

HELMo Campus Guillemins

Rapport- Heltec ESP32 LoraWan

UE46 : Sécurité de l'IOT – 3S1 B3 Cyber sécurité

Julien Diet – Thomas Huppe

03/12/2025

Table des matières

1. Introduction.....	3
1.1 Contexte et Méthodologie.....	3
1.2 Objectifs du laboratoire.....	3
2. Configuration de la toolchain et implémentation du nœud LoRaWAN	4
2.1 Mise en place de l'environnement de développement	4
2.2 Configuration du protocole LoRaWAN	4
3. Audit de sécurité offensif et Clonage du dispositif	5
3.1 Extraction du firmware (Dumping)	5
3.2 Analyse différentielle et définition du pattern	5
3.3 Exfiltration des secrets du binôme	6
3.4 Usurpation d'identité (Spoofing)	7
4. Conclusion et Remédiations	8
4.1 Bilan de l'attaque	8
4.2 Contre-mesures et Sécurisation	8
5. Bibliographie.....	9

1. Introduction

Ce rapport présente l'analyse de sécurité et les manipulations techniques réalisées dans le cadre du laboratoire de sécurité de l'IoT. Ce travail se concentre sur l'étude d'un objet connecté utilisant le protocole LoRaWAN, implémenté sur une carte de développement Heltec WiFi LoRa 32 (V3) architecturée autour d'un microcontrôleur ESP32.

Au-delà de la simple mise en service d'un nœud réseau, ce laboratoire a pour vocation d'illustrer concrètement la criticité de la sécurité physique dans l'IoT. Il s'agit de démontrer comment l'accès matériel à un dispositif, souvent considéré comme le talon d'Achille de la sécurité des systèmes embarqués, permet de contourner les protections logicielles si des contre-mesures adéquates ne sont pas déployées.

1.1 Contexte et Méthodologie

Afin de reproduire un scénario d'attaque réaliste, ce laboratoire a été mené en binôme selon une approche symétrique. Dans un premier temps, chaque étudiant a configuré son propre dispositif pour établir une communication légitime et sécurisée avec la passerelle LoRaWAN. Dans un second temps, les équipements ont été échangés physiquement.

Cette phase offensive simule une situation où un attaquant parvient à dérober ou à accéder physiquement à un capteur déployé sur le terrain. L'objectif était alors d'éprouver la robustesse du matériel de mon binôme face à une tentative d'extraction de firmware. Cette démarche pratique permet de vérifier l'hypothèse théorique selon laquelle un accès physique non sécurisé (ports de débogage ouverts, absence de *Flash Encryption*) équivaut souvent à une compromission totale du système et de ses secrets.

1.2 Objectifs du laboratoire

Les travaux réalisés visent à valider plusieurs compétences techniques clés, allant de l'ingénierie embarquée à l'audit de sécurité offensif :

- **Déploiement de l'environnement de développement** : Installation et configuration de la chaîne de compilation (*toolchain*) ESP32 et des pilotes nécessaires pour interagir avec le microcontrôleur via l'interface série (USB-C).
- **Implémentation du protocole LoRaWAN** : Configuration logicielle du nœud pour rejoindre le réseau via le mode OTAA (*Over-The-Air Activation*), nécessitant la gestion de clés cryptographiques (AppKey, DevEui).
- **Exploitation des vulnérabilités matérielles** : Réalisation d'une extraction intégrale de la mémoire flash via le port série, profitant de l'absence de restrictions de lecture sur le bootloader par défaut.
- **Analyse et exfiltration de données** : Utilisation de techniques de rétro-ingénierie assistée par outil (Ghidra) pour localiser les identifiants stockés en clair dans le binaire, illustrant le risque de ne pas utiliser de stockage sécurisé (*Secure Element ou Encryption*).
- **Preuve de concept d'usurpation (Spoofing)** : Reprogrammation d'un dispositif tiers avec les secrets exfiltrés pour cloner l'identité du nœud victime sur le réseau LoRaWAN.

2. Configuration de la toolchain et implémentation du nœud LoRaWAN

La première phase du laboratoire consistait à rendre opérationnel notre propre dispositif afin de valider la communication avec la passerelle LoraWan. Cette étape est critique car elle permet de s'assurer que le matériel est fonctionnel avant de procéder à des manipulations plus offensives.

2.1 Mise en place de l'environnement de développement

L'architecture du microcontrôleur ESP32 nécessite une chaîne de compilation spécifique (*toolchain*) pour être programmée via l'IDE Arduino. La préparation de l'environnement de travail a suivi les étapes suivantes :

1. **Installation des pilotes de communication** : Le module Heltec communique avec le poste de travail via un pont USB-UART. L'installation des pilotes **CP210x** (Silicon Labs) a été nécessaire pour émuler un port série (identifié ici comme COM3) et permettre le téléchargement du firmware.
2. **Intégration du Board Manager** : Afin de rendre l'IDE Arduino compatible avec l'architecture spécifique de la carte, nous avons ajouté les définitions matérielles via le gestionnaire de cartes (*Boards Manager*), en sélectionnant le paquet "Heltec ESP32 Series Dev-boards".
3. **Bibliothèques LoRaWAN** : L'implémentation du protocole réseau repose sur la bibliothèque Heltec ESP32 Dev-boards qui abstrait la complexité de la modulation radio (couche physique LoRa) et de la gestion de la pile MAC LoRaWAN.

Avant d'implémenter la couche réseau, une validation matérielle élémentaire a été réalisée en téléversant un programme "Blink". Ce test a confirmé le bon fonctionnement du bootloader et l'intégrité de la liaison série, vecteurs qui seront ultérieurement exploités lors de l'attaque.

2.2 Configuration du protocole LoRaWAN

Pour la communication réseau, nous avons utilisé le code source LoRaWan.ino fourni dans les exemples du constructeur. Ce code est préconfiguré par défaut pour utiliser le mode d'activation OTAA (*Over-The-Air Activation*), via la variable overTheAirActivation = true.

Nous avons conservé ce mode de fonctionnement qui privilégie la sécurité en négociant dynamiquement les clés de session lors de la phase de jointure (*Join Procedure*), rendant de facto inutiles les paramètres ABP (NwkSKey et AppSKey) présents dans le code source.

Afin d'identifier notre matériel de manière unique sur le réseau, nous avons apporté les modifications manuelles suivantes au firmware :

- **devEui (Device Extended Unique Identifier)** : Nous avons personnalisé cet identifiant de 64 bits afin de définir une clé reconnaissable propre à notre dispositif.
 - *Valeur définie* : 0x7C, 0xB7, 0xD7, 0x7C, 0xD7, 0x07, 0x77, 0xC7.
- **appEui (Application Identifier)** : L'identifiant d'application a été ajusté pour correspondre à la configuration attendue.
 - *Valeur définie* : 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00.

- **appKey** : La clé racine AES-128 par défaut a été conservée pour cette première étape.

Après compilation et upload, le dispositif a initié avec succès une procédure de join, validée par la réception des paquets sur la console de la passerelle LoRaWAN. Cette étape a confirmé que nous disposions d'un dispositif sain et fonctionnel avant l'échange avec le binôme.

3. Audit de sécurité offensif et Clonage du dispositif

La seconde phase du laboratoire a consisté à simuler une attaque par accès physique. Après avoir échangé nos cartes avec mon binôme, l'objectif était d'extraire le firmware de son dispositif pour en récupérer les secrets d'authentification et usurper son identité sur le réseau.

Pour ce faire, nous avons adopté une méthodologie rigoureuse en deux temps : une première phase d'analyse sur notre propre matériel pour identifier la structure des données (le "pattern"), suivie de l'exploitation sur le matériel cible.

3.1 Extraction du firmware (Dumping)

L'accès physique au microcontrôleur permet d'interagir directement avec son bootloader via l'interface série. Comme vu dans le module sur les vulnérabilités matérielles, si les fusibles de protection (*eFuses*) ne sont pas configurés pour désactiver le téléchargement UART, la mémoire est lisible.

Nous avons utilisé l'outil esptool pour réaliser une copie brute de la mémoire flash. La commande suivante a permis d'extraire la plage mémoire contenant le code applicatif et les données (0x10000 à 0xF0000) vers un fichier binaire :

```
python -m esptool --chip esp32-S3 --port COM3 read_flash 0x10000 0xF0000 firmware.bin
```

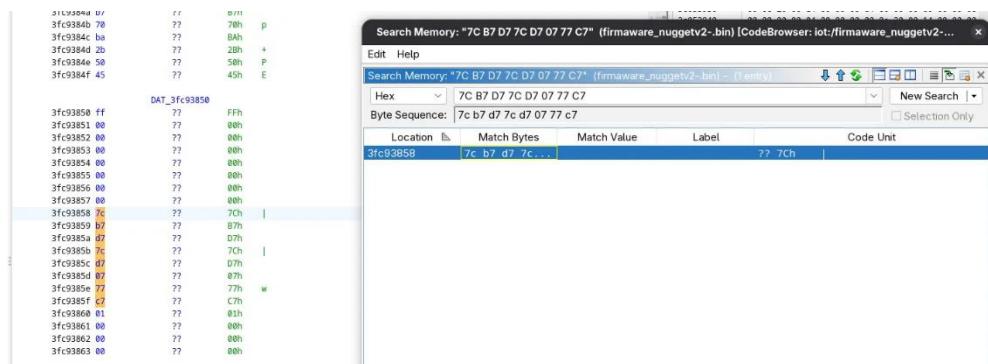
Cette opération a été effectuée deux fois : d'abord sur ma propre carte (pour analyse), puis sur la carte de mon binôme (pour l'attaque).

3.2 Analyse différentielle et définition du pattern

Une fois les binaires obtenus, nous avons procédé à une analyse statique à l'aide de l'outil de rétro-ingénierie Ghidra, équipé du plugin *ESP32 Flash Image Loader*.

Avant de tenter de localiser la clé inconnue de mon binôme, j'ai d'abord analysé mon propre firmware afin de définir un modèle de recherche fiable :

1. **Recherche de signature connue** : Sachant que ma propre clé DevEui était configurée à 0x7C, 0xB7, 0xD7, 0x7C, 0xD7, 0x07, 0x77, 0xC7, j'ai effectué une recherche de cette séquence d'octets précise dans la mémoire de mon dump.



2. **Identification de l'emplacement mémoire :** Cette recherche a abouti et m'a permis d'identifier l'adresse mémoire exacte (offset) où le compilateur stocke cette variable globale. J'ai ainsi pu confirmer que la clé est stockée de manière contiguë et en clair (*plaintext*) dans la section *.rodata*.

```

offset du début de la clé : 0x3fc94d58

firmware_julien.bin - (8 addresses selected)
3fc93845 82      ??      82h
3fc93846 48      ??      48h H
3fc93847 27      ??      27h '
3fc93848 fe      ??      FEh
3fc93849 c5      ??      C5h
3fc9384a b7      ??      B7h
3fc9384b 70      ??      70h p
3fc9384c ba      ??      BAh
3fc9384d 2b      ??      2Bh +
3fc9384e 50      ??      50h P
3fc9384f 45      ??      45h E
3fc93850 ff      ??      FFh
3fc93851 00      ??      00h
3fc93852 00      ??      00h
3fc93853 00      ??      00h
3fc93854 00      ??      00h
3fc93855 00      ??      00h
3fc93856 00      ??      00h
3fc93857 00      ??      00h
3fc93858 7c      ??      7Ch I
3fc93859 b7      ??      B7h
3fc9385a d7      ??      D7h
3fc9385b 7c      ??      7Ch I
3fc9385c d7      ??      D7h
3fc9385d 07      ??      07h
3fc9385e 77      ??      77h w
3fc9385f c7      ??      C7h
3fc93860 01      ??      01h
3fc93861 00      ??      00h
3fc93862 00      ??      00h
3fc93863 00      ??      00h

```

Cette étape de calibration était indispensable pour valider que les clés ne sont pas obfuscées et pour obtenir l'adresse précise à inspecter.

3.3 Exfiltration des secrets du binôme

Fort de la connaissance acquise sur mon propre firmware, j'ai appliqué le même schéma d'analyse au firmware extrait de la carte de mon binôme.

En naviguant directement à l'adresse mémoire identifiée lors de l'étape précédente, j'ai pu lire instantanément la clé secrète de la victime, sans avoir besoin de rechercher une valeur spécifique à l'aveugle. L'analyse a révélé la clé suivante :

- **Clé exfiltrée (devEui) :** 0x69, 0x69, 0x69, 0x69, 0x69, 0x69, 0x69, 0x69.

Cette démonstration confirme la vulnérabilité critique du stockage non sécurisé : la structure du binaire étant prédictible, la connaissance de l'emplacement mémoire suffit à compromettre n'importe quel dispositif utilisant le même firmware.

3.4 Usurpation d'identité (Spoofing)

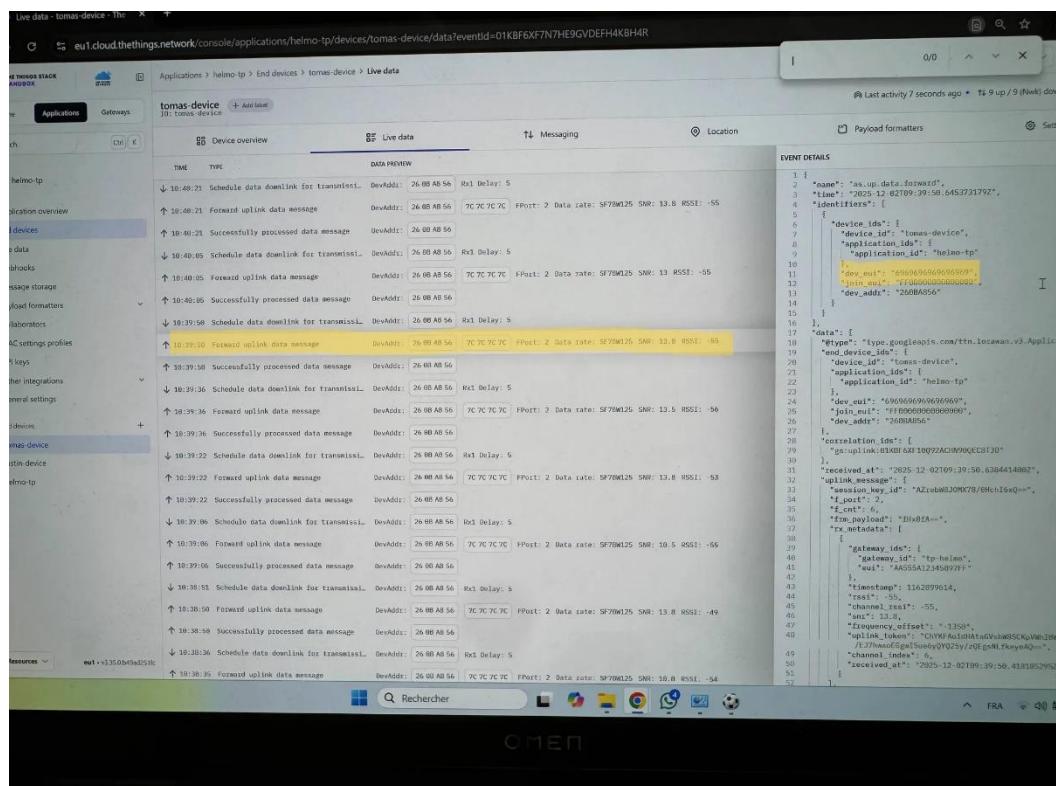
La dernière étape consistait à prouver la compromission en clonant le dispositif. Nous avons modifié notre propre code source pour y injecter le DevEui volé (0x69...) à la place du nôtre.

```
Ma clé défini au début du labo :  
uint8_t devEui[] = { 0x7C, 0xB7, 0xD7, 0x7C, 0xD7, 0x07, 0x77, 0xC7 }  
  
Il faut modifiée ma clé avec cette valeur (clé du binôme):  
uint8_t devEui[] = { 0x69, 0x69, 0x69, 0x69, 0x69, 0x69, 0x69, 0x69 }  
  
Cette clé ne doit pas changer :  
uint8_t appEui[] = { 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }
```

Afin de distinguer nos paquets de ceux du binôme légitime sur la passerelle, nous avons également modifié la charge utile (*payload*) du message pour envoyer une suite d'octets spécifique (0x7C).

```
static void prepareTxFrame( uint8_t port )
{
    /*appData size is LORAWAN_APP_DATA_MAX_SIZE which is defined in "commissioning.h".
    *appDataSize max value is LORAWAN_APP_DATA_MAX_SIZE.
    *if enabled AT, don't modify LORAWAN_APP_DATA_MAX_SIZE, it may cause system hanging or failure.
    *if disabled AT, LORAWAN_APP_DATA_MAX_SIZE can be modified, the max value is reference to lorawan
region and SF.
    *for example, if use REGION_CN470,
    *the max value for different DR can be found in MaxPayloadOfDatarateCN470 refer to DataratesCN470
and BandwidthsCN470 in "RegionCN470.h".
*/
    appDataSize = 4;
    appData[0] = 0x7C;
    appData[1] = 0x7C;
    appData[2] = 0x7C;
    appData[3] = 0x7C;
}
```

Après téléversement de ce firmware malveillant, le dispositif s'est connecté au réseau en se faisant passer pour celui du binôme. La console de la passerelle LoRaWAN a confirmé la réception des données signées avec l'identité de la victime, validant ainsi la réussite de l'attaque par clonage.



4. Conclusion et Remédiations

Ce laboratoire a permis de parcourir l'intégralité de la chaîne d'attaque sur un objet connecté, depuis l'accès physique jusqu'à l'usurpation d'identité numérique sur le réseau LoRaWAN.

4.1 Bilan de l'attaque

La réussite de la preuve de concept (PoC) démontre qu'un dispositif IoT standard, tel que l'ESP32 Heltec utilisé dans sa configuration par défaut, ne résiste pas à une analyse locale. L'accès au port USB-UART a suffi pour :

1. Contourner les mécanismes de sécurité périphériques.
2. Extraire l'intégralité de la propriété intellectuelle et des données sensibles (clés) contenues dans la mémoire flash.
3. Cloner parfaitement le comportement de l'objet légitime.

Ce résultat valide concrètement les principes théoriques vus en cours : la sécurité d'un système ne peut reposer uniquement sur la robustesse des protocoles cryptographiques de communication (AES-128 du LoRaWAN). Si l'extrémité (le *End-Device*) est physiquement compromise et mal sécurisée, la chaîne de confiance est rompue.

4.2 Contre-mesures et Sécurisation

Pour prévenir ce type d'attaque par extraction de firmware (*Read-back attack*), plusieurs mécanismes de défense doivent être implémentés lors de la phase d'industrialisation :

- **Désactivation des interfaces de débogage (eFuses)** : L'ESP32 dispose de fusibles électroniques programmables (eFuses) qui permettent de désactiver matériellement les interfaces JTAG et UART pour le téléchargement ou la lecture. Une fois "brûlés", ces fusibles empêchent définitivement l'utilisation d'outils comme esptool pour dumper la mémoire.
- **Chiffrement de la Flash (Flash Encryption)** : Cette fonctionnalité permet de chiffrer le contenu de la mémoire flash externe. La clé de déchiffrement est stockée dans les eFuses internes du processeur et n'est jamais accessible logiciellement. Même si un attaquant parvenait à extraire physiquement la puce mémoire, les données récupérées seraient inexploitables sans la clé matérielle.
- **Utilisation d'un Secure Element (SE)** : Pour une sécurité optimale, les clés critiques (comme l'AppKey ou le DevEui) ne devraient pas être stockées dans la mémoire flash générale, mais dans un composant cryptographique dédié (Secure Element) résistant aux attaques physiques et logiques.

En conclusion, ce laboratoire souligne l'importance de l'approche *Security by Design*. La protection contre l'accès physique n'est pas une option mais une nécessité critique pour tout déploiement IoT opérant dans un environnement non contrôlé.

5. Bibliographie

Arduino IDE : Environnement de développement intégré utilisé pour la programmation du microcontrôleur.

<https://www.arduino.cc/en/software/>

esptool.py : Utilitaire officiel d'Espressif basé sur Python pour communiquer avec le bootloader ROM des puces ESP32 (utilisé pour l'extraction du firmware).

pip install esptool

Ghidra : Suite d'outils de rétro-ingénierie logicielle (SRE) développée par la NSA, utilisée pour l'analyse statique du binaire.

<https://github.com/NationalSecurityAgency/ghidra>

Ghidra ESP32 Flash Image Loader : Extension pour Ghidra permettant de charger et mapper correctement les images flash des microcontrôleurs ESP32 (architecture Xtensa).

<https://github.com/saibotk/ghidra-esp32-flash-loader/releases>

Silicon Labs CP210x Drivers : Pilotes pour le pont USB vers UART (VCP) nécessaires à la détection de la carte Heltec sur le port série.

<https://www.silabs.com/software-and-tools/usb-to-uart-bridge-vcp-drivers>

Heltec ESP32 Series Documentation : Documentation technique et guide de démarrage rapide pour la carte Heltec WiFi LoRa 32 (V3).

https://docs.heltec.org/en/node/esp32/esp32_general_docs/quick_start.html

Heltec ESP32 Arduino Core : Paquet de gestion de carte pour l'IDE Arduino (Json).

https://resource.heltec.cn/download/package_heltec_esp32_index.json

Bours, J. (2023-2024). *Cybersécurité IoT - Module 2 : Vulnérabilité IoT*. Technifutur.

Bours, J. (2023-2024). *Cybersécurité IoT - Module 4 : Lora*. Technifutur.