Remerciements

Table des matières

Chapitre 1

Introduction

L'Internet des objets (IoT, en anglais) est un paradigme dont les premiers déploiements ont quelques années (voire plus, si l'on parle de réseau de capteurs). D'un point de vue sécurité, l'IoT a une surface d'attaque très importante, du fait du nombre de technologies, de protocoles, du type de déploiement et du nombre d'acteurs différents. Ce projet s'applique aux réseaux d'objets connectés longue porté du type LoRaWAN (Long Range Wide Area Network).

1.1 Contexte du projet

Dans le cadre de notre première année de Master 1 (Cyber-sécurité des Systèmes Embarqués) nous avons eu l'occasion de réaliser un projet d'une durée de 2 mois en parallèle de nos cours. Nous avons choisit le projet "Création d'un réseau LoRaWAN sécurisé" car il correspond à des technologies mis en œuvre pour l'IoT, qui nous intéresses.

1.2 LoRaWAN et la problématique de sécurité

Chapitre 2

Organisation

2.1 Méthode de travail

Pour gérer les modifications du projet nous avons utilisé un outil de versionning appelé *Github*, celui-ci nous permet de stocker les programmes ainsi que les différents documents nécessaire au projet. Pour nous organiser tout au long de la période du projet nous avons établi un diagramme de *GANTT*. Nous avons gardé à jour pendant toute la durée du projet. Pour avoir une gestion de projet plus précise (tâches à effectuer chaque semaines), nous avons utilisé l'onglet *Project* de notre *repository Github*, qui utilise un tableau de *Kanban*. Dans cet onglet nous indiquions pour chaque semaine, les différentes tâches à faire. Les tâches ont 3 états *À faire*, *En cours* et *Fini* nous déplacions les taches d'une colonne à l'autre en fonction de leurs statu, nous ajoutions aussi de nouvelles tâches au cour de la semaine si besoin.

Nous avons choisi une approche en spirale (méthode *Agile*) pour notre organisation vis à vis du développement. En effet, sur les conseils de notre encadrant, ce modèle nous permet de d'appliquer les différentes couches de sécurisation une à une et de revenir aux étapes précédentes si besoin pour modifier et compléter le dispositif, ou de passé à une autre après avoir effectué et validé des tests.

2.2 Logiciels utilisés

Pour gérer le versionning de notre projet nous avons utilisé *Git* car c'est un des outils que l'on nous a présenté lors de nos cours et dont nous avions déjà connaissance. Pour le développement du programme du nœud, nous avons utilisé *Visual Studio Code* qui est un éditeur de texte puissant et possédant plusieurs extensions facilitant la programmation dont *Pycom* qui nous a permis de développer un premier nœud. Nous avons aussi utilisé *STM32CubeIDE 1.0.2* pour le développement

sur carte car il nous a permis d'écrire un programme simple pour implémenter des fonctions de sécurité. Pour le debugage et le tests des fonctions de sécurité nous avons utilisé *St-Link* et *St-Link Utility* qui permettent de lire et de voir les *bytes* d'option d'un microcontrôleur STM32, de plus *ST-Link Utility* est disponibles uniquement sur Windows que nous avons du utilisé œuvre via une machine virtuelle supervisé par le logiciel *VirtualBox*. Pour effectuer le reverse engineering des binaires extrait de la mémoire par le biais de *St-Link* nous avons utilisé le logiciel *Ghidra* car c'est un logiciel que les Master 2 ont utilisé et c'est le seul logiciel de reverse engineering que nous connaissions.

Pour mettre en place la *Box LoRa* (passerelle, network server et application server) nous utilisons l'OS *ChirpStack* car il permet d'utiliser et de de configurer facilement ces 3 services du LoRaWAN de plus il peut être utilisé sur la Raspberry Pi qui nous sert de support pour cette partie du projet.

Pour finir avons utilisé comme OS Linux et plus précisement les distributions *Manjaro* qui est basé sur *Arch Linux* et *Xubuntu* qui est basé sur *Ubuntu* qui est lui même basé sur *Debian*

2.3 Répartition des tâches

Pour ce projet nous sommes deux étudiants, dans la première partie du projet (premier mois) nous avons tout deux étudié le LoRaWAN car nous n'avions pas de connaissances sur ce sujet auparavant. Pour valider nos connaissances acquises durant cette étape de recherche nous avons mis en place un démonstrateur à présenter à notre deuxième jalon .

A la reprise du projet en décembre, nous avons chacun travaillé sur une partie différente du projet. François à travaillé sur la sécurisation de la *Box LoRa* et Arthur sur la sécurisation du nœud.

La Box LoRa est appareil permettant d'utiliser plusieurs services, un service LoRaWAN mais peut aussi contenir des services de mail par exemple. L'objectif de cette partie est d'empêcher un utilisateur d'avoir accès aux ressources d'un autre. Par exemple, le service mail ne doit pas interférer avec le service LoRaWAN. Pour cela il a fallu apporté des modifications à l'OS *ChirpStack*.

Le nœud LoRaWAN à besoins de clés pour ce connecter au réseau, qu'il doit dans notre cas stocker dans sa mémoire. Si un attaquant parvint à récupérer ces clés, il pourra espionner le trafic LoRaWAN ou connecter sont propre nœud à la place du nôtre. Pour sécurisé ces clés il a fallu utiliser des solutions de sécurité déjà présentes dans le microcontrôleur ou utiliser un composant de sécurité pour stocker les clés.

Chapitre 3

Réalisation et Tests

Au cours de ce projet nous devions effectué 3 réalisations, un démonstrateur, un nœud sécurisé et une *Box LoRa* sécurisé.

3.1 Démonstrateur

Comme nous l'avons dis dans le chapitre précédant nous avons dans un premier temps réalisé un démonstrateur non sécurisé d'un réseau LoRaWAN. Le but était d'utiliser nos connaissances du LoRaWAN et de les démontrer lors d'une présentation. Pour créer ce démonstrateur nous avons utiliser l'OS *ChirpStack*[1] sur la *Raspberry Pi* équipé de son *shield* antenne *iC880A* pour le nœud nous avons utilisé la carte *Fipy*[2] avec sont *shield PySence*.

L'OS ChirpStack permet de mettre très facilement un réseau LoRaWAN en fonctionnement car il intègre déjà les différents services de passerelle, serveur réseau et serveur d'application. Ces différents service sont aussi facilement configurable grâce à une interface web.

La carte *Fipy*, est une carte de développement orienté pour l'utilisation de réseau sans fils comme WiFi, Bluetooth ou LoRaWAN. Cette carte ce programme en *Python* avec l'extension *Pymakr* pour les éditeur de text *Atom* et *Visual Studio Code*. Elle nous a permis de créer le noeud LoRaWAN facilement car la documentation donne des exemples pour ce type de programme.

Lors du développement de ce démonstrateur, nous n'avons pas eu de problèmes particulier autre que des problèmes de configuration de l'extension *Pymakr*.

En fonctionnement le nœud envoyait une chaine de caractère à la passerelle puis on lisait cette chaine caractère grâce au serveur d'application.

3.2 Nœud

Pour le rendu final du projet le nœud doit fonctionner sur une carte B-L072Z-LRWAN1[3] qui est une carte de découverte produite par STmicroelectronics orienté pour le développement de solutions basées sur des réseaux *LoRaWAN* ou *SigFox*. Cette carte est équipé d'une MCU *STM32L072CZ* qui est basé sur un Arm Cortex M0+.

Pour rappelle, dans l'architecture LoRaWAN, le rôle du nœud est de transmettre une information qui serra traité par l'application server. Dans le cadre de notre scénario le campus connecté, nous transmettons une donnée publique qui est la température. Nous nous sommes donc concentré, sur l'authentification du nœud LoRaWAN et l'intégrité de la donnée.

Pour cela nous devions éviter que notre nœud soit remplacé par un autre nœud lequel pourrait envoyer des informations erronées.

3.2.1 Surface d'attaque du nœud

Sur la figure 3.1 vous pouvez voir les différents composant autour de notre programme.

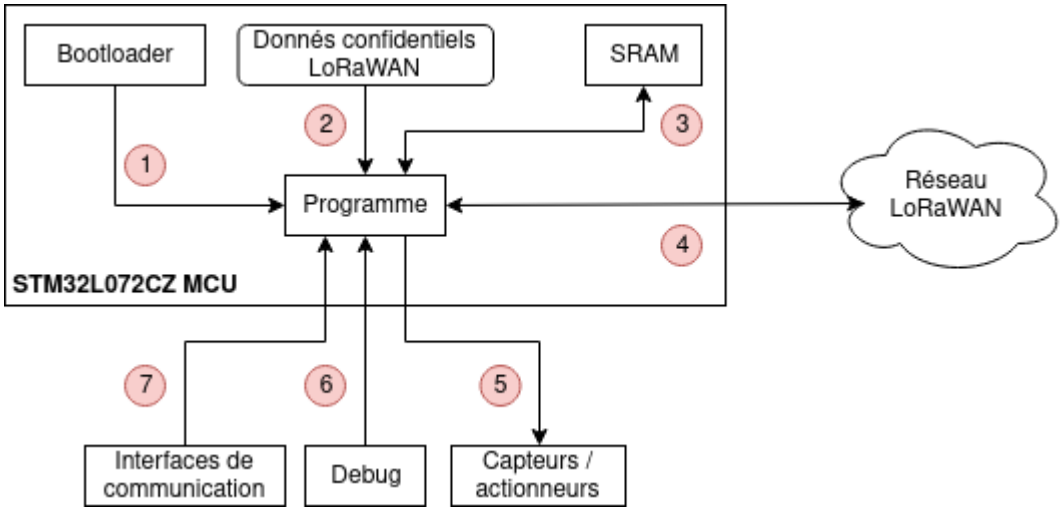

FIGURE 3.1 – Schéma présentant les différents éléments autour de la carte B-L072Z-LRWAN1

Il y a plusieurs vecteur d'attaque possible dans nôtre cas, chaque points de la figure 3.1 est un cible potentielle :

1. Changer le mode de démarrage pour utiliser un autre programme que celui contenu dans la mémoire Flash et avoir accès à tout les contenu du micro

contrôleur.

2. Accès à des parties du programme sensibles pour les copier ou les voler
3. Utilisation de faille dans la SRAM comme un buffer overflow pour voler des information de dans ou provoquer un dénie de service sur la carte.
4. L'observation des communications et/ou usurpation des appareils du réseau
5. Un attaquant pourrait remplacer le *shield* contenant les capteurs pour envoyer des fausses valeurs et mettre dans un état non déterminé.
6. Utilisation des ports de debugages pour avoir un accès complet au micro contrôleur.
7. Utilisation des Interfaces de communication pour avoirs accès au micro contrôleur

Pour chacune des attaques ci-dessus on peut trouver une ou plusieurs contremesures :

1. Autoriser qu'un seul mode de démarrage. Désactiver le démarrage depuis le port de débugage
2. Mettre la mémoire en mode execution only. Utilisé une MPU(Memory protection unit). Créer des zones mémoires sécurisé
3. Utilisé une MPU(Memory protection unit). Créer des zones mémoires sécurisé
4. Chiffrer et signer les communications.
5. Utilisation d'un système pouvant détecter une intrusion au niveau de la carte.
6. Désactiver les fonctionnalités de débugage
7. Rendre les interfaces de communications difficiles d'accès. Desactivé les interfaces si on ne les utilises pas.

Pour ce projet nous nous sommes concentré sur deux types d'attaques, la première **Une lecture de la mémoire (Dump mémoire)** et la deuxième **Accès à la mémoire par une partie du programme non autorisé**

Nous avons commencer par faire des recherche sur quel méthode de connexion était la plus intéressante à sécuriser par rapport à nôtres scénario.

Le protocole LoRaWAN permet à un nœud de ce connecter de 2 façons différentes, OTAA (Over The Air Activation), ABP (Activation by personalization).

3.2.2 Nœud LoRaWAN avec une carte B-L072Z-LRWAN1

Dans un premier temps nous avons trouvé un projet sur github qui permet d'utiliser cette facilement pour le LoRaWAN.

3.2.3 Tests Unitaire

3.2.4 Tests Intégration

3.2.5 Problèmes rencontré

3.2.6 Conclusion

3.3 Passerelle

3.3.1 Tests Unitaire

3.3.2 Tests Intégration

3.3.3 Problèmes rencontré

3.3.4 Conclusion

Chapitre 4

Conclusion

Table des figures

Annexe A

Protocole caractérisation flexiForce

Bibliographie

[1] Orne Brocaar. Chirpstack. https://www.chirpstack.io/.

[2] Pycom. Fipy. https://pycom.io/product/fipy/.

[3] STMicroelectronics. B-l072z-lrwan1. https://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-mpu-eval-tools/stm32-mcu-mpu-eval-tools/stm32-discovery-kits/b-l072z-lrwan1.html.