



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

1	Objets connectés	6
1.1	Architecture	6
1.2	Protocoles	6
2	Bluetooth Low energy	8
2.1	Différences	8
2.2	Protocole	8
2.3	Évolution	14
3	Preuve de concept	15
3.1	Travail demandé	15
3.2	Architecture	16
3.3	Interface	16
4	Étude de l'existant	17
4.1	Outils	17
4.2	Attaques	18
4.3	Mirage	19
5	Travail réalisé	22
5.1	Scan	22
5.2	Localisation	22
5.3	MITM	22
5.4	Hijack	23
5.5	Tests et validation	23
5.6	Améliorations	23

Tableaux

1	Cas d'utilisation et protocole Bluetooth adapté	8
2	Capacités d'entrée possibles	11
3	Capacités de sortie possible	11
4	Capacité d'entrées/sorties de l'appareil	11
5	Méthode d'appairage utilisée en fonction des capacités échangées	12
6	Comparaison des outils pour l'étude offensive du BLE	20

Figures

1	Répartition du spectre BLE en canaux	9
2	Étapes d'un échange BLE	10
3	Client et serveur GATT	14
4	Architecture du système	16

Due aux conditions exceptionnelles imposées par la pandémie du COVID-19, j'ai réalisé un sujet fourni par le laboratoire d'informatique LAB-STICC, affilié à l'université dans le cadre du stage de première année de master CSSE à l'UBS de Lorient.

Je devais d'abord travailler avec une carte HackRF pour étudier les protocoles de communication domotique (ZigBee, ZWave, Thread ...) mais due au confinement j'ai dû m'adapter et me suis rabattu sur du matériel et un protocole accessible: le Bluetooth Low Energy. Son intégration dans nombre d'appareils de bureautique en ont fait un choix pour la communication avec les systèmes embarqués constituant les objets intelligents. L'étude du protocole est facilitée par cette popularité, disposant de matériel dédié à moindre coût sur le marché ainsi qu'une flopée d'outils logiciels et d'audits de sécurité révélant et expliquant les vulnérabilités du protocole.

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 intégré dans la plupart des appareils de bureautique, il est rapidement devenu populaire dans l'internet des objets.

La première itération du BLE ne répond plus aux exigences de sécurité contemporaine et même si le protocole a 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 être interceptées, voir modifiées pour permettre des actions aux dépens de l'utilisateur.

1 Objets connectés

Avec l'explosion de l'internet de objets au cours de ces dernières années, toute une flopée d'objet du quotidien ont été 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 électroniques, 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 été utilisés pour la mise en place des réseaux de capteurs. Cette utilisation a 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* a largement pris le dessus car plus adapté avec son standard *Low Energy*. Conçu 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 développés 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 spécialisés ont développés *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'être obsolète, 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 périphériques 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 partie des protocoles *WPAN* et leur distance d'émission varie de quelques mètres jusqu'à 30 mètres.

La spécification Bluetooth 4.0, sortie en 2010, intègre le protocole LE (*Low Energy*) et permet au Bluetooth de toucher le marché des systèmes embarqués fonctionnant 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 fonctionnent avec une connexion continue et un transfert en mode flux.

Le BLE, visant à réduire la consommation d'énergie, n'établit 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 reçue, 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 besoins différents (voir tbl. 1).

Tableau 1: Cas d'utilisation et protocole Bluetooth adapté

Besoin	Flux données	Transmission données	Localisation	Reseau capteurs
Appareils	ordinateur, smartphone, casque, enceinte, voiture	accessoires bureautique ou fitness, équipement médical	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 définit 4 profils en fonction du rôle de l'appareil: **P**eripheral, **C**entral, **B**roadcaster, **O**bserver. 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 très populaire pour les beacons. L'*Observer* est opposé, il ne fait qu'écouter les annonces, n'établira jamais de connexion.

Le *Peripheral* et le *Central* forment la seconde paire et permettent la mise en place d'une architecture client-serveur. Le *Peripheral* joue le rôle 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 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 choisis pour ne pas interférer 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.

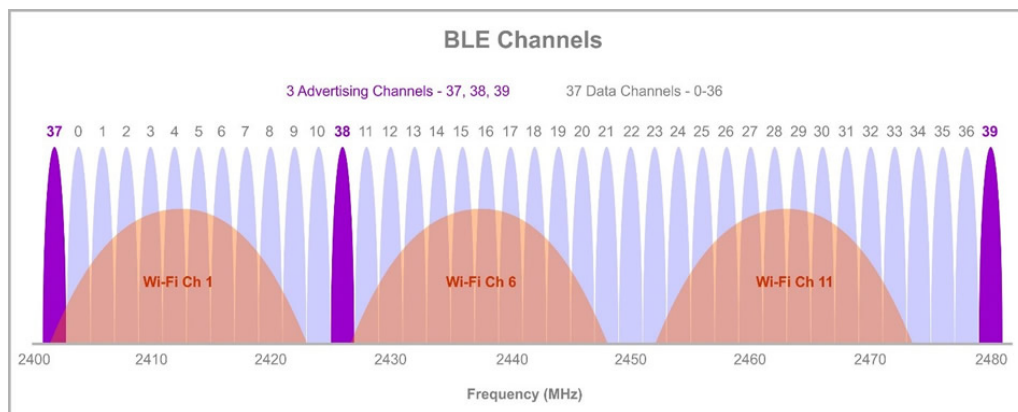


Figure 1: Répartition du spectre BLE en canaux⁵

Couche logique

1. Annonces

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

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

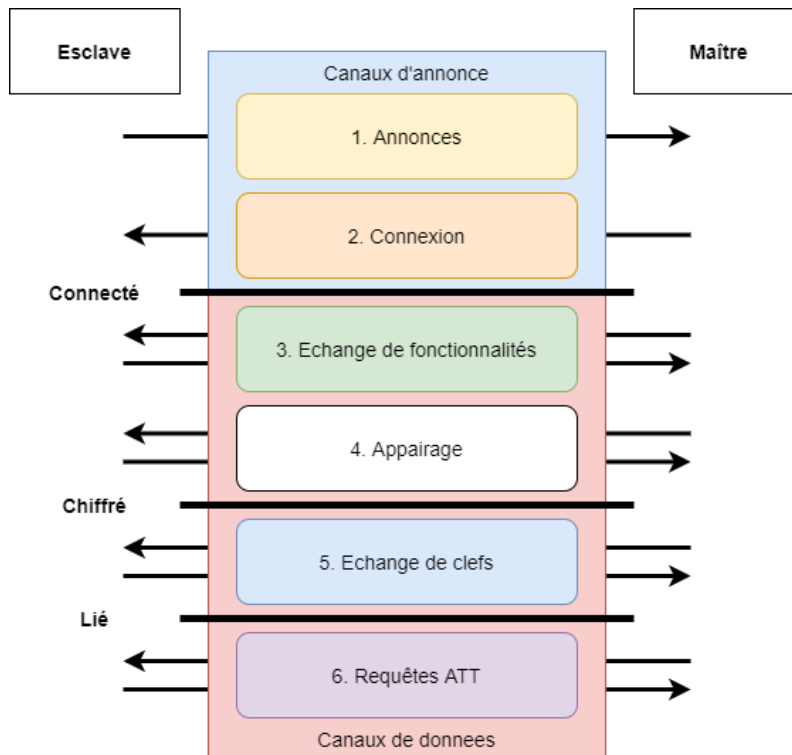


Figure 2: Étapes d'un échange BLE

constituent 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 BLE⁷. 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 adresse 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 profiles ne sont certes pas exhaustifs mais permettent une intégration fonctionnelle avec un maximum d'appareils et prévoient un moyen d'intégrer des données propriétaires non standardisées⁸.

2. Connexion

Lorsqu'un *central* reçoit 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/>

⁸https://www.silabs.com/community/wireless/bluetooth/knowledge-base.entry.html/2017/02/10/bluetooth_advertisingGsf

lui envoie 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 Address*).

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 transmet 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 protocole 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 possibles⁹

Capacité	Description
No input	pas la capacité d'indiquer <i>oui</i> ou <i>non</i>
Yes/No	mécanisme permettant d'indiquer <i>oui</i> ou <i>non</i>
Keyboard	clavier numérique avec mécanisme <i>oui/non</i>

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 échangées¹⁰

	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 méthode *JustWorks*. C'est celle par défaut lorsque deux appareils ne disposent pas des capacités nécessaires pour une autre. Elle est notamment beaucoup présente pour les objets connectés puisque n'intégrant pas de mécanismes pour un appairage plus complexe (clavier ou écran).

Passkey permet d'authentifier l'appairage pour se protéger des usurpations d'identité (*Spoofing* et *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 matériel cryptographique: la clef temporaire (*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 être trouvée 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'échange du matériel cryptographiques et est donc résistante à l'écoute passive (*eavesdropping*) mais toujours vulnérable à l'usurpation d'identité avec *JustWorks*.

5. Chiffrement

L'établissement du chiffrement de la connexion est ensuite réalisé par dérivation de la première clef temporaire transmise via la méthode d'appairage choisie et d'autres matériel cryptographique échangés via le protocole BLE. La clef obtenue est dite court terme (*Short Term Key*) car elle ne sera utilisée que pour cette connexion et devra être re-générée à 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 été exigé lors de l'échange des capacités. La clef long terme (*LTK*) 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 compréhension actuelle du protocole BLE et du fonctionnement de l'appairage, il semble recommandé de mettre en place une connexion sécurisée dès que possible. Il est

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

également judicieux d'éviter la méthode *JustWorks* au maximum et stocker une session pour éviter de réitérer l'appairage.

Cependant, il est assez simple de forger un échange de capacités pour rétrograder la connexion en *legacy* et forcer *JustWorks* via les capacités échangées. C'est pourquoi certains appareils attendent des capacités et exigences minimales pour établir une connexion, sans quoi celle-ci est avortée. C'est notamment le cas d'appareils propriétaires conçus pour fonctionner ensemble.

6. Requêtes

Les échanges sont réalisés sur la base d'une architecture client-serveur. Le *central* (client) interroge le *peripheral* (serveur) avec le protocole *ATT* (*ATtribute Protocol*). Chaque requête mène soit à une réponse du serveur, soit à la mise en place d'une notification lors d'un évènement (valeur changée ou disponible).

Les requêtes et réponses possibles sont standardisées dans le *GATT* (*Generic ATtributes*) pour permettre une interoperabilité maximale entre les appareils (comme pour le *GAP*). *GATT* et *GAP* partagent les mêmes profils, seul la structure change. Le serveur *GATT* peut être interrogé pour établir une liste exhaustive de toutes les fonctionnalités d'un appareil la ou le *GAP* choisit ce que contient l'annonce mais est limité par la taille du paquet (31 octets).

GAP

Dans le cas des *Peripherals* et *Centrals*, le *GAP* est principalement utilisé pour établir un profil du *peripheral* permettant la décision de connexion de la part du *central*.

Pour les *Boardcasters* et *Observers* il permet la communication unidirectionnelle (*Broadcaster* vers *Observer*) via les annonces, ceux-ci utilisant la diffusion plutôt qu'une connexion point à point. On retrouve cette utilisation pour les beacons publicitaires ou de localisation intérieur.

GATT

Pour l'échange 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 *caractéristiques*, chacune contenant une *valeur* et un ou plusieurs *descripteurs* fournissant des informations additionnelles sur la valeur (voir fig. 3).

À chacune de ses couches (service, caractéristique, 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/caractéristique/attribut/descripteur ne peut l'utiliser.

Un service correspond généralement a un profil (standardisé ou non) comme par exemple un thermomètre. Ce service exposerait des caractéristiques comme la température ou l'humidité. Chacune de ces caractéristique contient la valeur (donnée brute) et des descripteurs pour indiquer l'unité ou encore un facteur ou formule pour convertir la valeur donnée en résultat exploitable.

À moins de connaître exactement l'appareil et de l'interroger à l'aveugle via les *handles* (ce qui peut être le cas entre des appareils propriétaires), il faut procéder par étape en découvrant d'abors les services disponibles, puis chaque caractéristique par service et enfin

les valeurs de celles-ci.

Pour procéder à la découverte d'un appareil, le protocole *ATT* dispose d'un type de requête par couche à interroger (voir fig. 3). Une fois le service voulu trouvé (ou la cartographie totale de l'appareil réalisée), on peut lire, écrire ou souscrire à des attributs directement par *handle*. Le *GATT* met en place un système de droits par attribut pour protéger la lecture, l'écriture et la souscription par le client.

Le *GATT* définit également des services standardisés appelés primaire et secondaire censés être présent sur tous les appareils BLE afin de connaître les fonctionnalités standardisées (service primaire) et propriétaires (service secondaire) de l'appareil. Comme les *handle* sont définies arbitrairement par le serveur *GATT*, les profils standards et leurs services/caractéristiques sont identifiés par un *UUID* standardisé identique dans tout les appareils BLE.¹¹

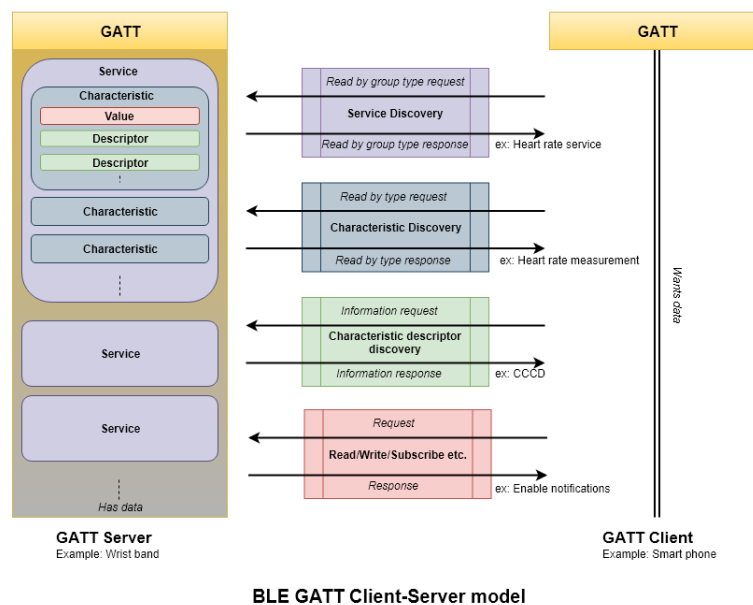


Figure 3: Client et serveur GATT¹²

2.3 Évolution

Depuis sa première itération en 2010 dans la version 4.0 des spécifications Bluetooth le BLE a évolué pour intégrer des mesures de sécurité avec l'ajout des connexions sécurisées *LE* en 4.2 puis la diversification des topologies avec l'introduction du *mesh* pour les réseaux de capteurs en 5.0 et dernièrement l'amélioration de la localisation intérieur (*Indoor Positionning System*) pour une précision de l'ordre du centimètre grâce aux systèmes angle d'arrivée et de départ (*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 versions ultérieures du protocole (aujourd'hui en version 5.1). Néanmoins cela permet d'étudier et comprendre les mécanismes du BLE depuis sa création, puis voir les alternatives qui ont été proposées pour mitiger 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 répandu dans les appareils connectés BLE. La majeure partie des appareils connectés sont simples, ne réalisant 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 mécanismes proposés à partir de la version 4.2 du BLE sont beaucoup plus robustes. Ils apportent une connexion sécurisée (LE Secure Connections ou LESC) se basant sur Diffie Hellmann pour l'échange de clefs ainsi qu'une nouvelle méthode d'appairage authentifiée (Comparaison numérique).

Même si le BLE a toujours proposé des mesures de sécurité, la majorité des constructeurs ne les utilisent pas et mettent en place des chiffrements au niveau de la couche application (le BLE chiffre depuis la couche lien).

Ces mesures de sécurité propriétaires sont souvent basées sur des algorithmes reconnus comme AES et des méthodes comme le *challenge-response* pour authentifier un appareil. Une clef unique est intégrée dans chaque appareil, celle-ci sera soit distribuée au propriétaire de l'appareil lors de la création du compte ou le téléchargement de l'application associée, soit gardée par le constructeur qui transmettra directement les commandes depuis l'utilisateur à l'appareil (via l'application ou directement, si l'appareil est connecté au réseau mondial). Ces mécanismes exposent cependant beaucoup plus d'informations que le chiffrement BLE depuis la couche lien. Les requêtes *ATT* et *GATT* transissent en clair et pour pallier à cette fuite d'informations les constructeurs évitent les requêtes standardisées dans le BLE et préfèrent utiliser des protocoles personnalisés dans la couche application, celle-ci étant chiffrée.

Ces chiffrements propriétaires sur la couche application sont hors de portée de mon sujet mais ont fait couler beaucoup d'encre. Plusieurs présentations et leurs *whitepaper* sont disponibles dans les conférences *black hat*¹³, *Defcon*¹⁴ ou encore *SSTIC*¹⁵.

Maintenant il s'avère que beaucoup d'appareils autonomes simples ne mettent en place aucune mesure de sécurité, qu'elle soit standardisée ou propriétaire, car les données qui transissent ne sont pas jugées sensibles. C'est notamment le cas des objets domotiques autonomes comme les télécommandes pour lampes dites connectées.

3.1 Travail demandé

Mettre en place un outil basé sur un framework offensif permettant de répertorier et faciliter l'analyse des appareils et connexion BLE alentours. Cet outil est facilement portable sur diverses cartes de développement comme la Raspberry Pi car conteneurisé avec *Docker* et

¹³<https://www.blackhat.com/>

¹⁴<https://www.defcon.org/>

¹⁵<https://www.sstic.org/>

se basant sur du matériel USB pour l'étude du protocole.

3 ports USB suffisent pour permettre de conduire toutes les attaques proposées par le framework offensif utilisé: 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 intégrée à 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 détournement par brouillage d'une connexion précédemment identifiée (*hijacking*)

3.2 Architecture

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

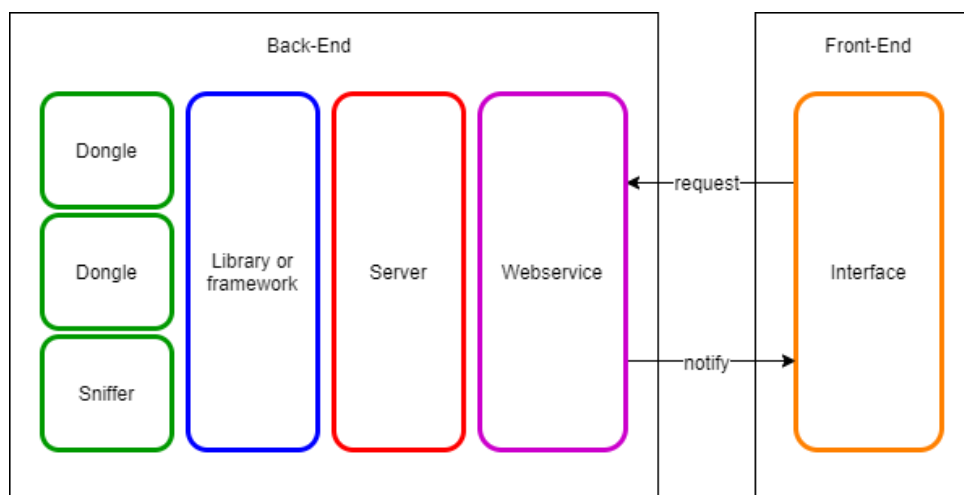


Figure 4: Architecture du système

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 transmissions 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 récupérer, analyser et modifier les échanges entre appareils permettant de réaliser des audits de sécurité ou mettre en place des attaques exposant des vulnérabilités. L'analyse du trafic sans fil BLE demande une antenne couvrant la bande utilisée par les 40 canaux du protocole ainsi qu'un système assez rapide pour scanner puis suivre les communications lors des sauts de fréquence. Les radio-logiciels (*SDR*) ne sont donc pour la plupart pas adaptés car trop lents ou trop cher pour les fonctionnalités voulues : des outils spécialisés dans l'analyse et l'attaque du BLE sont disponibles pour une fraction du prix.

Le premier outil, utilisé dans tous nos appareils BLE, est la puce intégrée pour les communications BLE. La récupération (*sniffing*) et analyse ou modification d'un trafic sans fil étant interdit, ces puces utilisent un *firmware* propriétaire conforme aux rôles déterminés dans les spécifications BLE. Même si il reste possible d'analyser le trafic (notamment en utilisant *Wireshark*¹⁶) entre la puce et un appareil BLE, il n'est pas possible d'étendre les capacités de celle-ci sans modifier le *firmware*.

Tous les appareils ne disposant pas d'une puce BLE dédiée, les constructeurs ont développé des *dongles* intégrant 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 à proximité et d'endosser le rôle de *peripheral* ou *central* pour établir 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. Même si certains de ces *dongles* proposent des fonctionnalités intéressantes 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 requiert un moyen d'intercepter le trafic. Les *dongles* et *puces* embarquant des firmwares ne permettant pas cette fonctionnalité puisque destinés au grand public, beaucoup d'outils spécialisés ont été développés. On va retrouver des outils d'analyse de protocole sans fil généraux comme la *HackRF* ou sa version spécialement conçue 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 à un résultat précis (comme la réalisation d'une attaque).

Viennent ensuite les *sniffers* sous forme de dongle USB arrangées et plus ou moins personnalisables. Beaucoup sont basés sur les mêmes puces de *Nordic Semiconductor* ou *Texas Instrument* qui eux même proposent leurs sniffers^{17,18} et logiciels^{19,20} 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 Bluefruit²¹ de *Adafruit*.

Enfin, un outil open-source nommé *BTLEJack* permet non seulement l'étude mais la mise

¹⁶<https://www.wireshark.org/>

¹⁷<https://www.nordicsemi.com/Software-and-tools/Development-Kits/nRF51-Dongle>

¹⁸<http://www.ti.com/tool/CC2540EMK-USB>

¹⁹<https://www.nordicsemi.com/Software-and-tools/Development-Tools/nRF-Sniffer-for-Bluetooth-LE>

²⁰<http://www.ti.com/tool/PACKET-SNIFFER>

²¹<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 outil a été développé pour la carte *BBC Micro:Bit*²², une carte de développement bon marché à but éducatif, et est aujourd'hui compatible avec plusieurs autres cartes intégrant une puce nRF51 (notamment la *Bluefruit*). Basé sur les travaux de *BTLEJack*²³ et d'autres bibliothèques BLE en python, *Mirage*²⁴ permet des fonctionnalités identiques en supportant encore plus de cartes, de protocoles et d'attaques. Il comble le manque de flexibilité des précédents outils en intégrant plusieurs mécanismes permettant la mise en place d'attaques scénarisées entièrement personnalisées 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 étendra l'inventaire avec les connexions établies entre 2 appareils BLE. Là où 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 précédemment évoqués comme *smartRF* ou *nRFSniffer* suffisent pour repérer une communication.

Spoofing

C'est l'une des étapes 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 extrayant son profil *GATT*. L'attaquant peut rester connecté pour garder la victime silencieuse (un esclave connecté n'émettant pas d'annonces) puis, via un second *dongle* BLE, s'annonce comme étant l'appareil précédemment cloné.

Cette attaque est réalisable en utilisant simplement un *dongle HCI* et l'utilitaire *BLuez*. Les bibliothèques et frameworks d'attaque discutés plus auparavant (*GATTacker*, *BTLEJack*) intègrent également ces mécanismes.

Sniffing

Le *sniffing* est l'analyse voire le suivi d'une connexion BLE (suivant les capacités du *sniffer* utilisé). Un premier cas de figure est l'attente d'une nouvelle connexion pour se synchroniser avec afin de suivre les échanges. La seconde option, et la plus courante, est la synchronisation avec une connexion déjà établie : la difficulté ici réside en la récupération des paramètres de connexion. Il est nécessaire 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 imprédictibles et trop fréquents.

BTLEJack, et par conséquent *Mirage*, mettent en place un mécanisme permettant de retrouver ces informations de connexion à partir des échanges interceptés lors du *scanning*. Le *sniffing* de communications sans synchronisation quant à lui est une fonctionnalité très répandue et intégrée à tous les sniffers BLE vu antérieurement.

²²<https://microbit.org/>

²³<https://github.com/virtual-labs/btlejack>

²⁴<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 venant se placer entre l'émetteur et le receveur ciblés, se faisant passer pour l'un après de l'autre en usurpant leurs identités. Dans le cas du BLE on utilisera deux *dongles*, un pour usurper l'esclave cible et un autre pour maintenir une connexion avec celui-ci. On doit d'abord usurper l'esclave cible via du *spoofing* puis attendre la connexion d'un maître, une fois le maître connecté à notre *dongle* usurpateur on se retrouve en situation de *Man-In-The-Middle* entre l'esclave et le maître: on peut suivre et modifier le trafic avant de le retransmettre. A noter que l'on peut également usurper le maître si les appareils ciblés attendent un appareil précis pour s'appairer.

Plusieurs outils sont dédiés à cette attaque car populaire et simple à mettre en œuvre: *GATTacker* et *BTLEJuice* facilitent la mise en place en automatisant les étapes à partir de la cible choisie. Ce sont d'assez anciens outils qui aujourd'hui souffrent de lacunes de part les technologies utilisées. Basés sur *Noble* et *Bleno*, des bibliothèques en JavaScript basées sur NodeJS et permettant de manipuler le BLE, ils manquent de flexibilité et ne permettent pas entre autre la coexistence d'appareils BLE, obligeant l'utilisation de machine virtuelle pour chaque *dongle*. *Mirage* reprend le fonctionnement de ces outils, l'intégrant en tant que module, mais basé sur de nouvelles bibliothèques, notamment *PyBT*²⁵ permettant de simuler le comportement d'un appareil BLE en s'affranchissant des contraintes imposées par leurs équivalents JavaScript, *Noble* et *Bleno*.

Jamming

Le brouillage de communication est également une attaque assez populaire et implémentée dans bon nombre d'outils sur le marché. Le but est de créer du bruit sur le canal au moment de la transmission pour corrompre le message, le rendant inutilisable par le receveur. Concernant le BLE, *Ubertooth One* dispose des capacités nécessaires pour brouiller les canaux d'annonce ainsi qu'une communication établie par retransmission simultanée. *BTLEJack* implémente également le brouillage au sein de son firmware personnalisé et, ayant déjà un mécanisme permettant de se synchroniser avec une communication, peut également brouiller une communication établie.

Hijacking

Le principe est de voler une connexion entre 2 appareils en forçant une déconnexion de l'un pour prendre sa place. Cette nouvelle attaque, implémentée 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 tabl. 6), j'ai choisi *Mirage* car il dispose de la flexibilité voulue pour implémenter des attaques scénarisées: accès aux couches bas niveau pour récupérer informations comme force du signal et calibrage (nécessaires pour calculer la position d'un appareil lors de la localisation). Il dispose également d'une implémentation d'un *central* et *peripheral* personnalisables pour réaliser un réseau de tests sur lequel vérifier l'implémentation des attaques. Enfin, il supporte une variété de

²⁵<https://github.com/mikeryan/PyBT>

composants matériel²⁶ ainsi que toutes les attaques nécessaires²⁷ pour la preuve de concept, rendant le développement plus simple.

Concernant le matériel nécessaire, la preuve de concept nécessite d'abord deux *dongles HCI* compatibles avec Mirage et supportant le changement d'adresse *BD* pour le *spoofing*. *Mirage* se base sur les numéros de constructeurs des *dongles* définis par le Bluetooth²⁸ pour savoir s'ils sont compatibles. Il supporte une variété de constructeurs dont le *CSR8510*²⁹ de Qualcomm, puce très populaire dans les *dongles HCI* basiques et permettant le changement d'adresse *BD*.

Concernant le *sniffer* nécessaire pour la plupart des attaques, *Mirage* se basant sur *BTLEJack*, les cartes supportées par celui-ci le sont aussi par *Mirage*. Même si *Mirage* supporte d'autres cartes comme celles de développement de *Nordic Semiconductor* ou l'*Ubertooth One*, elles ne sont pas adaptées pour mon projet car trop onéreuses pour les fonctionnalités exploitées dans la preuve de concept.

Je me suis donc tourné vers les cartes compatibles avec *BTLEJack*³⁰. *Bluefruit* d'Adafruit, *Waveshare BLE400*³¹ et les kits *nRF51*³² demandent une reprogrammation via un périphérique externe utilisant le port *SWD*. Ne disposant pas du matériel nécessaire pour la reprogrammation, et celui-ci étant assez onéreux, j'ai choisi la *BCC Micro:Bit* : carte avec laquelle *BTLEJack* a été originalement développé.

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

Logiciel	<i>scan</i>	<i>sniff</i>	<i>mitm</i>	<i>jam</i>	<i>hijack</i>	<i>locate</i>	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	<i>dongle HCI</i> + <i>Bleno/Noble</i>
GATTacker	oui	non	oui	non	non	non	<i>dongle HCI</i> + <i>Bleno/Noble</i>
BTLEJack	oui	oui	oui	oui	oui	non	<i>BCC Micro:Bit</i> , cartes basées sur puce nRF51
Mirage	oui	oui	oui	oui	oui	possible	<i>dongle HCI</i> , <i>Ubertooth</i> , <i>nRF</i> , cartes compatibles avec <i>BTLEJack</i>

Intégration

Mirage est un “*framework offensif pour l'audit des protocoles sans fil*”³³. Il a été pensé pour le *pentest* (audit de sécurité) donc un usage exclusivement en ligne de commandes (*CLI*). Même si le framework se veut beaucoup plus modulaire et extensible que ses prédécesseurs (*BTLEJack* notamment), cette modularité a été pensée pour l'interface en ligne de commandes.

Mirage fournit des briques logicielles pour communiquer avec des appareils (*dongles*, *sniffers*) ainsi que décortiquer les protocoles, ce qui constitue le cœur du framework. Ces briques

²⁶<https://homepages.laas.fr/rcayre/mirage-documentation/devices.html>

²⁷<https://homepages.laas.fr/rcayre/mirage-documentation/modules.html>

²⁸<https://www.bluetooth.com/specifications/assigned-numbers/company-identifiers/>

²⁹<https://www.qualcomm.com/products/csr8510>

³⁰<https://homepages.laas.fr/rcayre/mirage-documentation/devices.html#btlejack-device>

³¹<https://www.waveshare.com/ble400.htm>

³²<https://www.waveshare.com/nrf51822-eval-kit.htm>

³³<https://homepages.laas.fr/rcayre/mirage-documentation/index.html>

logicielles de base sont utilisées par des *modules* dans un but précis, par exemple réaliser le brouillage d'une communication BLE. Mirage fournit un certain nombre de modules fournissant les attaques retrouvées dans les autres outils d'audit du BLE (*BTLEJack*, *GATTacker*, ...) précédemment discutées. Similaire à la philosophie d'Unix, ces modules sont spécialisés dans une tâche précise et peuvent être assemblés entre eux pour réaliser des fonctionnalités 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 étapes d'une attaque via les *scénarios*: chaque module accepte un scénario surchargeant son flux d'exécution et ses méthodes.

mirage extensible certes mais seulement en CLI selon ces règles. Pas pensé pour vivre dans un back-end, fait pour le pentest: une attaque puis shutdown => état instable après une attaque, non fiable des appareils utilisés pour conduire l'attaque car fermeture des connexions non effectuée normalement fait par shutdown de mirage. Création API pour intégration avec Flask mais couches CLI et Modules fortement liées, problèmes de personnalisations de l'API sans modifier le cœur des modules ou de mirage => instantiation 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 sélectionnées demandent des actions utilisateurs car modification à la volée d'informations, seul le scan et la localisation sont autonomes. Décision de laisser le front pour la démo de localisation et de scan pour la sensibilisation à la fuite d'informations passive BLE (smartphones, airpods, pc). Utilisation CLI Mirage avec modules personnalisés pour conduire les attaques choisies (et même plus puisque mirage dispose d'une multitude d'attaques).

5 Travail réalisé

5.1 Scan

sniffer micro bit Repertorier les appareils BLE ainsi que connexion etablies a proximite
 Mecanisme d'annonces publique sur canaux 37,38,39, canaux de donnees scannable a
 la recherche de connexions Tout appareil BLE module ble_sniff modifié pour scanner
 indefiniement les connexion alentours Mitigation indirecte integree au protocole par le
 channel hopping et channel map Parametres de connexion personnalisé (channel map,
 channel hop and time), eviter les valeurs par défaut, ne pas emettres d'annonces inutiles

5.2 Localisation

BBC MicroBit Localisation des appareils annoncant alentours avec RSSI et calibrage
 Standard iBeacon, fuite d'info GAP Appareils bureautique (smartphones, pc, ecouteurs
 etc), beacons pub Extraction rssi depuis firmware microbit, equation distance avec rssi
 et txpower lissé sur le temps pour reduire fluctuation Calibrage necessaire par la cible
 car BLE difference puissances emissions, seuil sensibilité micro bit base nRF51 +6dbm,
 fluctuation rssi Reduire payload adv GAP, utiliser type ADV correspondant (pas adv_ind
 tout le temps), certains ne contiennent pas de payload

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 donnees (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 modifier le firmware car la recompilation
 demande une environnement precis et complexe a mettre en place.
4. GUI

5.3 MITM

Deux dongles CSR4.0 Clonage et usurpation du *peripheral* dans l'attente d'une connexion
 du *central* pour voir/modifier puis relayer le trafic. Appairage sans authentication,
 methode de chiffrement faible (crackle pour briser le chiffrement car acces connect req)
 Objets connectes autonomes Module mirage ble_mitm Etre au bon endroit au bon moment,
peripheral non connecté et *central* intention de connexion, phase appairage presente (non
 bonding) Utilisation de methodes d'appairage authentifiee comme PassKey ou NumComp
 + utilisation connexion securisee LE + mise en place de session LTK

5.4 Hijack

Matos Fct Vulns abusees Cibles Implementation Difficultes Mitigations/protections

Utilise MicroBit et CSR4.0 Prendre place d'un appareil dans une connexion. Se base sur écoute passive puis brouillage Pas d'appairage Capteurs/actionneurs (lampes) Module mirage ble_hijack Pas acces a phase de connexion ni d'appairage => demande recover parametres de connexion Mitigé par channel hopping, protégé par appairage

5.5 Tests et validation

scenarios sur ble-master et slave A partir de reseau de test mis en place avec Mirage mock master et slave avec 2 CSR4.0 Connexion factice car aucun appareil BLE (coronavirus modification du sujet)

Scan Connexion longue et incertaine

Locate demande txpower pour localiser car calibrage BLE Valeurs +/-30cm, seuil sensibilité -45dBm to -70dBm dans les faits (-30 to -90 sur le papier)

MITM Manque device pour test

Hijack Longue et incertaine

5.6 Améliorations

Scan / Hijack connexion parrallelisation de sniffers pour augmenter les chances de trouver les canaux de donnees utilises

Locate best multiples antennes AOA/AOD pour augmentation de precision sinon avoir plus de points de mesures (min 3) avec RSSI ajouter des methodes d'amelioration precision RSSI (path loss modele, filtre kalmann ...)