Université Bretagne Sud

Master 1 Ingénierie de Systèmes Complexes

Spécialité Cybersécurité des Systèmes Embarqués

Promotion 2019-2020

Étude du protocole BLE

Stage Master 1

Gidon Rémi

Avril/Juin 2020

Sommaire

Tableaux

Figures

Dans le cadre du master CSSE nous étudions l'internet des objets (IoT) et leurs aspects sécurité. Le protocole réseau sans fil Bluetooth Low Energy (BLE) permet une consommation réduite pour les objets fonctionnant sur batterie, visant notamment les objets connectés. Aujourd'hui integré dans la plupart des appareils de bureautique, il est rapidement devenu populaire dans l'internet des objets.

La première itération du BLE (sortie en 2011) ne répond plus aux exigences de sécurité contemporaine et même si le protocole à su évoluer depuis pour répondre à ces besoins, beaucoup d'appareils utilisent encore la version originale n'intégrant pas ces mécanismes. Ce sont pour la plupart des appareils conçus pour fonctionner sur batterie et communiquer en point à point. On va retrouver les capteurs corporels pour santé ou fitness mais également des mécanismes plus sensibles tels des cadenas ou serrures. Les communications (incluant parfois des données personnelles) peuvent êtres interceptées, voir modifiées pour permettre des actions aux dépends de l'utilisateur.

1 Objets connectés

Avec l'explosion de l'internet de objets au cours de ces dernières années, toute une floppée d'objet du quotidien ont étés augmentés pour permettre la communication avec d'autres systèmes informatique dont nos smartphones ou encore des serveurs distants (via notre Wi-Fi). Ces objets dits intelligents étendent leur équivalent mécanique en intégrant des composants éléctroniques, permettant notamment le contrôle à distance.
Face à l'engouement du public, les constructeurs s'efforcent de proposer des objets toujours plus *intelligents* et connectés, souvent au détriment de la sécurité. Ces améliorations engendrent une augmentation de la surface d'attaque: les objets connectés sont confrontés aux mêmes challenges que ceux des systèmes informatiques traditionnels en plus de leur fonction primaire.

1.1 Architecture

Les objets connectés ont commencés par proposer des communications avec nos smartphones, notre routeur Wi-Fi ou notre ordinateur. Celles-ci permettent d'utiliser ces objets comme télécommande de contrôle (via une application dédiée la plupart du temps) ainsi que communiquer aux services distants du constructeur en s'appuyant ces fonctionnalités (Wi-Fi, données mobiles).
Ces architectures point a point connectent un appareil directement à un controleur (*smartphone*, *PC*) duquel il est dépendant pour accéder aux services distants (si il y a). On retrouve cette architecture dans les appareils autonomes qui réalisent une fonction d'augmentation sur un produit existant (comme les cadenas connectés). Ces appareils n'ont souvent pas besoin d'un service distant puisqu'ils proposent un modèle d'interactions simple et local avec un seul utilisateur.

Les objets connectés ont rapidement étés utilisés pour la mise en place des réseaux de capteurs. Cette utilisation à été largement introduite pour les particuliers avec la domotique. Les interactions avec ces réseaux ont d'abors été locales, au sein du domicile, puis globales pour permettre un controle depuis n'importe quel emplacement. Cette tendance pousse les constructeurs à exposer leurs objets connectés au réseau mondial: certains ont optés pour incorporer des puces Wi-Fi directement dans leurs objets quant a d'autres ont proposés une solution plus long terme en mettant un place une passerelle (appelée *hub*) gérant les interactions avec le monde extérieur.
L'architecture *hub* est aujourd'hui vastement utilisée dans la domotique. Un appareil dédié est considéré comme *hub* par lequel les capteurs, beaucoup plus simples, communiquent. Des protocoles spécialisés ont fait leur apparition comme *Zigbee*[1], *Z-Wave*[2], *ANT+*[3] ou encore *Thread*[4].

1.2 Protocoles

Comme évoqué précédemment, beaucoup d'objets connectés ont profité des protocoles intégrés dans les appareils utilisés comme controleur. Cela englobe le *Wi-Fi*, le *Bluetooth* et dernièrement le *NFC*. De ces trois, le *Bluetooth* à largement pris le dessus car plus adapté avec son standard *Low Energy*. Concu en tant que *WPAN* (réseau sans fil personnel) il permet de communiquer dans un rayon de 10 mètres, suffisant pour les interactions

[1] https://zigbeealliance.org/
[2] https://www.z-wave.com/
[3] https://www.thisisant.com/
[4] https://www.threadgroup.org/

locales. Le *NFC* est assez récent, il à été conçu pour les interaction proches (une dizaine de centimètres) et intentionnelles comme les paiements. Enfin le *Wi-Fi* aurait du être le plus populaire, puisque chaque foyer dispose d'une box internet, mais sa consommation est telle qu'un objet sur batterie ne tient que quelques heures au maximum (voir les ordinateurs portables, disposants pourtant de larges batteries). Il n'est pas adapté au besoin puisqu'il permet de hauts debits avec faible latence pour une consommation élevée la ou les objets connectés utilisent de faibles débits sur des transmission occasionnelles pour économiser leurs batteries.

Avec la démocratisation des objets connectés de nouveaux protocoles spécialisés ont fait leur apparition. Même si le *BLE* (*Bluetooth Low Energy*) s'adapte pour répondre aux besoins de ce marché, il n'a été conçu pour les objets connectés mais les objets intelligents en permettant un contrôle par l'utilisateur.
Beaucoup de grands constructeurs (notamment Google et Apple) ont developpés leur standard, le vantant et l'imposant avec leurs produits et architectures propriétaires. *Apple Home* utilise *Darwin* (iOS, macOS) comme contrôleur ainsi que son propre protocole (*HAP*) pour ses objets connectés. Google a mis en place le protocole *Thread*, interopérable avec *Google Home* (*hub*) et *Android* (*contrôleur*). Bien avant, des constructeurs specialisés ont developpés *Zigbee*, *Z-Wave* ou encore *ANT+*.
Beaucoup de protocoles se battent pour avoir accès à un marché juteux encore instable car en plein développement. Cependant beaucoup d'objets connectés n'ont pas besoin de l'interconnexion qu'apportent ces protocoles et le *BLE* est loin d'etre desuète, étant continuellement retravaillé et proposant des améliorations intéressantes dans ce milieu.

2 Bluetooth Low energy

Le protocole a principalement été designé par Nokia pour répondre au besoin d'un protocole sans fil peu gourmand en énergie permettant la communication avec les péripheriques personnels (téléphone portable, montre, casque audio). Nommé *Wibree*, il a été intégré au standard Bluetooth sous le nom *Low Energy*.
Le Bluetooth ne comprend pas seulement un protocole mais une multitude d'entre eux (BR, EDR, HS) qui ont en commun de permettre la communication (et l'échange de données) sans fil avec des périphériques personnels. Ils font partit des protocoles *WPAN* et leur distance d'émission varie de quelques mètres jusqu'à 30 mètres.
La spécification Bluetooth 4.0, sortie en 2011, intégre le protocole LE (*Low Energy*) et permet au Bluetooth de toucher le marché des systèmes embarqués fonctionnants sur batterie.

2.1 Différences

Les autres protocoles du Bluetooth sont principalement connus et utilisés pour le transfert de contenu multimédia, que ce soit des fichiers entre ordinateurs comme de la musique avec un casque ou encore une voiture. Ils fonctionnement avec une connexion continue et un transfert en mode flux.
Le BLE, visant à reduire la consommation d'énergie, n'établie pas de connexion continue. L'appareil reste la plupart du temps en mode veille, pouvant émettre des annonces, dans l'attente d'une connexion qui aura pour effet d'arrêter la transmission d'annonces. Pour chaque requête reçu, une réponse pourra être renvoyée ou une notification mise en place périodiquement.
Les appareils BLE et Bluetooth BR/EDR ne sont pas compatibles, n'utilisant pas les mêmes technologies/protocoles et répondant à des besoin différents (voir tbl. 1).

Tableau 1: Cas d'utilisation et protocole Bluetooth adapté

Besoin	Flux données	Transmisson données	Localisation	Reseau capteurs
Appareils	ordinateur, smartphone, casque, enceinte, voiture	accessoires bureautique ou fitness, equipement medical	beacon, IPS, inventaire	automatisation, surveillance, domotique
Topologie	point à point	point à point	diffusion (1 à N)	mesh (N à N)
Technologie	Bluetooth BR/EDR	Bluetooth LE	Bluetooth LE	Bluetooth LE

2.2 Protocole

Pour permettre une interopérabilité maximale entre les appareils BLE, le standard défini 4 profils en fonction du role de l'appareil: `Peripheral`, `Central`, `Broadcaster`, `Observer`. Chaque appareil se conformant au standard ne doit implémenter qu'un seul de ces rôles à

la fois.
Le *Broadcaster* ne communique qu'avec des annonces, on ne peut pas s'y connecter. Ce mode est tres populaire pour les beacons. L'*Observer* est sont opposé, il ne fait qu'écouter les annonces, n'établiera jamais de connexion.
Le *Peripheral* et le *Central* forment la seconde pair et permettent la mise en place d'une architecture client-serveur. Le *Peripheral* joue le role du serveur et est dit *esclave* du *Central* qui endosse le rôle du client et *maître*.
Le *peripheral* transmet des annonces jusqu'à recevoir une connexion d'un *central*, après quoi il arrête de s'annoncer car ne peut être connecté qu'à un *central* à la fois. Le *central* écoute les annonces de *peripherals* pour s'y connecter, puis interroge ses services via les requêtes *ATT/GATT* (*Generic Attribute*).

Couche physique

Le BLE opère dans la bande ISM 2.4GHz tout comme le Wi-Fi. Contrairement aux canaux Wi-Fi de 20MHz, le BLE découpe le spectre en 40 canaux de de 2MHz (plage de 2400 à 2480MHz).
Le protocole met en place le *saut de fréquence*, consistant à changer de canal d'émission tout les laps de temps donné, pour réduire le risque de bruit sur les fréquences utilisées (la bande ISM 2.4Ghz étant libre d'utilisation).
Sur les 40 canaux que compose le spectre, 3 sont utilisés pour la transmission d'annonce. Ils sont choisit pour ne pas interferer avec les canaux Wi-Fi car les deux protocoles sont amenés à coexister (voir fig. 1).
Les 37 autres canaux sont utilisés pour les connexions. Chaque connexion va utiliser un sous-ensemble des 37 canaux (appelé carte des canaux) pour éviter les interférences avec les autres protocoles et connexions BLE. Un seul canal transmet des données à la fois mais tous les canaux de la carte sont utilisés pour le saut de fréquences.

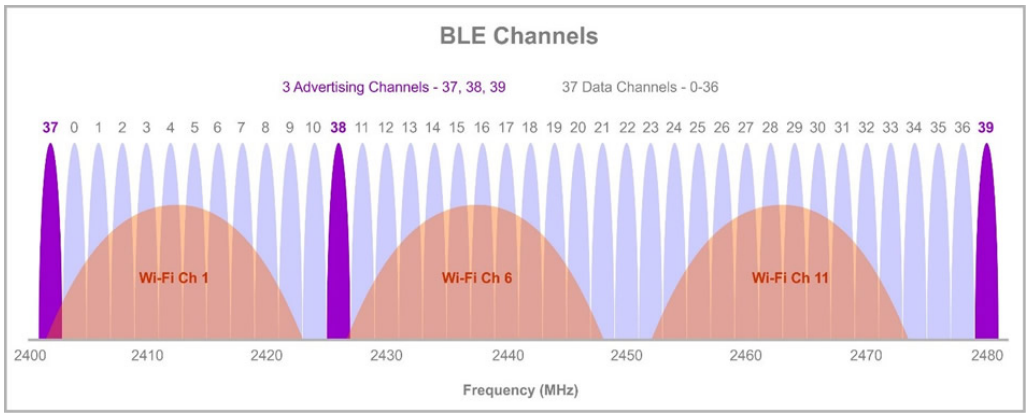

Figure 1: Répartition du spectre BLE en canaux[5]

Couche logique

1. Annonces

Le *peripheral* indique sa présence avec des annonces émises périodiquement. Ces annonces contiennent sont addresse Bluetooth (permettant une connexion) et des données qui

[5]https://www.accton.com/Technology-Brief/ble-beacons-and-location-based-services/

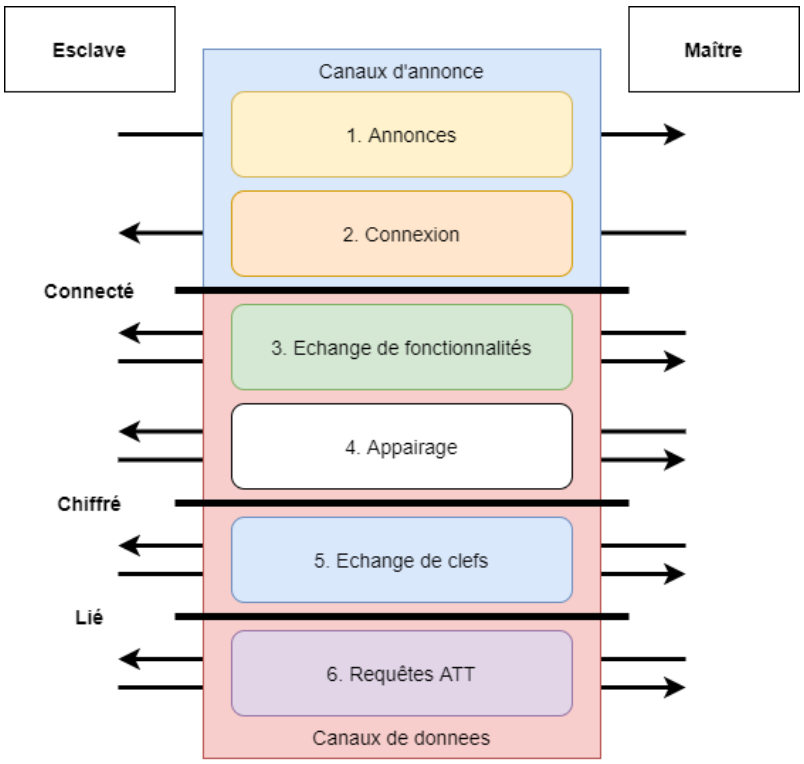

Figure 2: Étapes d'un échange BLE

consituent un profile (appelé *GAP*[6]). Ces données permettent aux *centrals* de savoir si il est capable de réaliser les fonctionnalités recherchées.
La spécification Bluetooth définit des profiles type pour des applications communes dans les appareils BLE[7]. Cela inclus par exemple les capteurs corporels pour le sport, les capteurs médicaux de surveillance (pour les diabetiques notamment), la domotique (termomètres, lampes), etc.

Dans un environnement BLE, les *centrals* ne peuvent pas reconnaître leurs *peripherals* à part avec une addresse Bluetooth fixe, mécanisme de moins en moins utilisé car vulnérable à l'usurpation. Les *peripherals* génèrent donc des adresses aléatoires et l'identification se fait via les données du *GAP* contenues dans l'annonce. Ce mécanisme permet à n'importe quel *central* de s'appairer à n'importe quel *peripheral* proposant le profil recherché.
Par exemple, une application de smartphone BLE pouvant gérer la température pourrait s'appairer et utiliser n'importe quel appareil BLE qui implémente le profil standardisé pour les termomètres dans le *GAP*.
Les profils ne sont certes pas exhaustifs mais permettent une intégration fonctionnelle avec un maximum d'appreils et prévoient un moyen d'intégrer des données propriétaires non standardisées[8].

2. Connexion

Lorsqu'un *central* reçois une annonce d'un *peripheral* auquel il souhaite se connecter, il

[6]https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile/
[7]https://www.bluetooth.com/specifications/gatt/services/
[8]https://www.silabs.com/community/wireless/bluetooth/knowledge-base.entry.html/2017/02/10/bluetooth_advertisin-hGsf

lui envoit une intention de connexion sur les canaux d'annonce. Ce message contient tout les paramètres communs pour établir une connexion sur les canaux de données: carte des canaux utilisés, temps entre chaque saut de fréquence, nombre de canaux sautés par saut, adresse unique de la connexion (appelée *Access Adress*).
Ce message (nommé *CONNECT_REQ*) est crucial lors d'attaques car il permet la synchronisation avec une connexion pour l'écoute passive et est donc jugé sensible puisque transmit sur les canaux d'annonces avant la mise en place du chiffrement.

3. Capacités

Le BLE voulant garder une interopérabilité maximale entre les appareils mais tout les appareils ne disposant pas des mêmes fonctionnalités embarqués, il est définit plusieurs méthodes d'appairage en fonction des capacités disponibles sur les deux appareils.
Chaque appareil va transmettre ses capacités à l'autre ainsi que ses exigences sur la connexion à établir. Les capacités sont déduites des fonctionnalités présentes physiquement sur l'appareil et les exigences dépend de la version du procole actuellement supportée par celui-ci.
Les exigences comprennent la protection aux attaques *MITM* par l'authentification de l'appairage, l'établissement d'une connexion sécurisée (*LE secure connection*), la mise en place d'une session (*Bonding*) pour une reconnexion future ainsi que l'utilisation d'un canal autre que le BLE (comme le *NFC*) pour la transmission de secrets menant au chiffrement (*Out Of Band* ou *OOB*).

Tableau 2: Capacités d'entrée possibles[9]

Capacité	Description
No input	pas la capacité d'indiquer *oui* ou *non*
Yes/No	mécanisme permettant d'indiquer *oui* ou *non*
Keyboard	claver numérique avec mécanisme *oui*/*non*

Tableau 3: Capacités de sortie possible

Capacité	Description
No output	pas la capacité de communiquer ou afficher un nombre
Numeric Output	peut communiquer ou afficher un nombre

Tableau 4: Capacité d'entrées/sorties de l'appareil

	No output	Numeric output
No input	NoInputNoOutput	DisplayOnly
Yes/No	NoInputNoOutput	DisplayYesNo
Keyboard	KeyboardOnly	KeyboardDisplay

4. Appairage

[9] https://www.bluetooth.com/blog/bluetooth-pairing-part-1-pairing-feature-exchange/

En fonction des capacités et des exigences émises par chacun des appareils, une méthode d'appairage est sélectionnée (voir tbl. 5).

Tableau 5: Méthode d'appairage utilisée en fonction des capacités échangées[10]

	DisplayOnly	**DisplayYesNo**	**KbdOnly**	**NoIO**	**KbdDisplay**
DisplayOnly	JustWorks	JustWorks	PassKey	JustWorks	PassKey
DisplayYesNo	JustWorks	JustWorks	PassKey	JustWorks	PassKey
KbdOnly	PassKey	PassKey	PassKey	JustWorks	PassKey
NoIO	JustWorks	JustWorks	JustWorks	JustWorks	JustWorks
KbdDisplay	PassKey	PassKey	PassKey	JustWorks	PassKey

Je m'intéresse principalement à la methode *JustWorks*. C'est la methode par défaut lorsque deux appareils ne disposent pas des capacités nécessaires pour une autre. Elle est notamment beacoup présente pour les objets connectés puisque n'intègrant pas de mécanismes pour un appairage plus complexe.
Passkey permet d'authentifier l'appairage pour se protéger des usurpations d'identité (*MITM*) puisque partageant un secret via l'utilisateur (ou un autre canal dans le cas du *OOB*). *JustWorks* ne permet pas d'authentifier les appareils et le chiffrement est moins robuste que les autres méthodes mais permet tout de même d'établir une communication chiffrée.

La méthode d'appairage choisie permet de transmettre un des matériaux cryptographique: la clef temporaire (ou *Temporary Key*). Cette phase est plus ou moins sensible à l'écoute passive en fonction de la méthode d'appairage et des exigences émises lors de l'échange des capacités.
JustWorks avec connexion BLE 4.0 (dite *legacy*) est le mode le plus sensible puisque la clef temporaire est tout simplement zéro, ne disposant pas de moyen de transmettre une donnée par autre voie, elle peut donc etre trouvee rapidement par *brute-force*.
La connexion *LE secure*, introduite à partir de la version 4.2, utilise l'algorithme Diffie-Hellman sur courbes elliptiques (*ECCDH*) pour l'echange des materiaux cryptographiques et est donc resistante a l'ecoute passive (*eavesdropping*) mais toujours vulnerables a l'usurptation d'identite (*MITM*) avec *JustWorks*.

5. Echange de clefs

L'etablissement du chiffrement de la connexion est ensuite realisé par derivation a partir d'une premiere clef temporaire transmise via la methode d'appairage choisie et des autres parametres cryptographiques echanges via le protocole BLE. La clef obtenue est dite court terme (*Short Term Key*) car elle ne sera utilisee que pour cette connexion et devra etre re-generee a chaque nouvelle connexion.
Il est cependant possible de mettre en place une session en stockant une clef partagée dite long term (*Long Term Key*) si cela a ete exigé lors de l'echange des capacités. La clef long terme (*LTK* pour *Long Term Key*) est stockée et associée à l'appareil communiquant pour rétablir une connexion future sans avoir à refaire une phase d'appairage.

A partir de la comprehension actuelle du protocole BLE et du fonctionnement de l'appairage, il semble recommandé de mettre en place une connexion securisee des que possible. Il est

[10] https://www.bluetooth.com/blog/bluetooth-pairing-part-2-key-generation-methods/

egalement necessaire d'eviter la methode *JustWorks* au maximum.
Cependant, il est assez simple de forger un echange de capacités pour retrograder la connexion en *legacy* et forcer *JustWorks* via les capacités echangees. C'est pourquoi certains appareils attendent des capacites et exigences minimales pour etablir une connexion, sans quoi celle-ci est avortee. C'est notamment le cas d'appareils proprietaires concus pour fonctionner ensemble.

6. Requêtes

Les échanges sont realises sur la base d'une architecture client-serveur. Le *central* (client) interroge le *peripheral* (serveur) avec le protocole *ATT* (*ATTribute Protocole*). Chaque requete mene soit a une reponse du serveur soit a la mise en place d'une notification lors d'un evenemment (valeur changée ou disponible).
Les requetes et reponses possibles sont standardisées sous le *GATT* (*Generic ATTributes*) pour permettre une interoperabilité maximale entre les appareils (comme pour le *GAP*). *GATT* et *GAP* partagent les memes profiles, seul la structure change. Le serveur *GATT* peut etre interrogé pour etablir une liste exhaustive de toutes les fonctionnalites d'un appareil la ou le *GAP* choisit ce que contient l'annonce mais est limite par la taille du paquet (31 octets).

GAP

Dans le cas des *Peripherals* et *Centrals*, le *GAP* est principalement utilisé pour etablir un profil de l'esclave permettant la decision de connexion de la part du maitre.
Pour les *Boardcasters* et *Observers* il permet la communication unidirectionnelle (*Broadcaster* vers *Observer*) via les annonces, ceux-ci utilisant la diffusion plutot qu'une connexion point a point. On retrouve cette utilisation pour les beacons publicitaires ou de localisation interieur.

GATT

Pour l'echange de données lors de connexion point à point, le *GATT* est utilisé en mode client-serveur. L'architecture du serveur *GATT* est en entonnoir, la plus haute couche s'appelle un *service*, il encapsule des *caracteristiques*, chacune contenant une *valeur* et un ou plusieurs *descripteurs* fournissants des informations additionnelles sur la valeur (voir fig. 3).
A chacune de ses couches (service, caracteristique, valeur, descripteur) est attribué un identifiant unique appelé *handle*. La plage des indentifiant est partagée entre toutes les couches donc si un service a l'identifiant `0x01` aucun autre service/caracteristique/attribut/descripteur ne peut l'utiliser.

Un service correspond generalement a un profil (standardise ou non) comme par exemple un termometre. Ce service exposerait des caracteristiques comme la temperature, l'humidite ou autres. Chacune de ces caracateristique contient la valeur (donnée brute) et les descripteurs peuvent indiquer l'unité ou encore un facteur ou formule pour convertir la valeur donnée en resultat exploitable.

A moins de connaitre exactement l'appareil et de l'interroger en mode aveugle via les identifiants (ce qui peut etre le cas entre des appareils proprietaires), il faut proceder par etape en decouvrant d'abors les services disponibles, puis chaque caracteristique par service et enfin les attributs de celles-ci.

Pour proceder a cette decouverte d'un appareil, le protocole *ATT* dispose d'un type de requete par couche a interroger (voir fig. 3). Une fois le service voulu trouvé (ou la cartographie totale de l'appareil realisée), on peut lire, ecrire ou souscrire a des attributs directement par *handle*. Le *GATT* met en place un systeme de droits par attribut pour proteger la lecteur, l'ecriture et la souscription par le client.

Le *GATT* définit egalemet des services standardisés appelé primaire et secondaires censés etres present sur tout les appareils BLE afin de connaitre les fonctionnalités standardisés (service primaire) et proprietaires (service secondaire) de l'appareil. Comme les *handle* sont definies arbitrairement par le serveur *GATT*, les profils standards et leurs services/caracteristiques sont identifiés par un *UUID* standardisé unique a travers tout les appareils BLE.[11]

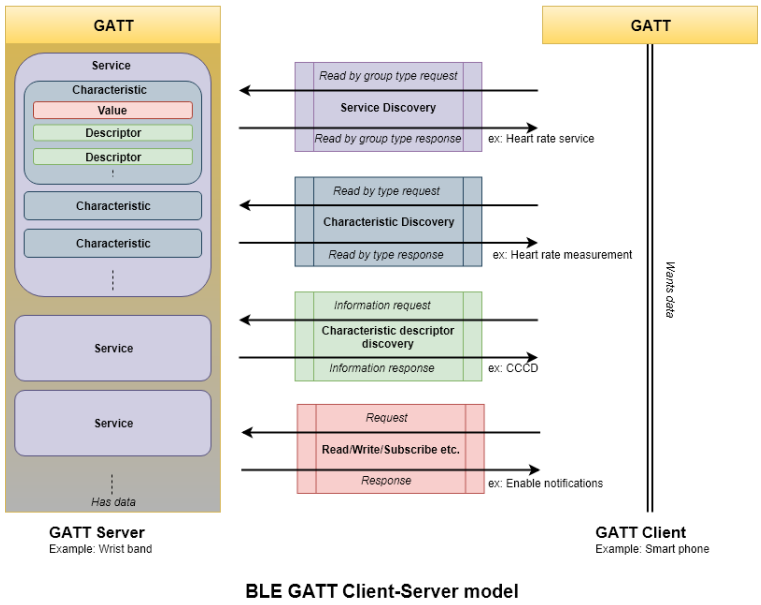

Figure 3: Client et serveur GATT[12]

2.3 Evolution

Depuis sa premiere iteration en 2011 dans la version `4.0` des specifications Bluetooth le BLE a evoluer pour integrer des mesures de securite avec l'ajout des connexions securisees LE en `4.2` puis la diversification des topologies avec l'introduction du *mesh* pour les reseaux de capteurs en `5.0` et dernierement l'amelioration de la localisation interieur (*Indoor Positionning System*) pour une precision de l'ordre du centimetre grace aux systemes angle d'arrivée et de depart (*AOA/AOD*).

[11] https://www.bluetooth.com/specifications/gatt/services/

[12] https://fr.mathworks.com/help/comm/examples/modeling-of-ble-devices-with-heart-rate-profile.html

3 Preuve de concept

Sujet: Étudier puis mettre en place des attaques sur le protocole *Bluetooth Low Energy* (Bluetooth Smart)

Le but est d'exposer puis abuser des failles dans le protocole BLE 4.0 (première version). Ces failles sont connues et ont pour la plupart été corrigées dans les version ultérieures du protocole (aujourd'hui en version 5.1). Neanmoins cela permet d'etudier et comprendre les mecanismes du BLE depuis sa creation, puis voir les alternatives qui ont etes proposees pour mitigees ces attaques.
Je ne cible que les appareils supportant l'appairage en BLE 4.0 (dit *legacy*) avec méthode d'appairage *JustWorks*. C'est le niveau de sécurité minimal prit en charge par le protocole et très répandue dans les appareils connectés. La majeure partie des appareils connectés sont simples, ne realisant qu'une fonction d'augmentation, ne disposant pas de clavier ou d'écran et ne permettent pas ainsi l'utilisation de méthode d'appairage autre que *JustWorks*.
Les mecanisme proposees a partir de la version 4.2 du BLE sont beaucoup plus robustes. Ils apportent les connexion securisee (LE Secure Connections ou LESC) se basant sur Diffie Hellmann pour l'echange de clefs ainsi qu'une nouvelle methode d'appairage authentifiée (Comparaison numerique).

Meme si le BLE a toujours proposé des mesures de securité, la majorite des constructeurs ne les utilisent pas et mettent en place des chiffrements au niveau de la couche application (la ou le BLE chiffre depuis la couche lien).
Ces mesures de securité proprietaires sont souvent basees sur des algorithmes reconnus comme AES et des methodes comme le *challenge-response* pour authentifier un appareil. Une clef unique est integrée dans chaque appareil, celle-ci sera soit distribuee au proprietaire de l'appareil lors de la creation du compte ou le telechargement de l'application associee, soit gardee par le constructeur qui transmettra directement les commandes de l'utilisateur a l'appareil (via l'application ou directement si l'appareil est connecté au reseau mondial).
Ces mecanismes exposent cependant beaucoup plus d'informations que le chiffrement BLE depuis la couche lien. Les requetes ATT et GATT transittent en clair et pour pallier a la fuite d'informations les constructeurs evitent les requetes standardisee danas le BLE et preferent utiliser des protocoles personnalisés dans la couche application, celle-ci étant chiffrée.
Ces chiffrements proprietaires sur la couche application sont hors de portee de mon sujet mais ont fait couler beaucoup d'encre et plusieurs presentation et leurs *whitepaper* sont disponibles.

Maintenant il s'avere que beaucoup d'appareils autonomes simples ne mettent en place aucune mesure de securite, qu'elle soit standardisee ou proprietaire, car les donnees qui transittent ne sont pas jugees sensibles. C'est notamment le cas des objets domotiques autonomes comme les telecommandes pour lampes dites connectées.

3.1 Travail demandé

Mettre en place un outil basé sur un framework offsenf permettant de répertorier et faciliter l'analyse des appareils et connexion BLE alentours. Cet outil est facilement portable sur diverses cartes de developpement comme la raspberry Pi car conteneurisé avec *Docker* et se basant sur du materiel USB pour l'etude du protocole (dongle et sniffer).
3 ports USB suffisent pour permettre de conduire toutes les attaques proposées par le framwork offensif utilisé (Mirage): 2 dongles USB BLE 4.0 et une carte BBC Micro:bit.

Le projet suppose la mise en place de 3 attaques dont une nouvelle non integrée a Mirage: - Inventaire des appareils et connexions a proximité + localisation des appareils (*scan*) - Usurpation et mise en place d'un *Man In The Middle* sur un appareil selectionné (*spoofing*) - Synchronisation puis detournement par brouillage d'une connexion precedement identifiée (*hijacking*)

3.2 Architecture

serveur flask + websockets application js (hyperapp) + socketio framework offensif Mirage (python) + bindings custom pour communication API et non CLI

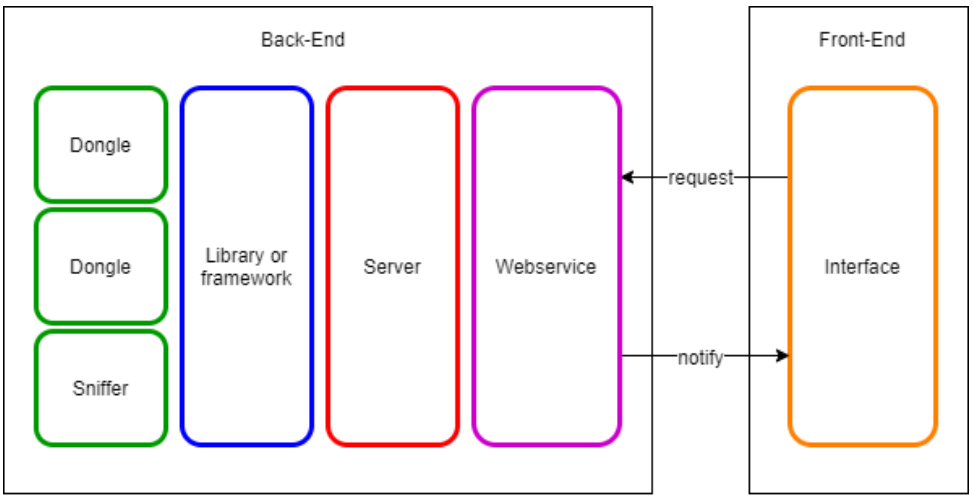

Figure 4: Architecture du système

3.3 Interface

But d'augmenter Mirage avec front-end pour demo et conduire attaques *type*

Technologie JS *Elm* (react trop lourd mais meme principe)

On retrouve la carte des appareils et connexions identifiés avec leur distance et position estimée par rapport au système (voir fig. **??**: zone rouge *Scan*).
Pour chaque cible (appareil ou connexion), des attaques sont disponibles: - Récupération du profil ou modification des transimissions par usurpation pour un appareil BLE emettant des annonces (zone bleue *Devices*). - Déconnexion des appareils ou interception des communications entre deux appareils appairés (zone bleue *Connections*).

TODO Screen GUI HTML

4 Étude de l'existant

4.1 Outils

Les outils offensifs sur le protocole BLE permettent de recuperer, analyser et modifier les echanges entre appareils permettant de realiser des audits de securite ou mettre en place des attaques exposant des vulnerabilites. L'analyse du traffic sans fil BLE demande une antenne couvrant la bande utilisee par les 40 canaux du protocole ainsi qu'un systeme assez rapide pour scanner puis suivre les communications lors des sauts de frequence. Les radio-logiciels (*SDR*) ne sont donc pour la plupart pas adaptés car trop lents ou trop cher pour les fonctionnalités voulues: des outils specialises dans l'analyse et l'attaque du BLE sont disponibles pour une fraction du prix.

Le premier outil, utilisé dans tout nos appareils BLE, est la puce intégrée pour les communications BLE. La recuperation (*sniffing*) et analyse ou modification d'un traffic sans fil etant interdit, ces puces utilisent un *firmware* proprietaire conforme aux roles determinés dans les specifications BLE. Même si il reste possible d'analyser le traffic (notamment en utilisant *Wireshark*[13]) entre la puce et un appareil BLE, il n'est pas possible d'etendre les capacites de celle-ci sans modifier le *firmware*.
Tout les appareils ne disposant pas d'une puce BLE dédiée, les constructeurs ont developpes des *dongles* integrant ces puces et permettant de communiquer avec tout appareil USB via une interface nommée *HCI* (*Host-Controller Interface*). Allié aux outils standard du protocole BLE comme *BlueZ*, la pile protocolaire BLE du noyau Linux, ces *dongles* permettent de découvrir les appareils a proximité et d'endosser le role de *peripheral* ou *central* pour etablir une communication avec n'importe quel autre appareil BLE. Les utilitaires *hcitool*, *hciconfig* et *gatttool* de *BlueZ* permettent par exemple de manipuler les annonces et extraire le profil *GATT* d'un appareil BLE. Meme si certains de ces *dongles* proposent des fonctionnalites interessantes comme le changement d'adresse Bluetooth (*Bluetooth Address* ou *BD*), ils n'ont pas été conçus dans une optique de sécurité, et sont peu flexibles pour un usage offensif.

La plupart des attaques sur le protocole BLE requierts un moyen d'intercepter le traffic. Les *dongles* et *puces* embarquants des firmware ne permettant pas cette fonctionnalite puisque destines au grand public, beacoups d'outils specialises ont etes developpes. On va retrouver des outils d'analyse de protocole sans fil generaux comme la *HackRF* ou sa version specialement concue pour le BLE nommée *Ubertooth One*. Ces cartes sont assez cher mais hautement personnalisables depuis les couches bas niveau. Elles demandent un certain background de connaissances sur le protocole et les modulations sans fil pour arriver a un resultat precis (comme la realisation d'une attaque).
Viennent ensuite les *sniffers* sous forme de dongle USB arrangées et plus ou moins personnalisables. Beaucoup sont basés sur les memes puces de *Nordic Semiconductor* ou *Texas Instrument* qui eux meme proposent leurs sniffers[14,15] et logiciels[16,17] pour l'analyse du protocole BLE. Dans les initiatives plus open-source, mais pas encore totalement personnalisable sans reprogrammation de la puce, on peut citer le Bluefruit[18] de *Adafruit*.
Enfin, un outils open-source nommé *BTLEJack* permet non seulement l'etude mais la mise

[13] https://www.wireshark.org/
[14] https://www.nordicsemi.com/Software-and-tools/Development-Kits/nRF51-Dongle
[15] http://www.ti.com/tool/CC2540EMK-USB
[16] https://www.nordicsemi.com/Software-and-tools/Development-Tools/nRF-Sniffer-for-Bluetooth-LE
[17] http://www.ti.com/tool/PACKET-SNIFFER
[18] https://www.adafruit.com/product/2269

en place d'une multitude d'attaques sur le protocole Ble via reprogrammation de la carte avec un firmware personnalisé. Cet outils a ete developpe pour la carte *BBC Micro:Bit*[19], une carte de developpement bon marché a but educatif, et est aujourd'hui compatible avec plusieurs autres cartes intégrant une puce `nRF51` (notamment la *Bluefruit*). Basé sur les travaux de *BTLEJack*[20] et d'autres librairies BLE en python, *Mirage*[21] permet des fonctionnalités identiques en supportant encore plus de cartes, de protocoles et d'attaques. Il comble le manque de flexibilité des precedants outils en integrant plusieurs mecanismes permettant la mise en place d'attaques scénarisées entierement personnalisees depuis les couches protocolaires basses et facilite l'ajout de fonctionnalités au sein du framework.

4.2 Attaques

Scanning

Le *scanning* consiste à répertorier des appareils BLE à proximité. Dans le cas d'attaque on etendra l'inventaire avec les connexion établies entre 2 appareils BLE. Là ou les *dongles HCI* suffisent pour intercepter les annonces diffusées, l'analyse des communications établies requiert un *sniffer* capable de suivre les 37 canaux de données.
La pile protocolaire *BlueZ* permet le *scan* des *advertisements* (annonces) tandis que plusieurs outils precedements evoques comme *smartRF* ou *nRFSniffer* suffisent pour repérer une communication.

Spoofing

C'est l'une des étape du *Man-In-The-Middle* qui permet d'usurper un esclave BLE. Après identification de la victime (via annonce ou adresse BD), l'attaquant la clone en s'y connectant et extractant son profile *GATT*. L'attaquant peut resté connecté pour garder la victime silencieuse (un esclave connecté n'emettant pas d'annonces) puis, via un second *dongle* BLE, s'annonce comme étant l'appareil precedemment cloné.
Cette attaque est realisable en utilisant simplement un *dongle HCI* et l'utilitaire *BLueZ*. Les librairies et frameworks d'attaque discutés plus auparavant (*GATTAcker*, *BTLEJack*) integrent egalement ces mecanisme.

Sniffing

Le *sniffing* est l'analyse voir le suivis d'une connexion BLE (suivant les capacites du *sniffer* utilisé). Un premier cas de figure est l'attente d'une nouvelle connexion pour se synchroniser avec afin de suivre les echanges. La seconde option, et la plus courante, est la synchronisation avec une connexion deja etablie: la difficulté ici réside en la recuperation des parametres de connexion. Il est necessaire de retrouver la carte des canaux utilisés (*channel map*) ainsi que le *hop increment* (nombre de canaux sautés) et *hop interval* (temps entre chaque saut) pour se synchroniser, sans quoi il est impossible de suivre une connexion car les sauts impredictibles et trop frequents.
BTLEJack, et par conséquent *Mirage*, mettent en place un mecanisme permettant de retrouver ces informations de connexion a partir des echanges interceptés lors du *scanning*. Le *sniffing* de communications sans synchronisation quant a lui est une fonctionnalité très répandue et integrée a tout les sniffers BLE vu antierieurement.

[19] https://microbit.org/
[20] https://github.com/virtualabs/btlejack
[21] https://homepages.laas.fr/rcayre/mirage-documentation/index.html

Man-In-The-Middle

L'attaque *Man-In-The-Middle* concerne n'importe quelle communication: l'attaquant peut modifier les communications en vennant se placer entre l'emetteur et le recepteur ciblés, se faisant passer pour l'un apres de l'autre en usurpant leurs identités. Dans le cas du BLE on utilisera deux *dongles*, un pour usurpé l'esclave cible et un autre pour maintenir une connexion avec celui-ci. On doit d'abors usurpé l'esclave cible via du *spoofing* puis attendre la connexion d'un maitre, une fois le maitre connecté à notre *dongle* usurpateur on se retrouve en situation de *Man-In-The-Middle* entre l'esclave et le maitre: on peut suivre et modifier le traffic avant de le retransmettre. A noté que l'on peut egalement usurpé le maitre si les appareils ciblés attendent un appareil précis pour s'appairer.
Plusieurs outils sont dediés a cette attaque car populaire et simple a mettre en oeuvre: *GATTAcker* et *BTLEJuice* facilitent la mise en place en automatisant les etapes a partir de la cible choisie. Ce sont d'assez ancien outils qui aujourd'hui souffrent de lacunes de part les technologies utilisees. Basés sur *Noble* et *Bleno*, des librairies en JavaScript basées sur NodeJS et permettant de manipuler le BLE, ils manque de flexibilité et ne permettent pas entre autre la coexistence d'appareils BLE, obligant l'utilisation de machine virtuelle pour chaque *dongle*. *Mirage* reprend le fonctionnement de ces outils, l'integrant en tant que module, mais basé sur de nouvelles librairies, notamment *PyBT*[22] permettant de simuler le comportement d'un appareil BLE en s'affranchissant des contraintes imposees par leurs equivalent JavaScript, *Noble* et *Bleno*.

Jamming

Le brouillage de communication est egalement une attaque assez populaire et implementee dans bon nombre d'outils sur le marché. Le but est de creer du bruit sur le canal au moment de la transmission pour corrompre le message, le rendant inutilisable par le recepteur. Concernant le BLE, *Ubertooth One* dispose des capacités nécessaire pour brouiller les canaux d'annonce ainsi qu'une communication etablie par retransmission simultanee. *BTLEJack* implemente egalement le brouillage au sein de son firmware personnalisé et, ayant deja un mecanisme permettant de se synchroniser avec une communication, peut egalement brouiller une communication etablie.

Hijacking

Le principe est de voler une connexion entre 2 appareils en forcant une deconnexion de l'un pour prendre sa place. Cette nouvelle attaque, implementée par *BTLEJack* et reprise dans *Mirage*, utilise les différences de *timeout* entre le *central* et le *peripheral* pour forcer le *central*, via l'utilisation de brouillage sur les paquets du *peripheral*, à se déconnecter et prendre ainsi sa place au sein de la communication.

4.3 Mirage

Après comparaison entre les outils disponibles (voir tbl. 6), j'ai choisie *Mirage* car il dispose de la flexibilite voulue pour implementee des attaques scenarisee: acces aux couches bas niveau pour recuperer informations comme force du signal et calibrage (necessaires pour calculer la position d'un appareil lors de la localisation). Il dispose egalement d'une implementation d'un *central* et *peripheral* personnalisables pour realiser un reseau de tests sur lequel verifie l'implementation des attaques. Enfin, il supporte une varieté de

[22] https://github.com/mikeryan/PyBT

composants matériel[23] ainsi que toutes les attaques necessaires[24] pour la preuve de concept, rendant le developpement plus simple.

Concernant le matériel necessaire, la preuve de concept necessite d'abors deux *dongles HCI* compatibles avec Mirage et supportant le changement d'adresse *BD* pour le *spoofing*. *Mirage* se base sur les numeros de constructeurs des *dongles* definit par le Bluetooth[25] pour savoir s'ils sont compatibles. Il supporte une variete des constructeur dont le *CSR8510*[26] de Qualcomm, puce tres populaire dans les *dongles HCI* basiques et permettant le changement d'adresse *BD*.
Concernant le *sniffer* necessaire pour la pupart des attaques, *Mirage* se basant sur *BTLE-Jack*, les cartes supportées par celui-ci le sont aussi par *Mirage*. Même si *Mirage* supporte d'autres cartes comme celles de developpement de *Nordic Semiconductor* ou l'*Ubertooth One*, elles ne sont pas adaptées pour mon projet car trop onereuses pour les fonctionnalités exploitees dans la preuve de concept.
Je me suis donc tourné vers les cartes compatibles avec *BTLEJack*[27]. *Bluefruit* d'Adafruit, Waveshare *BLE400*[28] et les kits *nRF51*[29] demandent une reprogrammation via un peripherique externe utilisant le port *SWD*. Ne disposant pas du materiel necessaire pour la reprogrammation, et celui-ci etant assez onéreux, j'ai choisit la *BCC Micro:Bit*: carte avec laquelle *BTLEJack* à ete originalement developpé.

Tableau 6: Comparaison des outils pour l'étude offensive du BLE

Logiciel	*scan*	*sniff*	*mitm*	*jam*	*hijack*	*locate*	Matériel
nRFSniffer	oui	oui	non	oui	non	non	puce `nRF51`
TIsmartRF	oui	oui	non	non	non	non	puce `CC25xx`
BTLEJuice	oui	non	oui	non	non	non	*dongle HCI + Bleno/Noble*
GATTAcker	oui	non	oui	non	non	non	*dongle HCI + Bleno/Noble*
BTLEJack	oui	oui	oui	oui	oui	non	*BBC Micro:Bit*, cartes basées sur puce `nRF51`
Mirage	oui	oui	oui	oui	oui	possible	*dongle HCI*, *Ubertooth*, *nRF*, cartes compatibles avec *BTLEJack*

Intégration

Mirage est un "*framework offensif pour l'audit des protocoles sans fil*"[30]. Il a ete pense pour le *pentest* (audit de sécurité) donc un usage exclusivement en ligne de commandes (*CLI*). Meme si le framework se veut beaucoup plus modulaire et extensible que ces predecesseurs (BTLEJack notamment), cette modularite a ete pensee pour l'interface en ligne de commandes.
Mirage fournit des briques logicielles pour communiquer avec des appareils (dongles, sniffers) ainsi que decortiquer les protocoles, ce qui constitue le coeur du framework. Ces briques

[23] https://homepages.laas.fr/rcayre/mirage-documentation/devices.html
[24] https://homepages.laas.fr/rcayre/mirage-documentation/modules.html
[25] https://www.bluetooth.com/specifications/assigned-numbers/company-identifiers/
[26] https://www.qualcomm.com/products/csr8510
[27] https://homepages.laas.fr/rcayre/mirage-documentation/devices.html#btlejack-device
[28] https://www.waveshare.com/ble400.htm
[29] https://www.waveshare.com/nrf51822-eval-kit.htm
[30] https://homepages.laas.fr/rcayre/mirage-documentation/index.html

logicielles de base sont utilisees par des *modules* dans un but precis, par exemple realiser le brouillage d'une communication BLE. Mirage fournie un certain nombre de modules fournissant les attaques retrouvees dans les autres outils d'audit du BLE (*BTLEJack*, *GATTAcker*, ...) precedement discutes. Similaire a la philosophie d'Unix, ces modules sont specialisés dans une tache precise et peuvent etres assemblés entre eux pour realise des fonctionnalites plus complexes comme la connexion avec le module *ble_connect* puis l'extraction d'informations avec *ble_discover*. Enfin, il est possible de modifier les etapes d'une attaque via les *scénarios*: chaque module accepte un scénario surchargeant son flux d'execution et ses methodes.

mirage extensible certes mais seulement en CLI selon ces regles Pas pense pour vivre dans un back-end, fait pour le pentest: une attaque puis shutdown => etat instable apres une attaque, non fiabilite des appareils utilises pour conduire l'attaque car fermeture des connexions non effectué normalement fait par shutdown de mirage Creation API pour integration avec Flask mais couches CLI et Modules fortements liées, problemes de personnalisations de l'API sans modifier le coeur des modules ou de mirage => instanciation manuelle de l'app et des modules au besoin

Interfaces

Inutile et long de reconduire l'interface CLI de mirage en API pour le back, la rendre accessible depuis GUI via websockets puis la rendre en HTML/CSS/JS. Les attaques selectionnees demandent actions utilisateurs car modification a la volee d'informations, seul le scan et localisation sont autonomes. Decision de laisse le front pour demo localisation et scan pour sensibilisation fuite d'informations passive BLE (smartphones, airpods, pc).
Utilisation CLI Mirage avec modules personnalisés pour conduire les attaques choisies (et meme plus puisque mirage dispose d'une multitudes d'attaques).

5 Travail realise

5.1 Scan et localisation

1. comment trouver ? ble_sniff, scan 3 canaux d'annonces => devices ble_sniff scan 37 canaux de donnees => connections PB connexions pe long, tres long car jeux chat et souris, possible para (mirage para les annonces avec 3 btlejack) sur x canaux donnees pour accelerer
2. Comment calculer ? RSSI => peu reliable, necessite un calibrage, path-loss, dead-reckoning TOA => necessaire de controller appareil emetteur, ici attaque donc impossible

fingerprinting => liste d'appareils et leurs positions connues, impossible dans attaque trilateration => necessite points de donnes (parrallele GPS) AOA/AOD => necessite materiel adequat et BLE 5.1 => triangulation ou AOA+TOA/RSSI

3. comment integrer ? ble_locate facon de faire de mirage avec module CLI, cleanup des threads/fifo entre scan device et connections extension du ble_sniff et device btlejack pour integrer mes changements: but ne pas modifer le firmware car la recompilation demande une environnement precis et complexe a mettre en place.
4. GUI

5.2 MITM

5.3 Hijack

5.4 Tests et validation

Connexion factice car aucun appareil BLE (coronavirus modification du sujet)