ECOLE DES MINES DE NANCY

Rapport du projet Atom matrix m5-stack

-

Security of cyber and physical systems

Auteurs :

Linxue LAI

Bilal FOURKA

Tuteur :

Laurent Ciarletta

20 Novembre 2020

I. Objectifs et partie technique du projet

I.1. Objectifs du projet

L'objectif du projet est de construire un serveur sécurisé sur ESP32 Atom matrix qui reçoit des requêtes http de la part d'un client Atom matrix. Ce client, étant à distance, se connecte au serveur grâce à une connexion WIFI. Le client commande le serveur qui exécute des commandes. Ainsi, le serveur peut constituer une ressource qui peut exploiter physiquement et numériquement la machine avec laquelle il est connecté.

I.2. Partie technique

A. Serveur

Afin de générer le serveur et diffuser le WIFI on aura besoin de plusieurs bibliothèques.

```
// Import required libraries
#include "WiFi.h"
#include "ESPAsyncWebServer.h"
#include <Wire.h>
#include "M5Atom.h"
```

On a besoin de créer plusieurs variables globales pour la création de l'Access point et ainsi lancer le Async WebServer.

```
// Set your access point network credentials
const char* ssid = "depinfo";
const char* password = "123456789";
// Set web server port number to 80
unsigned int port_ = 80;
AsyncWebServer server(port_);
```

Ensuite, on doit lancer l'Access Point et cela à l'aide de la bibliothèque WIFI.h. Cela doit se faire absolument dans la fonction setup d'Arduino.

```
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}
// Print local IP address and start web server
Serial.println("");
Serial.println("WiFi connected.");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
```

Donc, il ne reste qu'à écrire un serveur qui va recevoir les requêtes GET vers /light et exécuter un programme qui peut exploiter la ressource.

```
server.on("/light", HTTP_GET, [](AsyncWebServerRequest *request){
    // your code here to execute *attack*
    request->send_P(200, "text/plain", "Hacked");
 });
```

B. Client

De même que le serveur, le client a besoin des variables afin de se connecter au wifi et avoir une adresse IP.

```
// Replace with your network credentials
const char* ssid     = "depinfo";
const char* password = "123456789";

// Set web server port number to 80
 WiFiServer server(80);
```

Dans la fonction setup de l'Arduino, on utilisera ces variables pour se connecter et afficher l'adresse IP attribuée par le gestionnaire des adresses IP.

```
Serial.print("Setting AP (Access Point)...");
// Remove the password parameter, if you want the AP (Access Point)
to be open
WiFi.softAP(ssid, password);
IPAddress IP = WiFi.softAPIP();
```

```
Serial.print("AP IP address: ");
Serial.println(IP);
server.begin();
```

Dans la fonction Loop de l'Arduino, on teste à chaque fois si le bouton de esp32 est pressé. Dans ce cas, on fixe l'adresse du serveur et on envoie une requête Http GET. Après le traitement de la requête le serveur répond et envoie un message qu'on affiche pour s'assurer que l'appareil esp32 serveur a bien exploité la ressource.

```
if (M5.Btn.wasPressed())
  {
    Serial.println("button pressed");
    HTTPClient http;
    String serverName = "http://192.168.4.2/light";
    http.begin(serverName.c_str());
    int httpResponseCode = http.GET();
    if (httpResponseCode>0) {
      Serial.print("HTTP Response code: ");
      Serial.println(httpResponseCode);
      String payload = http.getString();
      Serial.println(payload);
    }
    else {
      Serial.print("Error code: ");
      Serial.println(httpResponseCode);
    }
    // Free resources
    http.end();
  }
```

C. Exemple du fonctionnement.

Le client commence par diffuser le WIFI et prend l'adresse 192.168.4.1 pour lui-même. Ensuite le serveur se connecte au réseau et lance le serveur qui répond aux requêtes HTTP. Ainsi, lorsque le client fait un clic sur le bouton de l'esp32, Le script qui existe dans serveur s'exécute et envoie un message indiquant l'état d'avancement de l'exploitation (Dans notre cas on renvoie si la ressource est exploitée ou pas).

```
M5Atom initializing...Setting AP (Access Point)...AP IP address: 192.168.4.1
button pressed
HTTP Response code: 200
Hacked
```

Figure : Log du client

```
Connecting to depinfo
.
WiFi connected.
IP address:
192.168.4.2
```

Figure : Log du serveur

Lien GitHub du projet : https://github.com/fourkaBilal1/Atom-Matrix-ESP32

II. La sécurité de la connexion entre les Atom M5 Stack

Théorie de sécuriser la communication

Nous avons abordé ci-dessus la partie technique et montré comment configurer une communication HTTP entre deux cartes ESP32 pour échanger des données via Wi-Fi sans connexion Internet (routeur). Mais il y a des risques de sécurité de HTTP :

1. Divulgation de la confidentialité

Étant donné que HTTP lui-même est une transmission en texte clair, le contenu de la transmission entre l'utilisateur et le serveur peut être visualisé par l'intermédiaire. En d'autres termes, les informations sur les sites Web que nous recherchons, visitons et les pages sur lesquelles nous cliquons sur Internet peuvent être obtenues par les « intermédiaires ».

2. Détournement de page

Le risque de fuite de la vie privée est relativement caché et les utilisateurs ne peuvent fondamentalement pas le percevoir. Mais l'impact d'un autre type de piratage est très évident et un piratage de page très direct, c'est-à-dire une altération directe de la page de navigation de l'utilisateur.

Selon la classification des chemins de détournement, il y a des détournement

DNS, détournement de client et détournement de lien.

Tenons compte des risques ci-dessus, les considérations de sécurité doivent être prises en compte. Dans la documentation officielle [1] des produits ESP32, une méthode de sécurisation de la communication à distance est mentionnée. Il est recommandé d'utiliser le TLS standard pour sécuriser la communication.

Dans cette partie, nous parlons d'une méthode de protection de la communication d'ESP32 à l'aide du Framework Arduino, et de ses capacités concernant TLS.

Dans les langages ou Framework standards pour le développement Web ou de bureau, nous sommes habitués à simplement passer des appels HTTP en utilisant https: //, ou à inclure de petits extraits de code pour les écouteurs HTTPS avec une clé de serveur et un certificat. Donc, sécuriser une API au niveau du transport n'est pas si difficile. Cependant, sur les choses embarqués, la situation est différente :

1. Parfois, les microcontrôleurs n'offrent pas de cryptage en silicium.

2. Certains jeux de puces réseau / émetteur-récepteur incluent le cryptage, et peuvent rendre cette fonctionnalité accessible à l'extérieur.

3. Certaines cartes comportent un élément sécurisé - c'est-à-dire une puce de sécurité distincte pour les primitives cryptographiques, le cryptage et le stockage des clés, etc.

Ainsi, les bibliothèques de communication ci-dessus doivent utiliser des fonctionnalités spécifiques, et la majorité d'entre elles restent spécifiques à la carte / framework / OS. La demande de HTTPS est déterminée par la spécification de chiffrement et implique un ensemble de fonctions cryptographiques.

L'ESP32 comprend des fonctionnalités de cryptage sur puce, par exemple ce qui est nécessaire pour toutes les crypto-monnaies liées au WiFi: l'Arduino Core d'ESP32 comprend un port de Arm Mbed TLS (voir dans tools / sdk / include / mbedtls) et également OpenSSL.

Théorie de la mise en œuvre

Pour le serveur HTTPS, nous pouvons utiliser ESP32 comme serveur Web pour une API REST mais en utilisant HTTPS. La plupart du code du serveur HTTP pour Arduino fonctionne avec EthernetServer ou WiFiServer, mais il n'y a pas de TLS ou de lien vers le port mbedtls en dessous. Un exemple d'implémentation HTTPS peut être consulté sur ce lien [2]. Il utilise OpenSSL pour gérer TLS et propose une API de serveur HTTP pour créer des points de terminaison et des ressources, etc.

La couche TLS utilise des certificats CA de confiance pour valider que le point de terminaison / serveur distant est vraiment celui qu'il prétend être. L'API spécifiée accepte un certificat CA pour effectuer la validation du serveur.

Pour le client HTTPS, l'Arduino Core pour ESP32 contient des ports de Arm's Mbed TLS et openssl, mais ils sont enfouis plus profondément dans le SDK, de sorte qu'ils n'apparaissent généralement pas lors du codage pour Arduino Framework avec tous ces WiFiClients, digitalRead, loop et installer. De plus, les bibliothèques ESP32 incluent un WiFiClientSecure, qui s'interface avec mbedtls pour établir une connexion TLS sécurisée.

En surface, WiFiClientSecure est utilisé égal à WiFiClient, car il hérite de cette classe. De plus, il est nécessaire de définir le certificat CA de l'autorité de certification qui a signé le certificat du serveur. Ceci afin que l'appareil puisse être sûr de se connecter à un serveur qui a reçu un certificat de l'autorité de certification respective.

Conclusion

En guise de conclusion, ce projet était une opportunité de découvrir le monde de la cybersécurité et surtout son aspect physique. Cela nous a appris que ce n’est pas toujours une architecture réseau performante qui peut nous protéger mais aussi la capacité de traiter le risque de l'exploit physique de notre ressource.

Lien GitHub du projet : https://github.com/fourkaBilal1/Atom-Matrix-ESP32

Références :

[1] https://docs.espressif.com/projects/esp-jumpstart/en/latest/security.html

[2] github.com/fhessel/esp32_https_server

[3]https://www.thingforward.io/techblog/2018-07-18-https-on-the-esp32-server-and-client-side.html