaptateur ASIX ?



- ✓ VID / PID
- ✓ Manufacturer String
- ✓ Product String
- ✓ Numéro de série
- ✓ Adresse MAC
- ✓ Alimentation du bus

Illustration : [lien]

Et pourquoi pas un adaptateur ASIX ?

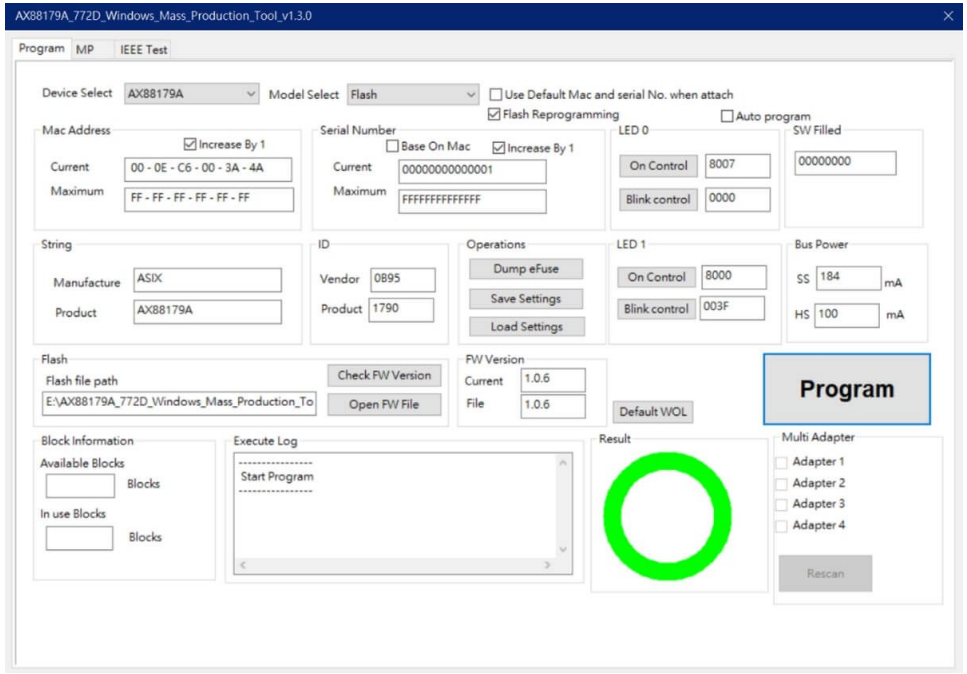


Illustration : [lien]

- ✓ VID / PID
- ✓ Manufacturer String
- ✓ Product String
- ✓ Numéro de série
- ✓ Adresse MAC
- ✓ Alimentation du bus

- ✘ Descripteurs de configuration
- ✘ Descripteurs d’interface
- ✘ Descripteurs d’endpoint
- ✘ eFuse pouvant limiter la programmation

Le protocole USB

- Architecture **hôte-périphérique**
- 4 types de **transferts**
  - Control
  - Bulk
  - Interrupt
  - Isochronous
- Transmission des données par **transactions**
  - Token
  - Data
  - Handshake
- **Classes** (Mass Storage, CDC, HID..)
- Buffers de données unidirectionnels : **endpoints**

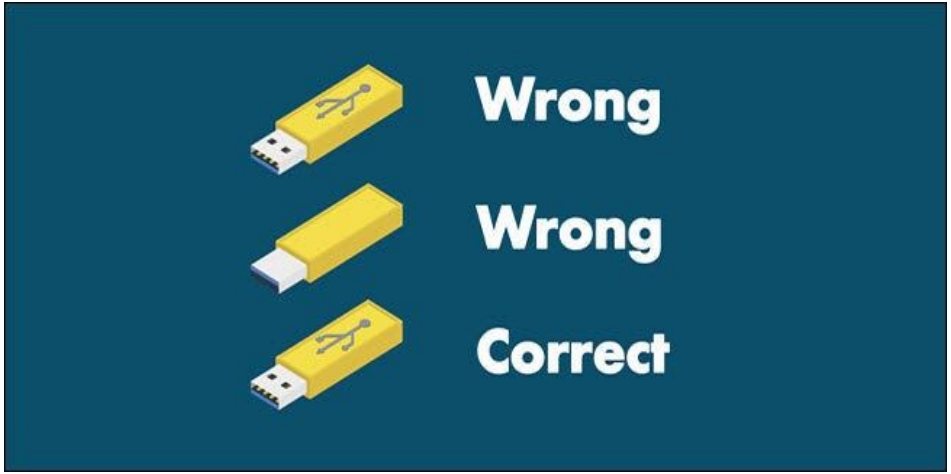


Illustration : [lien]

Le protocole USB - les descripteurs

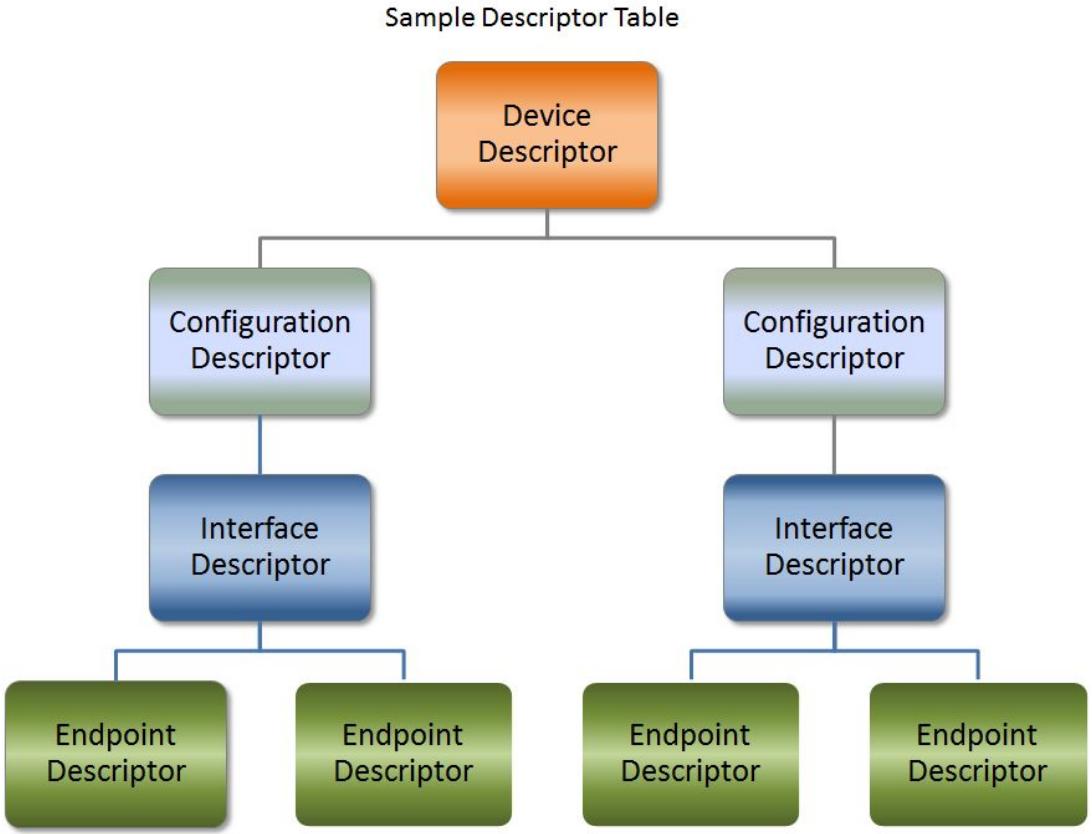


Illustration : [lien]

Exemple de descripteurs

```
Device Descriptor:
  [...]
  bDeviceClass            2 Communications
  bDeviceSubClass         2 Abstract (modem)
  bDeviceProtocol         1 AT-commands (v.25ter)
  bMaxPacketSize0         8
  idVendor           0xf00d
  idProduct          0xb33f
  bcdDevice            0.00
  iManufacturer           1 TelematicCorp
  iProduct                2 TCU_2000
  iSerial                 4 123456789
  bNumConfigurations      1
  Configuration Descriptor:
    [...]
    bNumInterfaces          1
    bConfigurationValue     1
    [...]
    Interface Descriptor:
      [...]
      bNumEndpoints           2
      bInterfaceClass         2 Communications
      bInterfaceSubClass     12 Ethernet Emulation
      bInterfaceProtocol      7 Ethernet Emulation (EEM)
      iInterface              3 USB CDC EEM
      Endpoint Descriptor:
        bLength                 7
        bDescriptorType         5
        bEndpointAddress     0x83  EP 3 IN
        bmAttributes            2
          Transfer Type            Bulk
          Synch Type               None
          Usage Type               Data
```

Facedancer 2.9

- Combinaison de **platines électronique** et d'une **librairie Python** pour émuler un périphérique USB

- Exemple de cartes compatibles
  - Facedancer21 [https://goodfet.sourceforge.net/hardware/facedancer21/]
  - GreatFET [https://greatscottgadgets.com/greatfet/]
  - Luna/Cynthion [https://greatscottgadgets.com/cynthion/]

- La version 2.9 de la libraire Facedancer permet un mode "**Proxy**" avec un support de **MitM**
  https://github.com/greatscottgadgets/Facedancer

Illustration : [lien]

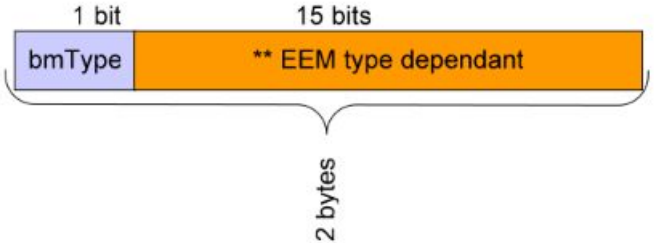
Facedancer - émulation d'un périphérique en quelques lignes

```python
from facedancer              import main
from facedancer.future       import *


@use_inner_classes_automatically
class HackRF(USBDevice):
    """ Device that emulates a HackRF enough to appear in ``hackrf_info``.
    """

    # Show up as a HackRF.
    product_string      : str = "HackRF One (Emulated)"
    manufacturer_string : str = "Great Scott Gadgets"
    vendor_id           : int = 0x1d6b
    product_id          : int = 0x0002

    # Most hosts won't accept a device unless it has a configuration  and an interface.
    class DefaultConfiguration(USBConfiguration):
        class DefaultInterface(USBInterface):
            pass

    # Vendor requests.
    @vendor_request_handler(number=14, direction=USBDirection.IN)
    @to_device
    def handle_control_request_14(self, request):
        # Theoretically, this is the point where you'd experiment
        # with providing one-byte responses and see what `hackrf_info` does.
        request.reply([2])
```

Illustration : [lien]

Protocole CDC-EEM

- **EEM** : **E**mulated **E**thernet **M**odule
- Utilise des transferts de type **Bulk**
- Se compose d'une **entête** (header - 2 octets - petit boutisme) et de **données** (payload)
- Deux types de paquets **EEM**
  - **Commande** (bmType = 1)
  - **Données** (bmType = 0)
- **CRC32** des trames Ethernet, fixé à 0xdeadbeef si bmCRC = 0



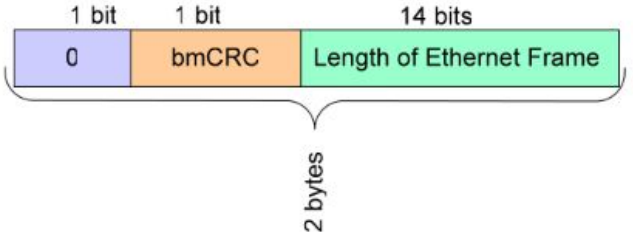
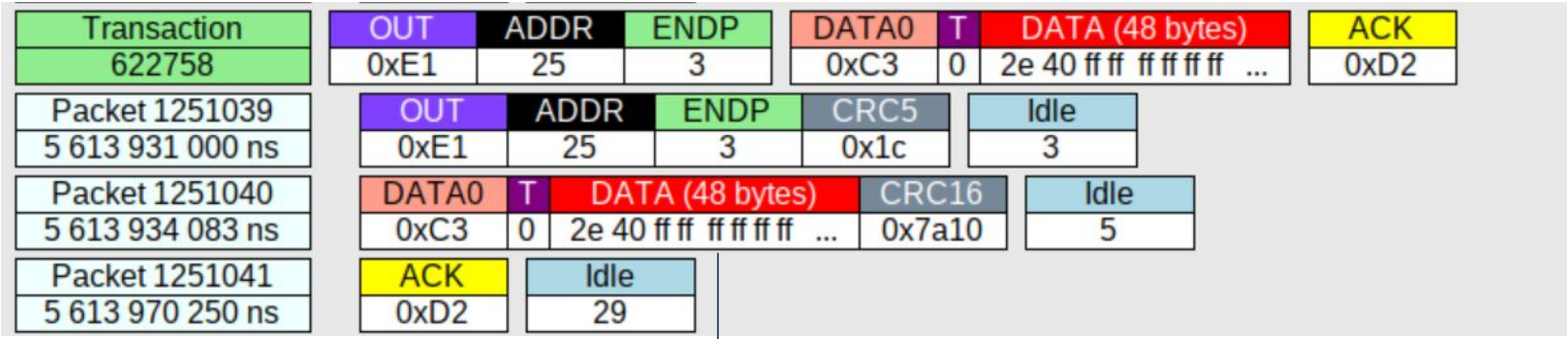


Illustration : [lien]

Protocole CDC-EEM - exemple de trame de données



Transaction	OUT	ADDR	ENDP	DATA0	T	DATA (48 bytes)	ACK
622758	0xE1	25	3	0xC3	0	2e 40 ff ff ff ff ff ff ...	0xD2

Packet 1251039	OUT	ADDR	ENDP	CRC5	Idle
5 613 931 000 ns	0xE1	25	3	0x1c	3

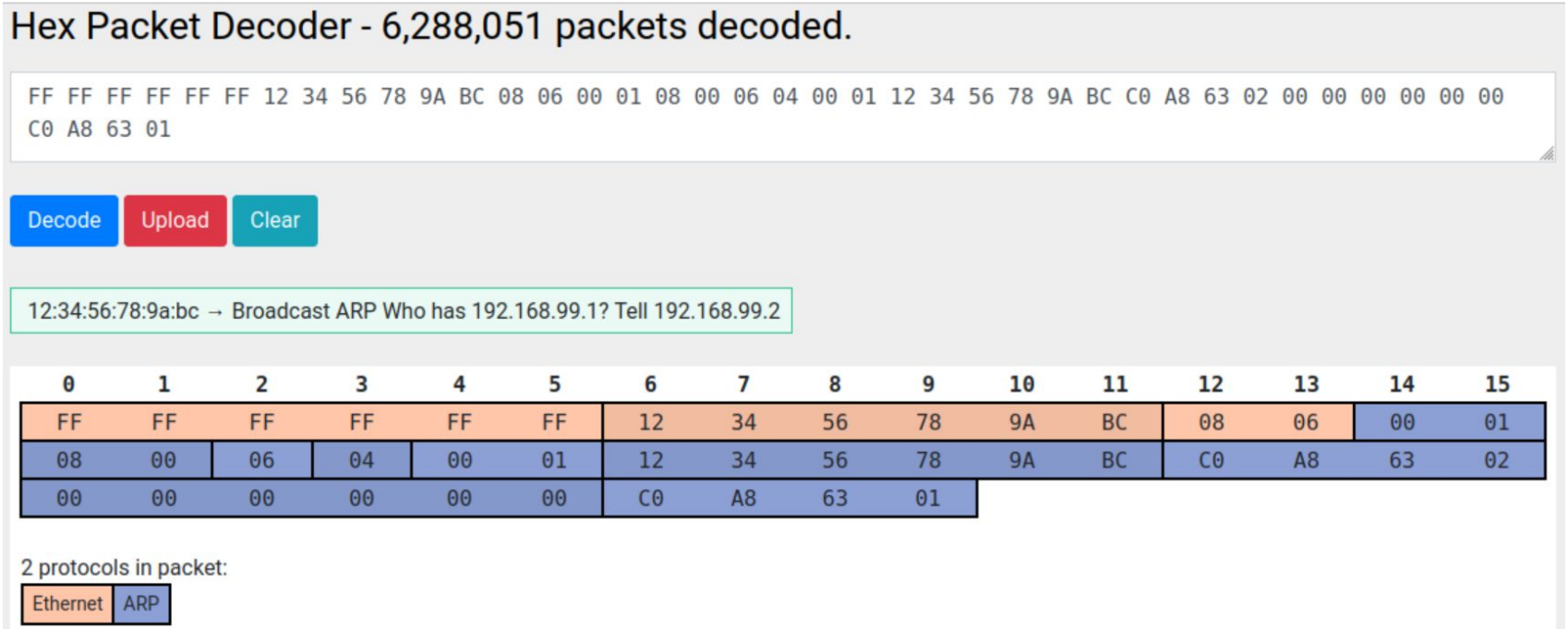
Packet 1251040	DATA0	T	DATA (48 bytes)	CRC16	Idle
5 613 934 083 ns	0xC3	0	2e 40 ff ff ff ff ff ff ...	0x7a10	5

Packet 1251041	ACK	Idle
5 613 970 250 ns	0xD2	29

```
2E 40 FF FF FF FF FF FF
12 34 56 78 9A BC 08 06
00 01 08 00 06 04 00 01
12 34 56 78 9A BC C0 A8
63 02 00 00 00 00 00 00
C0 A8 63 01 05 FA 98 8B
```

- **Header** : 0x402E
  - Type : Data
  - CRC : calculé
  - Payload : 46 octets
- **CRC** : 0x8B98FA05

Décodage de la trame Ethernet



Scapy : manipulation de paquets réseaux

https://scapy.net/

La minute Scapy

- Chaque protocole a sa **classe**
- Un paquet se compose de classes **encapsulées**
- **Exemple 1 :** forgeons une requête PING

```
>>> paquet = IP(dst='8.8.8.8')/UDP(dport=53)/ICMP()
>>> hexdump(paquet)
0000  45 00 00 24 00 01 00 00 40 11 A8 DA C0 A8 01 36  E..$....@......6
0010  08 08 08 08 00 35 00 35 00 10 2D 76 08 00 F7 FF  .....5.5..-v....
0020  00 00 00 00                                      ....
```

- **Exemple 2 :** vérifier si un protocole est présent

```
>>> ICMP in paquet
True
```

Décoder les paquets réseaux reçus avec Scapy

```
FF FF FF FF FF FF 12 34 56 78 9A BC 08 06 00 01 08 00 06 04 00 01 12 34 56 78 9A BC C0 A8 63 02 00 00 00 00 00 00
C0 A8 63 01
```

Decode | Upload | Clear

12:34:56:78:9a:bc → Broadcast ARP Who has 192.168.99.1? Tell 192.168.99.2

- Classe **Ether**

```
>>> print(data)
b'\xff\xff\xff\xff\xff\xff\x124Vx\x9a\xbc\x08\x06\x00\x01\x08\x00\x06\x04\x00\x01\x12
4Vx\x9a\xbc\xc0\xa8c\x02\x00\x00\x00\x00\x00\x00\xc0\xa8c\x01'
>>>
>>> Ether(data)
<Ether  dst=ff:ff:ff:ff:ff:ff src=12:34:56:78:9a:bc type=ARP |<ARP  hwtype=Ethernet (
10Mb) ptype=IPv4 hwlen=6 plen=4 op=who-has hwsrc=12:34:56:78:9a:bc psrc=192.168.99.2
hwdst=00:00:00:00:00:00 pdst=192.168.99.1 |>>
>>>
```

Fonctionnalités intégrées

- **ARP / ICMP**
  Transmission de la réponse si destination = adaptateur émulé

- **DHCP**
  Génération d'un bail statique

- **DNS**
  Envoi d'une réponse usurpée sur la base d'une liste personnalisable de noms de domaine

Illustrations : [lien]

Arguments supportés

- **--vid / --pid**
  Valeurs souhaitées du Vendor ID et Product ID

- **-m / -p / -s**
  Modification des Manufacturer String, Product String, Serial number

- **-- dns**
  Chemin d'accès vers un fichier contenant les couples 'nom de domaine / ip' cible pour intercepter les requêtes DNS

- **--ip / --netmask / --gw / --mac**
  Paramètres réseaux de l'adaptateur

Connexion au réseau

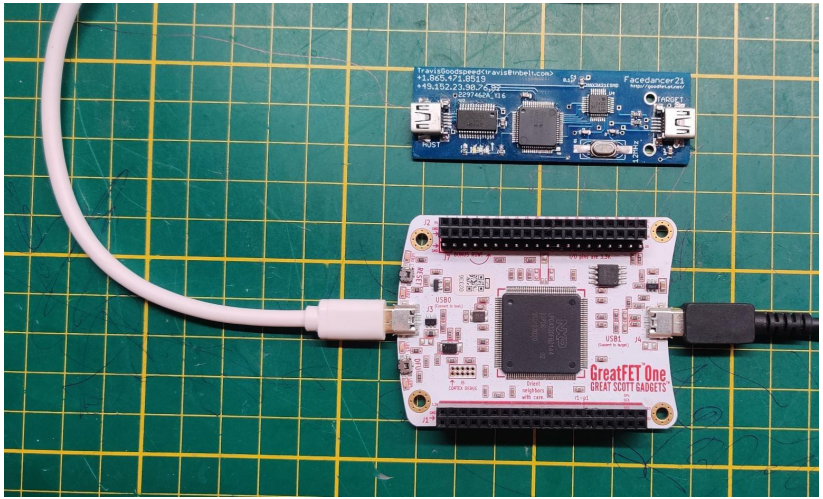
- Scapy fonctionne dans l'**userland**
- Lors de l'envoi de paquet **SYN-ACK**, le kernel transmet un paquet **RST**
- Solution : utilisation d'une interface **TUN**

```
sudo ip tuntap add name eem0 mode tun
sudo ip addr add 192.168.99.1 peer 192.168.99.2 dev eem0
sudo ip link set eem0 up
```

- Règles iptables **POSTROUTING/FORWARD** pour disposer d'une connectivité réseau

```
sudo iptables -t nat -A POSTROUTING -s 192.168.99.2 -j MASQUERADE
sudo iptables -A FORWARD -i eem0 -s 192.168.99.2 -j ACCEPT
sudo iptables -A FORWARD -o eem0 -d 192.168.99.2 -j ACCEPT
```

Démo !



- Règles iptables **POSTROUTING/FORWARD** pour disposer d'une connectivité réseau

Evolutions ?

- Support IPV6
- Mode “**Man in the Middle**” ECU <-> TCU
- Scanner **VID/PID**

Illustration : [lien]

Merci !

Quarkslab

Thank you

Contact information:

Email: contact@quarkslab.com

Phone: +33 1 58 30 81 51

Website: www.quarkslab.com

@quarkslab

Quarkslab