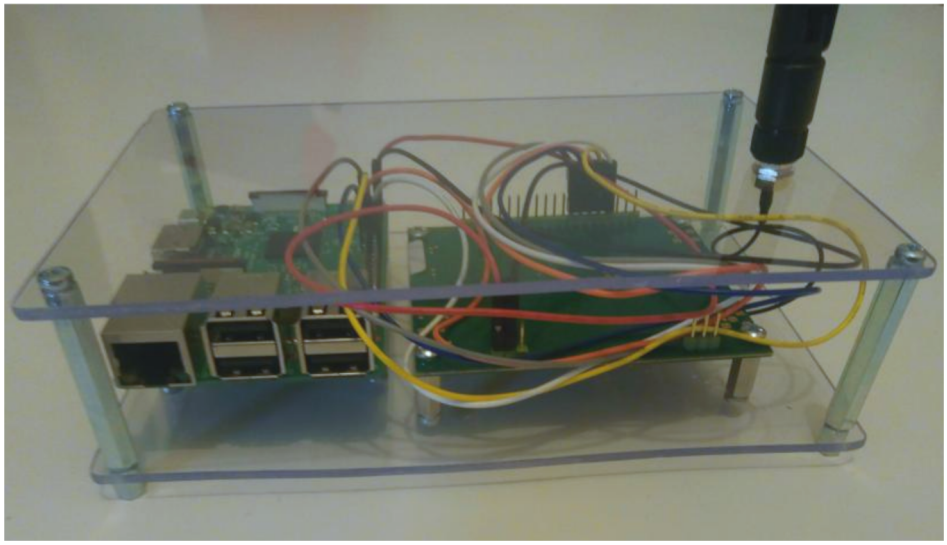
Faculté sciences & sciences de l'ingénieur

Université Bretagne Sud

Mise en place d'un réseau sécurisé LoRaWAN



François Sevaux Arthur Le Rest
M1 CSSE 2019 - 2020

Enseignant référent : Philippe Tanguy

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 portée du type LoRaWAN (Long Range Wide Area Network).

1.1 Contexte du projet

Dans le cadre de notre première année de Master 1 CSSE (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 choisi le projet "Création d'un réseau LoRaWAN sécurisé", car il correspond à des technologies mises en œuvre pour l'IoT, qui nous intéressent.

1.2 LoRaWAN et la problématique de sécurité

Dans ce contexte, il nous est demandé de mettre en place un réseau LoRaWAN sécurisé. Le premier et principal objectif est de créer un réseau LoRaWAN complet, mais simple et fonctionnel, dont les éléments de sécurité côté nœud et passerelle seront correctement mis en œuvre. Toute une démarche de tests unitaires devra être mise en place, pour tester chacune des parties séparément, puis l'ensemble collectivement.

Cependant, nous allons voir qu'un communication simplement déployée pose de nombreux problèmes de sécurité et de confidentialité. Le système initial ne suffit pas à protéger nos données face à un attaquant connaissant un peu le système.

Le deuxième objectif consistera donc à discuter de la surface d ?attaque de notre système. Un aspect analyse est donc demandé en prenant en compte les différentes versions du LoRaWAN, chacun des éléments du système etc.

Ainsi, les communications LoRaWAN ont de nombreux vecteurs d'attaque, dont nous devons prendre connaissance pour pouvoir y remédier.

1.3 Vocabulaire

Nous souhaitons faire ici un rappel des termes techniques que nous emploierons pour éviter tout malentendu.

LoRaWAN Long Range Wide Area Network. Protocole de communication.

Noeud Ensemble de composants qui peuvent recevoir et/ou envoyer de l'information via le protocole de communication LoRaWAN. Branche initiale d'un réseau LoRaWAN. Par exemple, un capteur relié à une carte/microcontrôleur et une antenne pour la communication vers l'extérieur.

Passerelle ou *Gateway* Élément de transfert. Permet de traduire et transférer les données venant du noeud vers les serveurs.

Network server Cerveau du réseau LoRaWAN, il génère les clefs et authentifie les noeuds. Il déchiffre aussi une partie des trames du réseau, reçues via la passerelle.

Application server Service qui va traiter l'information du capteur, il va déchiffrer la dernière partie du message.

Box LoRa Pour notre projet, sera un micro-ordinateur *Raspberry* qui va contenir la passerelle, le *network server* et l'*application server*

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 nécessaires au projet. Pour nous organiser tout au long de la période du projet, nous avons établi un diagramme de *GANTT*. Nous l'avons gardé à jour pendant toute la durée du projet. Pour avoir une gestion de projet plus précise (tâches à effectuer chaque semaine), 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 status, nous ajoutions aussi de nouvelles tâches au cours 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 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 passer 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 micro contrôleur 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 implémenter des éléments de sécurité dans l'OS, nous utilisons le logiciel *Yocto*, car nous l'avions déjà utilisé en début d'année.

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 tous 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 a 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 apporter des modifications à l'OS *ChirpStack*.

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

Chapitre 3

Réalisation et Tests

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

3.1 Démonstrateur

Comme nous l'avons dit dans le chapitre précédent 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 utilisé l'OS *ChirpStack*[1] sur la *Raspberry Pi* équipée de son *shield* antenne *iC880A* pour le nœud nous avons utilisé la carte *Fipy*[3] avec son *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 services sont aussi facilement configurables grâce à une interface web.

La carte *Fipy*, est une carte de développement orientée pour l'utilisation de réseaux sans fils comme WiFi, Bluetooth ou LoRaWAN. Cette carte se programme en *Python* avec l'extension *Pymakr* pour les éditeurs de texte *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 particuliers, autres que des problèmes de configuration de l'extension *Pymakr*.

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

3.2 Nœud

Pour le rendu final du projet le nœud doit fonctionner sur une carte B-L072Z-LRWAN1[4] qui est une carte de découverte produite par STmicroelectronics orientée pour le développement de solutions basées sur des réseaux *LoRaWAN* ou *SigFox*. Cette carte est équipée d'une MCU *STM32L072CZ* qui est basée sur une architecture Arm Cortex M0+.

Pour rappel, dans l'architecture LoRaWAN, le rôle du nœud est de transmettre une information qui serra traitée 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 concentrés, 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.

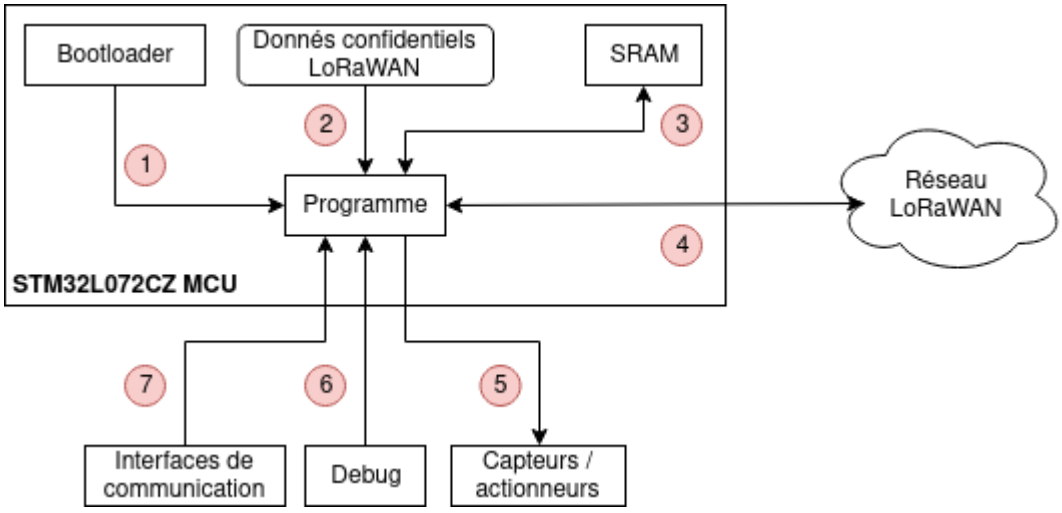


FIGURE 3.1 – Schéma présentant les différents éléments autour de la carte B-L072Z-LRWAN1

Il y a plusieurs vecteurs 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 accés à tous les contenus du micro

contrôleur.

2. Accès à des parties du programme sensibles pour les copier ou les voler
3. Utilisation de failles dans la SRAM comme un buffer overflow pour voler des informations dedans ou provoquer un déni 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 avoir 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 systéme pouvant détecter une intrusion au niveau de la carte.
6. Désactiver les fonctionnalités de débugage
7. Rendre les interfaces de communications difficiles d'accés. 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 commencé par faire des recherches sur quel méthode de connexion était la plus intéressante à sécuriser par rapport à nôtres scénario.

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

Fonctionnement activation OTAA

L'activation par **OTAA** met en jeux *DevEUI*, *AppEUI*, *AppKey* pour ce connecter à la *gateway* puis utilise *AppSKey* et *NetSKey* pour chiffrés la communication jusqu'à l'*application server*. Le protocole LoRaWAN ne précise pas si les clés

doivent être stocké en clair dans la mémoire ou doivent être chiffrés. Il faut *AppSKey* et *NetSKey* sont renouvelé à chaque Si un attaquant arrive à obtenir les trois premières clés alors, il pourra usurper l'identité de notre nœud. S'il parvient à obtenir les deux autres clés, il pourra uniquement déchiffrer l'information.

Fonctionnement activation ABP

L'activation par personnalisation met en œuvre *DevAddr*, *NwkSKey* et *AppSKey* ces deux dernière sont les clés permettant de chiffrer la communication ainsi, si un attaquant les récupéres, il peut tout aussi bien usurper l'identité de notre nœud que lire les informations envoyés.

Analyse des risques, pour la connexion d'un nœud à un réseau LoRaWAN

Dans un premier temps on remarque que le protocole LoRaWAN ne stipule pas si les clés doivent être stocké en clair dans la mémoire ou bien chiffré. Cela veut dire que si un attaquant fait un *dump* de la mémoire, il va pouvoir trouver facilement les clés. On remarque ensuite, que l'utilisation de l'activation par **OTAA** est plus sécurisé car elle utilise plus de clés et en renouvelle 2 à chaque connexions. La connexions par **ABP** est moins sécurisé car elle utilise uniquement trois clés, dont deux qui peuvent aussi permettre l'écoutes de la communication.

Dans le scénario que nous utilisons nous n'avons pas besoins de dissimuler l'information transmise, donc obtenir *NetSKey* et *AppSKey* lors d'une activation **OTAA** nous importe peut.

D'un point de vue technique le nombre de clés à récupérer par attaquant pour usurper l'identité de notre nœud est le même d'un mode d'activation à l'autre. Nous avons choisit d'utiliser et de protéger le mode d'activation **ABP**, car si un attaquant arrive à obtenir les clés il pourra lire les informations transmises et/ou usurpé l'identité de nôtres nœud, de plus nous trouvons intéressant de pouvoir observer les trames LoRaWAN ce propager dans l'air à l'aide d'un analyseur de spectre et de pouvoir les décodé avec un logiciel comme *GNURadio*.

3.2.2 Nœud LoRaWAN avec une carte B-L072Z-LRWAN1

Le premier objectif que nous nous sommes fixé était de créer un nœud LoRaWAN non sécurisé comme l'on avait fait avec le démonstrateur.

Pour cela nous avons trouvé le projet *LoRaMac-node*[2] sur *Github*, qui est un projet permettant de configurer facilement un nœud LoRaWAN avec des cartes de développement dont la carte B-L072Z-LRWAN1 que nous utilisons. De plus dans

sa documentation il est indiqué comment utiliser un composant de sécurité. Vous pouvez voir la carte B-L072Z-LRWAN1 sur la figure 3.2

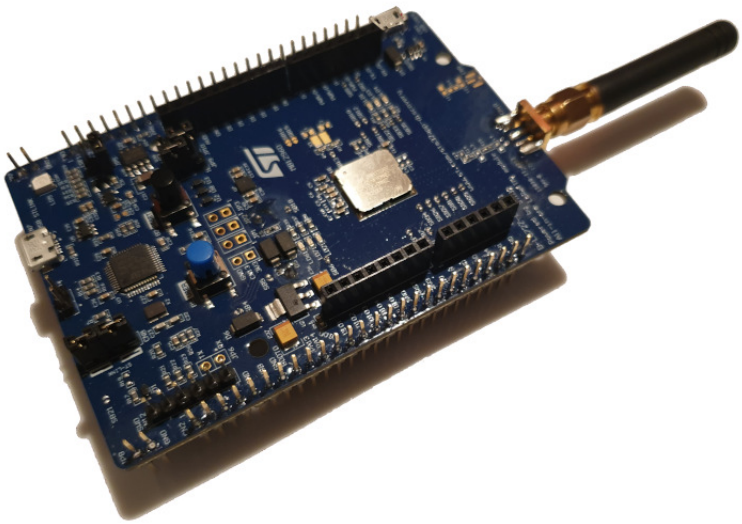


FIGURE 3.2 – Photographie de la carte utilisé comme nœud

Le projet *LoRaMac-node* permet d'utiliser les trois différentes class de nœud LoRaWAN et avec plusieurs versions du protocole 1.0 à 1.0.3.

Pour fonctionner sur différentes plateformes le projet utilise des fonctions utilisant d'autres fonctions propre aux cartes sur lesquels il peut être déployer. Pour compiler un programme sur les bonnes cartes il utilise, **Cmake** pour générer des *makefile* lesquels permettent ensuite d'inclure les *lirairie* correspondant à la cible.

3.2.3 Sécurisation carte STM32

Comme nous l'avons dit précédemment, le nœud doit pouvoir stocker de manière sécuriser des clés. Pour cela nous avons deux solutions, stocker les clés dans un élément sécurisé ou les stocker dans la mémoire du micro contrôleur de manière sécurisé.

Nous avons choisit dans un premier temps de stocker les clés dans la mémoire, afin d'éviter d'utiliser un élément de sécurisé qui pourrait augmenter le coûts de notre projet, et sa consommation d'énergie. Cependant si les mesures de sécurité offertes par cette première solution ne sont pas suffisantes nous pouvions dans un second temps utiliser un élément de sécurité.

Fonctions de sécurité dans les MCU STM32L0

Dans un premier temps de recherche nous avons trouvé que les MCU STM32L0 contenait déjà des contres mesures pour différents types d'attaques, prêtes a l'emploi à l'aide de fonctions *HAL*[1]

Vous pouvez trouver la liste des protection présentes dans notre MCU ci-dessous :

- **RDP** (Read Out Protection) : Protection de la mémoire Flash qui empêche la copie du code. Cette fonction prévient donc du reverse engineering fait à l'aide des outils de debugage. Empêche aussi de charger une nouveau programme sur la carte.
- **WRP** (Write Protection) : Sert à protéger une partie de la mémoire Flash d'un effacement ou d'une mise à jour.
- **PCROP** (Proprietary code read-out protection) : Permet de configurer certaines partie de la mémoire flash pour qu'elles soient uniquement accessible par le bus d'instruction du CPU (Execution Only).
- **Firewall** : Le par feu est un composant physique qui contrôle les accès à trois parties de la mémoire, la zone du code (mémoire flash), les donnés volatile (SRAM) et les donnés non volatile (Flash).
- **MPU** (Memory Protection Unit) : Mécanisme de protection qui permet de définir des droits d'accès à certaines zone de la mémoire.
- **Anti temper** : Détection d'une intrusion au niveau physique, permet de prendre les mesure adéquates comme effacer la mémoire par example.
- **IWDG** (Independant watchdog) : Watchdog Independant permettant de lever des *flags* si une tache prend plus de temps que celui qui lui est attribué.

Pour rappel les deux attaques contre les quelles nous allons protéger la carte sont : **Une lecture de la mémoire (Dump mémoire)** et **Accés à la mémoire par une partie du programme non autorisé**. Pour protéger le nœud de la première nous avons choisit de mettre d'utiliser les sécurité **RDP** et **PCROP**, pour la deuxième **PCROP** et **Firewall**.

Protection RDP La RDP offre différents niveaux de protection 0, 1, 2.

- **Le niveau 0**, équivaut à aucune protection la mémoire flash et complètement lisible, peut importe le mode de démarrage du contrôleur (par la RAM, par la flash, par le debuger ...). Ce mode doit être utilisé uniquement pour la phase de développement.
- **Le niveau 1**, empêche l'accès à la mémoire flash et à la mémoire SRAM2 par le debuger. Cependant lorsque le programme démarra à partir de la

1. Hardware Abstraction Layer

mémoire flash celui-ci à accès à la mémoire flash et la SRAM2.
— **Le niveau 2**, ce niveau empêche tout les accès aux mémoire depuis l'extérieur. **Attention après l'avoir activé on ne peut plus revenir en arrière**.

Il faut faire très attention lors de l'utilisation de la **RDP** car dès que l'on utilise une protection de niveau supérieure à 1, on ne peut plus mettre de programme dans la carte via le bootloader. Pour mettre à jour le programme il faut utiliser une *SFU*[2].

Protection PCROP La **PCROP** permet de sécuriser la mémoires par secteur[3]. Il sert principalement à protéger les propriétés intellectuelles d'un programme, mais il peut aussi, permettre de dissimuler des code sensible dans une mémoire comme un code permettant de générer des clés cryptographiques. Lorsque un secteur est sécurisé, son accès est uniquement possible via le bus de d'instruction. Si un secteur protégé est lu il retournera une erreur sur le bus de lecture.

Firewall Comme le **PCROP**, le **Firewall** protége des secteur de la mémoire définit à l'initialisation du programme. Le **Firewall** est un composant physique qui va contrôler les et filtrer les accès entre 3 parties : La zone du Code dans la mémoire Flash, une zone de mémoire volatile dans SRAM et une zone de mémoire non volatile dans la Flash. L'accès à un code sécurisé par un **Firewall** ce fais par une seule fonction, si une autre fonction essaie d'accéder à ce code, le **Firewall** générera un *reset* de la carte

Mass Erase Avant d'implanter ces contres mesures dans un programme nous avons chercher comment es désactiver pour éviter d'être bloqué avec une carte non utilisable et ainsi pouvoir effectuer plusieurs tests. Pour pour pouvoir passer du niveau **RDP** 1 à 0 et pour désactiver **PCROP**, il faut effectuer un *mass erase* sur la carte ce qui à pour conséquence de mettre toute les mémoire Flash et SRAM à 0. Pour effectuer une mass erase il y a deux possibilité, la première et d'écrire une fonction faisant un *mass erase* dans la RAM et la deuxième est d'utiliser l'utilitaire *ST-link utility* qui permet de faire en autre ce genre d'opération sans avoir à les programmer.

Avant d'implémenter les protections, nous avons réaliser des dumps de la mémoire lorsque un programme fonctionnait sur la carte pour vérifier que nous trouvions bien les clés écrites dans le code.

2. Secure Firmware Update
3. La taille des secteur dépend du micro contrôleur, dans nôtres cas, 1 secteur = 32 pages = 4Ko

Pour cela nous effectuons un dump de la mémoire à l'aide du logiciel *St-Link* et nous ouvrions le fichier récupéré avec *Ghidra* qui nous permettait de revenir presque au code C de base. Vous pouvez voir en figure 3.3 une des clés présente dans la mémoire.

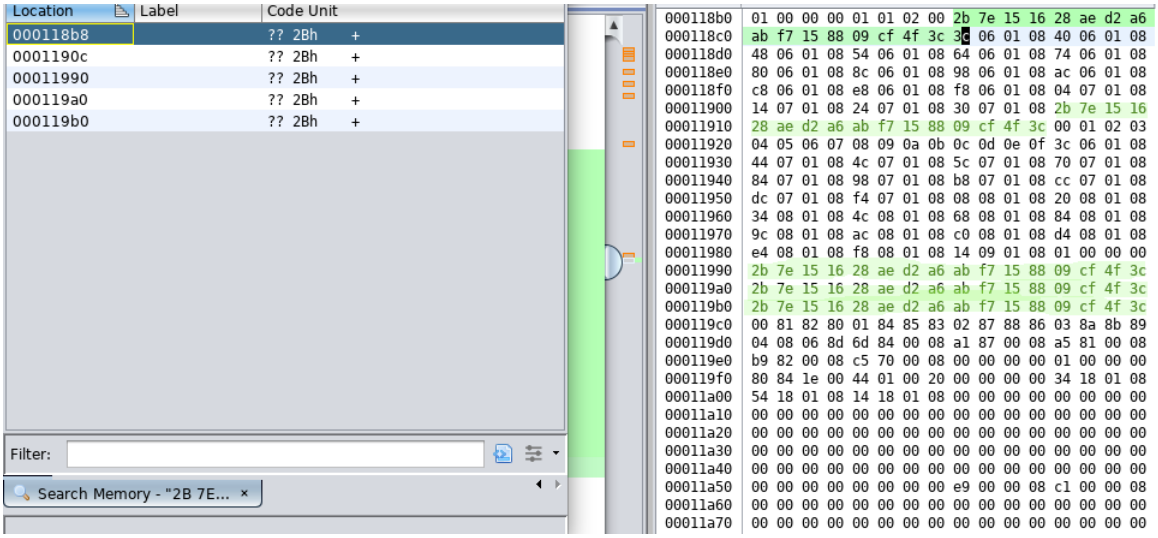


FIGURE 3.3 – Capture d'écran de *ghidra* sur laquelle on voit une des clés dans la mémoire

Pour être sur que nous trouvions bien la clés et pas des variables aléatoire formant la même suite nous avons essayé avec trois clés différente et nous cherchions à chaque fois les 3 clés, pour être sur que celle trouvé ne soit pas une coïncidence. Vous pouvez trouver en Annexe A les conclusions et note sur les *dump* de la mémoire.

3.2.4 Problèmes rencontrés

A ce jour, nous n'avons pas réussi à implémenter les contres mesures de sécurité présenté ci-dessus dans le programme LoRaMAC-node car, pour implémenter ces contres mesures il faut utiliser des fonctions *HAL* propres au STM32 qui sont définit dans des librairies différentes d'un micro contrôleur à l'autre.

Ces librairies ce trouves dans le projet *LoRaMAC-node*, mais lorsque nous utilisons les fonctions *HAL* pour par exemple activer le niveau 1 de **RDP** le compilateur ne parvient pas à faire le lien entre l'appelle des fonctions et leur définitions dans les *librairie*.

Hypothése 1

Les librairies utilisées pour définir les fonctions ne sont pas incluses dans le programme.

Pour vérifier si les fonctions sont bien définit, nous avons essayé de lister tout les *include* qu'appelle chaque librairie du programme. Pour vérifier si le programme finit bien par appeler la librairie nécessaire à nos fonctions.

Au pour finir nous n'avons pas réussi à trouver avec certitude quelle librairies étaient utilisés.

Pour palier à ce problème nous avons décidés de programmer les différentes contres mesures dans un programme simple dans lequel nous avons dissimulés une variable contenant des clés. Pour d'abords être sur que les contremesure fonctionne et que le code que nous implémenterons dans le programme *LoRaMAC-node* soit validé.

Une autre solution été possible, en effet avec le programme *St-Link Utility* il est possible de modifier les octets d'options d'un programme (non sécurisé), et ce même lorsque le programme est sur la carte. *St-Link utility* vas récupérer le contenu de la mémoire et l'écrire à nouveau avec les modification apporté. Cela nous permet entre autres de modifier facilement le Niveau **RDP**, et **PCROP** de n'importe quel programme non sécurisé sur la carte.

3.2.5 Tests Unitaire

Tests RDP

Pour tester le fonctionnement de la contre mesure **RDP** nous l'avons implanté dans notre programme simple. Puis nous avons essayés de lire la mémoire avec *St-Link Utility* s'il nous affichait un message d'erreur disant de désactiver *Read Out*

Protection nous allions vérifier les Octet d'option du programme et regardions le niveau de la *RDP*. Comme vous pouvez le voir sur la figure 3.4

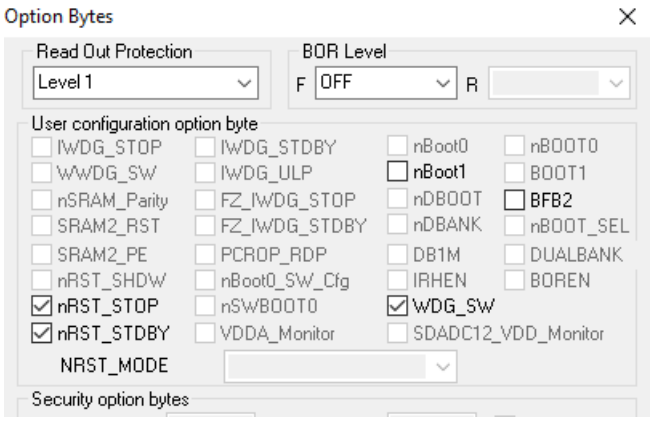


FIGURE 3.4 – Capture d'écran de *St-link utility* sur laquelle on voit le niveau RDP de la carte

Tests PCROP

Pour vérifier si *PCROP* a bien été implémenté il faut brancher la carte au programme *St-link Utility* et afficher les octets d'option pour voir les secteurs sécurisé, comme on peut le voir sur la figure 3.5

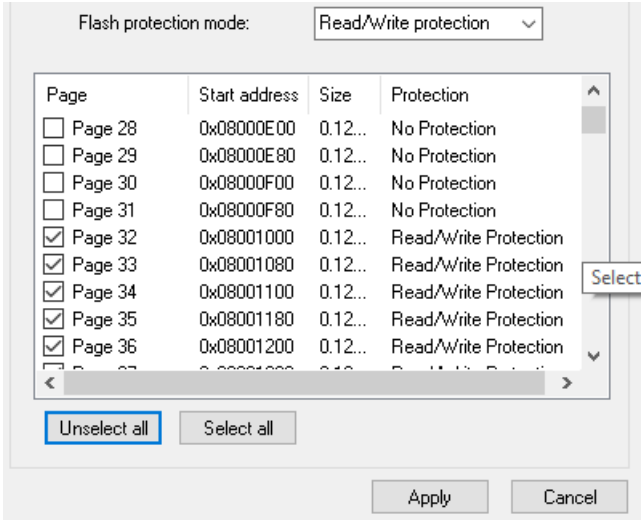


FIGURE 3.5 – Capture d'écran du logiciel *St-Link Utility* sur laquelle on peut voir des secteurs sécurisés et non sécurisés

3.2.6 Tests Intégration

Le tests d'intégrations effectué jusqu'à présent est la mise en fonctionnement d'un programme simple, de *RDP* et de *PCROP*. Pour cela on utilise de nouveaux la visualisation des octets d'options dans *St-link Utility*. Pour vérifier de la carte utilise bien le bon niveau RDP et qu'elle sécurise les bon secteurs mémoire. Pour vérifier que notre programme fonctionne toujours on regarde si les LEDs qu'il doit vérifier fonctionnes toujours.

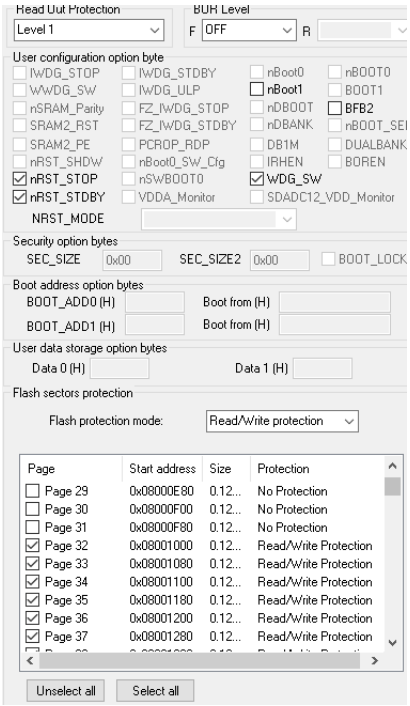


FIGURE 3.6 – Capture d'écran du logiciel *St-Link Utility* sur laquelle on peut voir les secteurs sécurisés et non sécurisés ainsi que le niveau de protection RDP

3.2.7 Conclusion

Pour la suite du projet il nous restera à trouver comment stocker une fonction à un endroit précis de la mémoire pour pouvoir utiliser *PCROP* car actuellement nous savons sécuriser les secteurs souhaités mais sans savoir où se trouve la fonction à sécuriser dans la mémoire.

Il nous restera aussi à utiliser le *Firewall* si besoins car il faudra faire une étude de besoins pour vérifier si *PCROP* n'est pas suffisant dans nôtres cas.

La partie nœud nous a permis de découvrir les différentes solutions de sécurité intégrés aux micro contrôleurs STM32.

3.3 Passerelle

3.3.1 Surface d'attaque de la *Box LoRa*

Concernant la *Box LoRa*, nous avons affaire aux problématiques suivantes : comment faire pour garantir une compartimentation parfaite et complète de tous les services proposés ?

Notre *Box LoRa* peut fournir plusieurs services. Le service initial minimum est la réception de données provenant du noeud, sa traduction et son utilisation (afficher une donnée sur un moniteur, faire actionner un moteur, etc). Mais on peut penser qu'un fournisseur voudrait proposer plusieurs services avec une telle box. Par exemple, pour une société qui voudrait utiliser les communications en LoRa pour relever des données de ses capteurs, mais également avec un service de messagerie (mails) en interne, ou bien une zone de dépot et partage de documents sur la *Box LoRa*, alors apparaissent certaines contraintes, notamment de sécurité.

En effet, il faut garantir les contraintes CIAN (Confidentialité, Intégrité, Authenticité, Non-répudiation) pour les données envoyées et reçues.

On peut donc voir que si chaque service n'est pas bien isolé des autres, ces contraintes pourraient ne pas être respectées respectées. Par exemple, si le compte utilisé pour la messagerie est un *super-utilisateur*, il peut avoir les droit pour aller modifier des fichiers de configuration importants, faire arrêter le système, modifier les données, etc.

Ainsi, nous devons apporter des solutions pour protéger notre *Box LoRa*

3.3.2 Solutions envisagées et solution retenue

Pour répondre à ces problématiques, nous avons étudié plusieurs solutions.

Premièrement, nous avons considéré les modèles d'habilitation. Ces modèles d'habilitation permettent de garantir, au sein d'une entreprise par exemple, un contrôle d'accès performant sur les ressources et les services. Ils permettent également de rationaliser le processus de demande d'accès des utilisateurs.

Les principaux sont :

— DAC (Discretionary Access Control)
— MAC (Mandatory Access Control)
— ABAC (Attribute Based Access Control)
— RBAC (Role Based Access Control).

Nous donnons ci-desssous une rapide description de leur fonctionnement :

Dans le modèle DAC (ou Contrôle d'Accès Discrétionnaire), chaque personne est administrateur de ses ressources. Le fait de posséder un objet permet de modifier les droits d'accès sur celui-ci.

Comme points positifs, on peut considérer : Facile à implémenter ;Offre une grande flexibilité ; Intégré à la plupart des systèmes d'exploitation.

Les points négatifs sont plus nombreux : Modèle inadapté à un système comportant un nombre important d'utilisateurs ; Ne reflète pas le flux réel de l'information dans un système, les informations autorisées pouvant être copiées d'un objet à un autre ; Aucune restriction ne s'applique à l'utilisation des informations lorsque l'utilisateur les a reçues ; Sujet à de nombreuses erreurs lors de l'attribution des autorisations par le propriétaire de l'objet.

Dans le modèle MAC, l'accès aux objets est restreint en fonction de la sensibilité des informations (classification) contenues dans les objets et du niveau d'autorisation de l'utilisateur de disposer d'informations d'une telle sensibilité. On aura par exemple : un technicien habilité niveau 3 ne pourra pas accéder à un dossier de niveau 5, mais il aura accès aux dossiers de niveau 1, 2 et 3.

Les niveaux proposés pour l'exemple sont les suivants :

— Niveau 1 : néant
— Niveau 2 : Non classifié
— Niveau 3 : Confidentiel, non-classifié
— Niveau 4 : Secret, confidentiel, non-classifié
— Niveau 5 : Top secret, secret, confidentiel, non-classifié

Le principal point positif est que cette méthode offre un niveau hautement sécurisé d'administration aux sources d'information. En effet, une autorité centrale applique les décisions de contrôle d'accès, et non par le propriétaire individuel d'un objet (ressource). Et le propriétaire ne peut pas modifier les droits d'accès.

Les points négatifs sont : Modèle très rigide. Il impose des restrictions sur l'accès des utilisateurs qui, conformément aux politiques de sécurité, ne permettent pas les modifications dynamiques. ; Inadapté aux systèmes repartis. ; Assez coûteux en étude, car il nécessite une planification prédéterminée pour être mis en ?uvre

efficacement. ; Assez coûteux en exploitation. Après la mise en œuvre, un mode de gestion complexe est nécessaire à cause de la mise à jour constante des étiquettes d'objet et de compte, pour collecter de nouvelles données.

Nous ne détaillerons pas les autres modèles d'habilitations.

Par rapport aux problématiques de sécurité définies plus haut, nous pensons que le modèle MAC est le plus adapté pour notre *Box LoRa*. Nous choisissons donc ce modèle comme angle d'attaque.

Ainsi, nous pouvons proposer l'utilisation de deux modules intéressants qui semblent être adaptés : SELinux et AppArmor. Ces deux logiciels sont un moyen de mettre en œuvre un système MAC. Nous allons faire un rapide comparatif afin de prendre une décision sur l'emploi d'un tel ou d'un autre.

Chapitre 4

Conclusion

Nous avons maintenant compris le fonctionnement général d'un système de communications complet en LoRaWAN. Avec l'expérience que nous avons acquise, nous sommes à même de voir les failles potentielles tout au long système et de déployer des moyens pour sécuriser au maximum la communication. En prenant encore plus de recul, et avec le mois supplémentaire de projet, nous allons pouvoir aller plus loin en attaquant nous même notre système pour tenter d'y déceler d'autres vulnérabilités. Les éventualités qui peuvent nous arriver sont très nombreuses et nous ne pouvons pas toutes les anticiper, néanmoins, nous faisons le maximum avec nos connaissances actuelles pour aller le plus loin possible dans la sécurisation.

En terme de bénéfices, ce projet nous a beaucoup appris sur notre façon de travailler et de nous organiser. Nous avons appris à faire des recherches documentaires très poussées, à faire de nombreux tests unitaires et d'intégration pour contrôler nos avancées, à travailler dans des domaines inconnus et nous avons pris beaucoup de plaisir à découvrir cet environnement.

Travailler en binôme nous a permis d'avoir une efficacité autre qu'en solitaire. Il faut savoir écouter l'autre pour pouvoir résoudre nos problèmes communs. Nous avons beaucoup appris en organisation et en méthodologie lors de ce projet.

Nos mois de projets ayant été entrecoupés de cours et autres séminaires, nous avons perdu un peu de temps pour les parties finales, qui demandent notamment de coder des morceaux. En ayant eu une plus large continuité, nous aurions pu commencer à coder beaucoup plus tot et avoir bien plus d'avance. Néanmoins, nous avons appris à travailler dans l'urgence, avec des contraintes à respecter (réunion hebdomadaires, comptes rendus, rapports, présentations, etc).

Ce projet fut une belle découverte de l'expérience du travail de projet en équipe dirigée.

Table des figures

Annexe A

Notes sur les différents dump de la mémoire

A.1 Test Dump mémoire STM32

Pour ces tests nous utilisons 3 clés différentes pour chaque tests. Afin de vérifier que nous trouvons bien la clés et pas des valeurs similaire trouvé dans la mémoire.

- **Clef 1** : 0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C
- **Clef 2** : 0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x5F, 0x3C
- **Clef 3** : 0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x3F, 0x3C

A.1.1 Conclusions sur les validations

On remarque que à chaque fois nous avons pu retrouver la/les clef(s) dans la mémoire en utilisant le debugger. Si l'on trouve plusieurs clés pour le programme LoRaMAC c'est car nous avons modifier à chaque fois uniquement les 2 clefs de sessions nécessaires. Le programme LoRaMAC mais en œuvre 5 clefs au total.

Nom du test	Programme	Clés	Nb clés 1	Nb 2	Nb 3
Validation 1	LoRaMAC	Clés 1	5	0	0
Validation 2	LoRaMAC	Clés 2	3	2	0
Validation 3	LoRaMAC	Clés 3	3	0	2
Validation 4	ProgrammeTest	Clés 1	1	0	0
Validation 5	ProgrammeTest	Clés 2	0	1	0
Validation 6	ProgrammeTest	Clés 3	0	0	1

Lorsque l'on met en place la sécurité : *RDP* de niveau 1, il n'est alors plus possible d'utiliser les ports de debugage pour venir lire la mémoire.

On remarque qu'il est facile de lire la mémoire d'un micro contrôleur pour un attaquant et qu'il est donc important de verrouiller les ports de debug.

Il faut aussi noter que pour passer d'un niveau RDP 1 à RDP 0, le procédé est tout aussi facile avec l'utilitaire st-link utility, bien qu'il efface intégralement la mémoire et efface donc le programme qu'elle contient.

Bibliographie

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

[2] Lora net / Semtech. Loramac-node. https://github.com/Lora-net/LoRaMac-node.

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

[4] 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.