InjectaBLE : injection de trafic malveillant dans une connexion Bluetooth Low Energy

Romain Cayre[1,3], Florent Galtier[1], Guillaume Auriol[1,2], Vincent Nicomette[1,2], Mohamed Kaâniche[1] et Géraldine Marconato[3]
[1]prenom.nom@laas.fr
[3]prenom.nom@airbus.com

[1] CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
[2] Univ de Toulouse, INSA, LAAS, F-31400 Toulouse, France
[3] APSYS.Lab, APSYS

Résumé. Ces dernières années, le Bluetooth Low Energy (BLE) s'est imposé comme l'un des protocoles de communication sans fil les plus populaire pour l'Internet des Objets (IoT). Par conséquent, plusieurs attaques visant le protocole ou ses implémentations ont été publiées récemment, illustrant l'intérêt croissant pour cette technologie. Plusieurs défis techniques majeurs restent cependant irrésolus à ce jour, tels que l'injection de trames, l'usurpation du *Slave* ou l'établissement d'une attaque de type Man-in-The-Middle dans une connexion établie. Dans cet article, nous décrivons une nouvelle attaque nommée *InjectaBLE*, permettant d'injecter du trafic malveillant dans une connexion établie. La vulnérabilité exploitée étant inhérente à la spécification du protocole, elle touche de fait l'ensemble des connexions BLE, indépendamment des équipements utilisés, la rendant particulièrement critique.

Dans cet article, nous décrivons les fondements théoriques de l'attaque, son implémentation en pratique, et nous explorons quatre scénarios offensifs critiques permettant de déclencher une fonctionnalité donnée de l'équipement ciblé, d'usurper les deux rôles impliqués dans une connexion ou de mener une attaque Man-in-the-Middle sur une connexion établie. En conclusion, nous décrivons l'impact de l'attaque et proposons plusieurs contre-mesures.

1 Introduction

De nos jours, les objets connectés font partie intégrante de notre vie : de nombreux objets de notre quotidien, des réfrigérateurs aux montres, intègrent des microcontrôleurs et des modems, leur permettant de communiquer avec leur environnement pour proposer de nouveaux services. Plusieurs protocoles de communication sans fil ont été développés ces dernières années pour mettre en œuvre ces services, parmi lesquels le protocole Bluetooth Low Energy (BLE). Le BLE fournit une pile protocolaire légère adaptée aux contraintes des objets connectés, permettant

aux équipements de communiquer facilement et de manière fiable avec une consommation d'énergie minimale. Le BLE est également largement déployé dans les smartphones, les ordinateurs et les tablettes, permettant des communications directes sans nécessiter la présence de passerelles supplémentaires dans le réseau. En conséquence, de nombreux équipements connectés à l'Internet des Objets (ou IoT) s'appuient déjà sur BLE pour communiquer avec leur environnement.

L'intérêt croissant pour cette technologie soulève des inquiétudes légitimes quant à sa sécurité. Ces dernières années, la sécurité de ce protocole a été activement étudiée à la fois d'un point de vue offensif et défensif, mettant en évidence de graves vulnérabilités dans sa spécification [5] et dans diverses implémentations. Certains articles se sont concentrés sur l'écoute passive d'une connexion BLE, rendue difficile par l'utilisation d'un algorithme de saut de fréquence, tandis que d'autres ont décrit des attaques actives telles que le brouillage ou les attaques *Man-in-the-Middle*. Cependant, à notre connaissance, toutes les techniques offensives publiées jusqu'à présent nécessitent que l'attaque soit menée avant l'établissement de la connexion BLE ciblée, ou sont basées sur des techniques particulièrement invasives telles que le brouillage. Même si certains articles mentionnent une attaque théorique basée sur l'injection de trame dans une connexion établie [17] ou la considèrent difficile à réaliser [19], elle n'a jamais été mise en œuvre en pratique et les conséquences offensives d'une telle stratégie n'ont pas été étudiés à notre connaissance.

Dans cet article, nous démontrons la faisabilité non pas seulement théorique mais également pratique de telles attaques, ce qui augmente considérablement la surface d'attaque du protocole. Nous présentons une nouvelle approche nommée *InjectaBLE* permettant d'effectuer une injection de trame arbitraire dans une connexion BLE déjà établie.

Nous expliquons d'abord ses fondements théoriques, puis présentons diverses expériences illustrant sa faisabilité.

Quatre scénarios offensifs critiques tirant parti de cette attaque par injection sont étudiés : nous montrons qu'un attaquant pourrait utiliser notre approche pour (1) déclencher furtivement une fonctionnalité spécifique d'un équipement, (2) et (3) usurper tout rôle (*Master* ou *Slave*) impliqué dans une connexion établie ou (4) effectuer une attaque de type *Man-in-the-Middle* pendant la connexion. Nous démontrons que la plupart de ces scénarios, jugés irréalistes jusqu'à présent, sont en fait relativement simples à réaliser et pourraient avoir de graves conséquences sur la sécurité des équipements BLE. Nous discutons enfin de l'impact de cette attaque et des contre-mesures potentielles.

En résumé, les principales contributions de l'article sont :
— la présentation d'une nouvelle attaque par injection dans une connexion BLE établie, de ses fondements théoriques à sa mise en œuvre pratique ;
— une analyse de sensibilité, permettant de comprendre l'impact de trois paramètres clés sur le succès de l'injection ;
— quatre scénarios d'attaque basés sur l'attaque par injection, permettant de déclencher de manière malveillante une fonctionnalité spécifique d'un appareil, d'usurper le rôle de tout équipement impliqué dans la connexion et d'effectuer une attaque de type *Man-in-the-Middle* lors d'une connexion établie ;
— la proposition de contre-mesures pour limiter l'impact de cette attaque.

Le papier est organisé de la façon suivante : la section 2 présente l'état de l'art de la sécurité offensive pour le protocole BLE et souligne l'intérêt de notre contribution vis à vis des travaux existants. La section 3 décrit le modèle de menace considéré et présente un aperçu de l'attaque ainsi que les principaux défis qui doivent être relevés pour que celle-ci réussisse. La section 4 introduit quelques concepts clés du protocole BLE (notamment le fonctionnement de la couche liaison), nécessaires à la compréhension de la stratégie d'attaque. Ensuite, la section 5 décrit les fondements théoriques de notre attaque et sa mise en œuvre pratique. La section 6 montre comment cette attaque peut être utilisée pour réaliser quatre scénarios d'attaque, tandis que la section 7 présente un ensemble d'expériences menées pour analyser l'impact de trois paramètres principaux sur le succès de l'attaque. La section 8 propose plusieurs contre-mesures qui pourraient être utilisées pour détecter notre attaque ou en atténuer l'impact. La section 9 conclut l'article.

2 Etat de l'art

Au cours de ces dernières années, plusieurs stratégies ou outils d'attaque visant le protocole BLE ont été publiés.

Le Bluetooth Low Energy propose un mode connecté basé sur un algorithme de saut de fréquence : par conséquent, l'écoute passive d'une connexion BLE est une tâche non triviale dans la mesure où l'attaquant doit être en mesure d'estimer les paramètres de cet algorithme pour pouvoir se synchroniser avec la connexion.

Dans [17], M. Ryan a démontré qu'une connexion spécifique peut être facilement sniffée si l'attaquant est en mesure d'intercepter le paquet

initiant la connexion, qui inclut les paramètres de l'algorithme de saut de fréquence. Il a également montré qu'un attaquant est en mesure d'inférer les paramètres d'une connexion déjà établie par l'analyse d'événements spécifiques. Cette approche a ensuite été améliorée par D. Cauquil dans [8], notamment pour déduire l'ensemble des canaux utilisés par la connexion. Dans [10], ce dernier a également adapté la stratégie d'écoute passive afin de supporter le nouvel algorithme de saut de fréquence introduit dans la version 5.0 de la spécification [5], appelé *Channel Selection Algorithm # 2* et basé sur un générateur pseudo-aléatoire. Enfin, un nouvel outil nommé *Sniffle* a également été publié [16] par S. Qasim Khan. Il offre des fonctionnalités intéressantes d'un point de vue offensif, telles que la prise en charge des nouvelles couches physiques introduites dans la spécification du BLE 5.0 ou un mode permettant de suivre un équipement donné sur les canaux d'*advertising* afin de maximiser les chances de capture d'un paquet d'initiation de connexion.

De multiples attaques actives ont également été présentées ces dernières années. Tout d'abord, les attaques basées sur le brouillage réactif ont été explorées par Brauer et al. dans [6]. Les auteurs ont ainsi décrit une attaque permettant de brouiller sélectivement certaines trames d'*advertising*. D. Cauquil a également présenté un nouvel outil offensif nommé *BTLEJack* [9] permettant de perturber une connexion existante entre deux équipements en bloquant les paquets transmis par l'un des équipements (implémentant le rôle dit *Slave*). La conséquence directe de cette stratégie de brouillage est une déconnexion de l'autre équipement (implémentant le rôle *Master*) ce qui permet potentiellement à l'attaquant de se synchroniser avec le *Slave* à la place du *Master* légitime. Cette attaque permet ainsi l'usurpation du rôle *Master* dans une connexion établie. Cette stratégie ne permet cependant pas d'usurper le rôle *Slave*, objectif pourtant pertinent d'un point de vue offensif. De plus, étant basée sur une technique de brouillage, elle est particulièrement invasive et facilement détectable.

Deux outils majeurs, *GATTacker* [15] de S. Jasek et *BTLEJuice* [7] de D. Cauquil, permettent d'effectuer une attaque de type *Man-in-the-Middle*. La stratégie utilisée par *GATTacker* consiste à cloner les trames d'*advertising* émises par l'équipement cible (appelé *Peripheral*) pour indiquer sa présence et à les émettre plus rapidement que l'équipement légitime, forçant le périphérique initiant la connexion (appelé *Central*) à se connecter sur l'équipement cloné contrôlé par l'attaquant. L'approche adoptée par *BTLEJuice* initie directement une connexion avec le périphérique cible, l'obligeant à arrêter l'émission de trames d'*advertising*, puis expose un clone du *Peripheral* au *Central*. Ces deux stratégies sont basées

sur l'usurpation de trames d'*advertising* : par conséquent, elles ne peuvent effectuer une attaque de type *Man-in-the-Middle* que si la connexion n'est pas déjà établie.

Plusieurs travaux ont également étudié la sécurité des mécanismes de chiffrement et d'authentification définis par le protocole BLE. En 2013, M. Ryan a présenté *CRACKLE* [18], un outil exploitant une faiblesse de la première version du processus d'appairage pour *bruteforcer* facilement les clés impliquées dans la sécurité du *mode connecté*. Dans [1], Antonioli et al. introduisent une attaque nommée *KNOB* (*Key Negotiation of Bluetooth*), permettant de diminuer l'entropie de la clé de 16 à 7 octets, réduisant ainsi considérablement le coût d'une attaque par force brute. Dans [2], les auteurs analysent le mécanisme nommé *Cross-Transport Key Derivation*, destiné à permettre le partage de clés entre Bluetooth BR/EDR et BLE, et présentent quatre attaques nommées *BLUR* basées sur le détournement de cette fonctionnalité. Ces attaques permettent d'usurper l'identité d'un appareil, manipuler le trafic ou établir une session malveillante. De même, Wu et al. ont présenté dans *BLESA* [21] une attaque active détournant le processus de reconnexion d'un équipement de type *Central* déjà appairé pour se faire passer pour l'équipement de type *Peripheral* correspondant et transmettre des données malicieuses non chiffrées. Von Tschirschnitz et al. ont présentés une stratégie d'attaque par confusion [20] visant à établir un appairage entre deux équipements en les forçant à utiliser des méthodes différentes. Bien que certaines de ces attaques puissent être utilisées pour usurper l'identité d'un équipement, aucune d'entre elles n'est utilisable dans le cadre d'une connexion établie.

Des recherches antérieures se sont également concentrées sur la découverte de vulnérabilités liées aux implémentations plutôt qu'à la spécification du protocole. Nous pouvons par exemple citer *Blueborne* [3] en 2017, ou *BleedingBit* [4] en 2018. Dans [14], Garbelini et al. ont présenté un outil de fuzzing nommé *SweynTooth* ciblant différentes piles protocolaires BLE et qui leur a permis de découvrir une douzaine de vulnérabilités. Malgré leur impact élevé, elles sont cependant liées à des implémentations spécifiques et ne peuvent donc pas être généralisées.

À notre connaissance, aucune des recherches existantes dans ce domaine ne s'est concentrée sur l'injection de trames malveillantes dans une connexion existante sans impliquer la déconnexion d'au moins un des deux équipements communicants. Dans [19], Santos et al. émettent l'hypothèse qu'il serait trop difficile de mettre en place une attaque par injection dans une communication BLE, et ont donc rejeté la possibilité d'une telle approche. Cependant, comme nous le démontrerons plus loin et

l'illustrerons expérimentalement, une telle attaque est pourtant possible. Nous démontrons également que cette approche peut être utilisée pour exécuter de nouveaux scénarios d'attaque qui n'ont pas encore été explorés, comme le détournement du rôle *Slave* ou l'exécution d'une attaque de type *Man-in-the-Middle* pendant une connexion établie.

3 Défis et modèle de menaces

Dans cet article, nous explorons un nouveau type d'attaque ciblant le protocole Bluetooth Low Energy, permettant l'injection de trames arbitraires dans une connexion établie. Le protocole BLE fournit un *mode connecté* permettant aux équipements impliqués de communiquer uniquement lors d'une fenêtre de réception ouverte dans un intervalle de temps précis, ce qui rend les attaques par injection difficiles à réaliser par conception. Selon la spécification [5], les équipements impliqués peuvent cependant élargir la fenêtre de réception pour compenser les dérives et désynchronisations d'horloges. Par effet de bord, cela ouvre également la possibilité à un attaquant de détourner cette fonctionnalité pour effectuer une attaque de type *race condition* (voir Figure 1). Nous avons concentré nos travaux sur l'analyse de la faisabilité d'une telle injection, et exploré des techniques permettant de résoudre les défis suivants :

- **(C1) identifier le moment propice pour injecter une trame ;**
- **(C2) générer une trame cohérente avec l'état actuel de la connexion ;**
- **(C3) valider si l'attaque a réussi ou a échoué.**

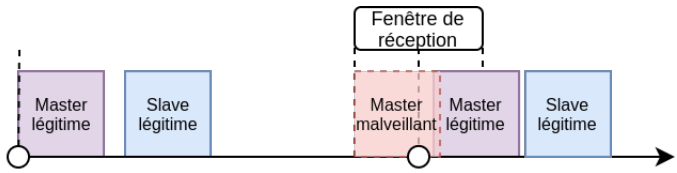

Fig. 1. Présentation de la stratégie d'injection

D'un point de vue offensif, l'attaque présentée dans cet article a un impact considérable : en effet, si plusieurs attaques visant la sécurité BLE ont déjà été étudiées dans des travaux antérieurs, aucune n'a permis d'interférer avec une connexion déjà établie sans interrompre la communication pour au moins l'un des équipements concernés. Les résultats présentés

dans cet article montrent qu'une telle attaque est possible et peut potentiellement être utilisée pour exécuter un large éventail de scénarios offensifs critiques, allant de l'utilisation illégitime des fonctionnalités de l'équipement ciblé à l'usurpation d'un rôle de la connexion. Nous considérons que cette nouvelle capacité offensive peut avoir un impact conséquent sur la disponibilité, la confidentialité et l'intégrité d'une communication BLE établie. En effet, la vulnérabilité présentée dans cet article est liée à la spécification du protocole lui-même, de sorte que tout équipement BLE est potentiellement vulnérable, indépendamment de l'implémentation de sa pile protocolaire. La menace est d'autant plus sérieuse que l'attaque est relativement simple à implémenter et peut être effectuée dès qu'un attaquant est à portée radio de la connexion ciblée. L'attaque est également compatible avec toutes les versions de BLE, de 4.0 à 5.2. Le modèle de menaces considéré est le suivant :

— l'attaquant doit être à portée radio des équipements impliqués,
— l'attaquant utilise seulement des puces standards supportant le BLE,
— l'attaquant est capable d'écouter passivement les communications BLE, ainsi que de forger et transmettre des trames arbitraires,
— l'attaquant n'a pas besoin d'exploiter une vulnérabilité de l'équipement ciblé.

Dans cet article, nous nous focalisons sur l'injection de trames au sein d'une communication non chiffrée. En effet, la plupart des communications BLE à l'heure actuelle n'utilisent pas ou peu les mécanismes de sécurité proposé par la spécification [22]. Cependant, la fonctionnalité nommée *window widening* menant à la vulnérabilité que nous exploitons est indépendante de l'utilisation de ces mécanismes de sécurité : il serait ainsi théoriquement possible d'exploiter la vulnérabilité même en présence d'une communication chiffrée. Si l'attaquant parvient à connaître la *Long Term Key*, alors il sera capable de forger une trame chiffrée valide et pourra donc reproduire les différents scénarios exposés dans cet article. S'il ne connaît pas cette clé, il ne peut générer des paquets valides mais il peut malgré tout mener une attaque de déni de service, par injection de paquets invalides, en exploitant la vulnérabilité présentée dans cet article.

4 Bluetooth Low Energy

Cette section présente un bref aperçu du protocole BLE, ainsi qu'une description plus détaillée de sa couche Liaison. Nous introduisons no-

tamment un certain nombre de mécanismes utilisés par le protocole et nécessaires à la compréhension de la stratégie d'injection.

4.1 Vue d'ensemble du protocole

Le Bluetooth Low Energy est une version simplifiée du Bluetooth BR/EDR, dédiée à des objets connectés fonctionnant sur batterie et nécessitant donc des communications moins gourmandes en énergie.

Sa pile protocolaire est séparée en deux parties : la partie *Controller* et la partie *Host*. Les couches basses sont gérées par la partie *Controller*, tandis que les couches hautes sont gérées par la partie *Host*.

La couche physique du protocole se base sur une modulation de fréquence dite *Gaussian Frequency Shift Keying*. Trois modes sont disponibles en BLE : une couche physique non codée avec un débit binaire de 1 Mbit/s ou de 2Mbit/s (respectivement appelées mode *LE 1M* et mode *LE 2M*), ainsi qu'une couche physique incluant du codage canal, à un débit binaire utile de 250 kbit/s ou 500 kbit/s (appelée mode *LE Coded*).

Le BLE opère dans les bandes ISM de 2.4 à 2.5 GHz, et y définit 40 canaux d'une largeur de 2 MHz. Trois canaux (37, 38 et 39) sont dédiés au mode *advertising* (permettant aux objets de diffuser des données en *broadcast* en utilisant des paquets appelés *advertisements*), tandis que les 37 autres canaux (numérotés de 0 à 36) sont dédiés au *mode connecté*, utilisé lorsqu'une connexion est établie entre deux objets.

Toutes les applications du BLE utilisant ce *mode connecté* se basent sur les couches *ATT* et *GATT*. Ces couches définissent un modèle de type client serveur, et fournissent une solution générique pour communiquer des informations entre équipements.

Plus spécifiquement, un serveur *ATT* contient une base de données d'*attributs* ; chaque *attribut* est composé d'un identifiant (ou *handle*), d'un type et d'une valeur. Un client *ATT* est capable d'interagir avec cette base de données en utilisant différentes requêtes ; par exemple, une *Read Request* permet au client de lire un *attribut* donné, tandis qu'une *Write Request* permet de modifier la valeur d'un *attribut*. La couche *GATT* ajoute une couche d'abstraction supplémentaire en définissant des *services* contenant différentes *caractéristiques*, et en créant ainsi des profils génériques pour différents types d'objets.

Le *Security Manager* fournit des procédures d'*appairage* et de *bonding* permettant de négocier différentes clés destinées à sécuriser les connexions. L'une des clés les plus importantes est la *Long Term Key*, qui permet d'établir une connexion chiffrée utilisant le chiffrement AES-CCM.

Le *Generic Access Profile* (*GAP*) introduit quatre rôles différents, chacun décrivant un profil d'objet différent. Deux de ces rôles concernent le *mode connecté* : le rôle *Peripheral* correspond à un objet pouvant diffuser des advertisements et auquel on peut se connecter, tandis que le rôle *Central* correspond à un objet pouvant recevoir ces advertisements et établir une connexion avec un autre objet.

Le *Peripheral* est aussi appelé *Slave*, car il joue le rôle d'esclave dans une connexion BLE ; le *Central* est aussi appelé *Master*.

4.2 Présentation de la couche Liaison

Notre stratégie d'injection se base principalement sur l'exploitation de certaines caractéristiques spécifiques de la couche Liaison du BLE. Cette sous-section fournit une description détaillée de ces caractéristiques.

Format des trames Chaque trame BLE transmise en utilisant le mode *LE 1M* est construite suivant le format décrit par la figure 2.

Préambule	Access Address	Protocol Data Unit (PDU)	CRC
1 octet	4 octets	variable	3 octets

Fig. 2. Format de trames pour le mode *LE 1M*

Le préambule est utilisé par le récepteur pour localiser le début d'une trame BLE. L'*Access Address* indique le mode utilisé, *mode connecté* ou *advertising*.

Init. Address	Adv. Address	Access Address	CRC Init	WinSize	WinOffset	Hop Interval	Latency	Timeout	Channel Map	Hop Increment	SCA
6 octets	6 octets	4 octets	3 octets	1 octet	2 octets	2 octets	2 octets	2 octets	5 octets	5 bits	3 bits

Fig. 3. PDU d'un CONNECT_REQ

Etablissement d'une connexion Quand un *Peripheral* n'est pas connecté à un *Central*, il diffuse des advertisements sur les canaux dédiés. Le payload contient en général des informations permettant d'identifier rapidement l'objet, comme son nom ou son constructeur.

Pour établir une connexion avec un *Peripheral*, le *Central* émet un advertisement dédié appelé *CONNECT_REQ* juste après la réception d'un advertisement de ce *Peripheral*. Le payload de niveau Liaison (ou *Link Layer PDU*) correspondant, décrit en figure 3, inclut des paramètres utilisés par la connexion. Le champ *Access Address* est utilisé par les deux acteurs pour chacune des trames échangées en mode connecté.

Sélection des canaux La *Channel Map* et le *Hop Increment* (voir figure 3) sont utilisés par l'algorithme de sélection de canaux. Une connexion BLE utilise en effet un mécanisme de saut de fréquences pour éviter les interférences avec les autres connexions BLE, ainsi que d'autres protocoles de communication sans-fil.

Deux algorithmes de sélection de canaux sont actuellement utilisables : *Channel Selection Algorithm #1*, qui est basé sur une simple addition modulaire, et *Channel Selection Algorithm #2*, basé sur un générateur de nombres pseudo-aléatoires. Ces deux algorithmes donnent des résultats qu'un attaquant pourra potentiellement prédire pour suivre une connexion établie (voir [17] et [10]). Dans cet article, nous nous sommes concentrés sur l'algorithme *Channel Selection Algorithm #1*, qui est le plus couramment utilisé ; cependant, l'approche proposée peut être aisément adaptée à l'autre algorithme.

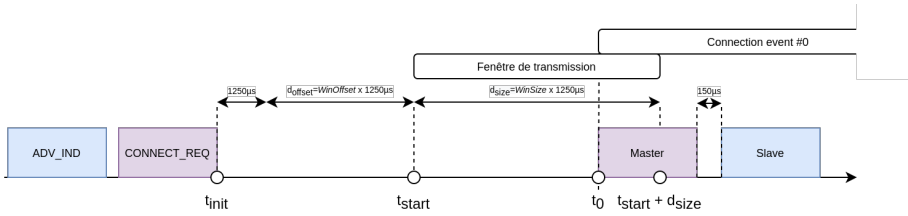

Fig. 4. Etablissement d'une connexion BLE

Fenêtre de transmission Les champs *WinSize* et *WinOffset* (voir Figure 3) sont utilisés pour définir la *fenêtre de transmission*. En effet, la première trame de la connexion est transmise sur le premier canal choisi par le *Central* à l'instant t_0 durant une *fenêtre de transmission* définie par la formule 1 :

$$\begin{cases} t_{start} \leq t_0 \leq t_{start} + d_{size} \\ t_{start} = t_{init} + 1250\mu s + d_{offset} \end{cases} \quad (1)$$

où :

— t_{init} est l'instant de la fin de transmission de la trame *CONNECT_REQ*,
— $d_{offset} = WinOffset \times 1250\mu s$
— $d_{size} = WinSize \times 1250\mu s$

t_0 indique le début du premier *connection event*, et est utilisé comme temps de référence pour les prochains *connection events*. Cette phase initiale est illustrée par la Figure 4.

Connection events Considérons un *connection event* commençant à l'instant t_n, appelé *anchor point* et correspondant au début de transmission par le *Master* d'une trame destinée au *Slave*. t_0 correspond au premier *anchor point*. Quand le *Slave* reçoit la trame, il attend durant une période de 150 μs appelée *durée inter-trames* (ou *inter-frame spacing*) avant d'envoyer à son tour une trame au *Master*. Un bit appelé *More Data* (*MD*) dans l'en-tête des trames lui permet d'indiquer si plus de données doivent être envoyées durant le *connection event*. Si l'équipement n'a pas de données à transmettre, il enverra une trame vide.

La période entre deux *anchor points* consécutifs est donnée par la valeur du *Hop Interval*, selon la formule 2 :

$$d_{connInterval} = HopInterval \times 1250\mu s \tag{2}$$

Chaque fois qu'un *connection event* se termine, le prochain canal est choisi selon l'algorithme de sélection de canaux utilisé. Chaque *connection event* est aussi identifié par un entier non signé de 16 bits appelé *connection event count*. La Figure 5 illustre le fonctionnement de deux *connection events* consécutifs.

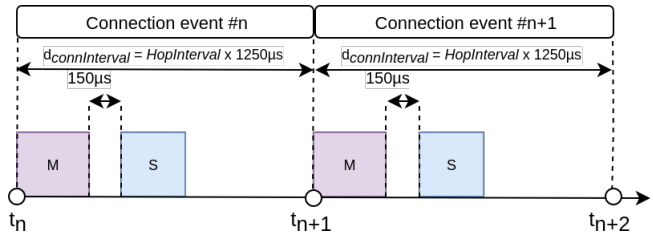

Fig. 5. Deux *connection events* consécutifs

Acquittements et contrôle de flux Chaque trame BLE transmise durant une connexion contient deux champs de 1 bit dans l'en-tête de son payload de niveau liaison (*LL PDU*), indiquant respectivement le *Sequence Number* (*SN*) et le *Next Expected Sequence Number* (*NESN*). Chaque équipement possède également deux compteurs sur 1 bit, respectivement nommés *transmitSeqNum* et *nextExpectedSeqNum*. Le compteur *transmitSeqNum* est incrémenté de un (modulo 2) si la trame précédemment émise a été acquittée par l'autre acteur de la connexion. Le compteur *nextExpectedSeqNum* est incrémenté de un (modulo 2) si la trame attendue (correspondant à la valeur du compteur) a été reçue. Un exemple d'évolution des compteurs et valeurs des bits *SN* et *NESN* au cours d'un échange est représenté dans la Figure 6.

Fig. 6. Evolution des compteurs et des bits SN et NESN

Modification des paramètres en cours de connexion Le protocole BLE offre la possibilité de modifier les paramètres utilisés par l'algorithme de sélection de canaux en cours de connexion. Un *Master* est généralement capable de gérer plusieurs connexions simultanément, et peut avoir besoin

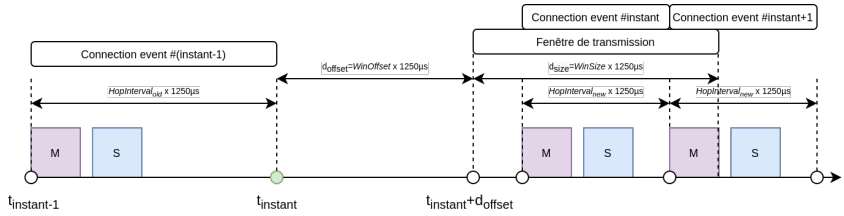

Fig. 7. Procédure de *connection update*

de modifier les paramètres d'une connexion en cours pour optimiser le suivi d'autres connexions. Il peut également considérer un canal comme bruité en cas de taux de pertes de trames élevé sur celui-ci, et décider de ne plus l'utiliser. La couche Liaison fournit deux trames de contrôle, *CONNECT_UPDATE_IND* et *CHANNEL_MAP_IND*, permettant de modifier respectivement le *Hop Interval* et la *Channel Map*.

Ces trames contiennent la nouvelle valeur du champ à modifier, ainsi qu'une valeur sur deux octets appelée *instant*. Quand le *Slave* reçoit une de ces trames, il enregistre dans ses paramètres la valeur concernée, puis attend que le *connection event count* atteigne la valeur *instant*. Ensuite :

— Dans le cas d'une trame *CONNECT_UPDATE_IND*, une fenêtre de transmission similaire à celle utilisée à l'établissement de la connexion est calculée à partir des valeurs de *WinOffset* et *WinSize* fournies par cette trame. Une fois cette phase de resynchronisation réalisée, les nouveaux paramètres sont utilisés.

— Dans le cas d'une trame *CHANNEL_MAP_IND*, la nouvelle *Channel Map* sera utilisée pour les prochains *connection events*.

Slave latency La *Slave Latency*, initialement proposée par le *Master* dans la trame *CONNECT_REQ* (voir Figure 3) et pouvant être modifiée à l'aide de la procédure de *connection update*, permet au *Slave* de ne pas écouter chaque *connection event* pour limiter sa consommation énergétique. Ainsi, une *slave latency* égale à trois permet au *Slave* d'ignorer jusqu'à trois *connection events* consécutifs sans provoquer la terminaison de la connexion, comme illustré par la figure 8.

5 InjectaBLE : injection de trafic dans une connexion établie

Dans cette section, nous présentons l'attaque *InjectaBLE*, permettant d'injecter des trames arbitraires dans une connexion établie. Celle-ci

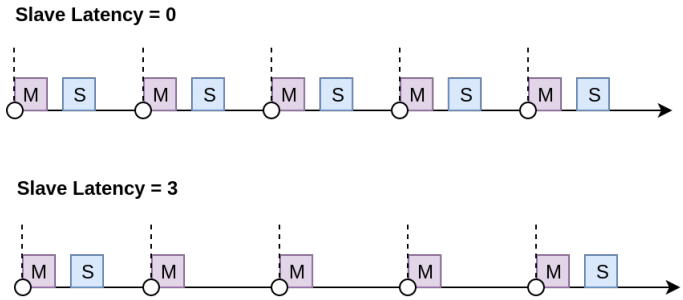

Fig. 8. Présentation du mécanisme de Slave Latency

nécessite d'identifier un instant précis où une trame peut être injectée avec succès, que nous appelons *point d'injection*. Les sous-sections 5.1 et 5.2 décrivent certaines caractéristiques de la couche liaison du protocole nous permettant d'identifier un tel point d'injection (défi **C1** de la section 3). La sous-section 5.3 décrit comment injecter une trame cohérente avec l'état actuel de la connexion (défi **C2**), tandis que la sous-section 5.4 décrit la conception d'une heuristique destinée à vérifier le succès de l'injection (défi **C3**).

5.1 Clock (in)accuracy

Comme mentionné précédemment, le début de transmission d'une trame du *Master* dans un *connection event* donné est utilisé comme instant de référence, ou *anchor point*. Théoriquement, pour un *anchor point* t_n, le prochain *anchor point* t_{n+1} peut être calculé selon la formule 3 :

$$t_{n+1} = t_n + d_{connInterval} \tag{3}$$

Un attaquant ne peut pas injecter de trame à cet instant spécifique, étant donné que la trame injectée provoquerait inévitablement une collision avec la trame du *Master* légitime. Cependant, les équipements légitimes impliqués dans une connexion établie utilisent plusieurs *timers* basés sur une horloge spécifique nommée *Sleep Clock*. Cette horloge étant susceptible de présenter une dérive temporelle, le *Slave* ne peut faire l'hypothèse que sa *Sleep Clock* est parfaitement synchronisée avec le *Master*. Par conséquent, le *Slave* compense cette dérive potentielle en écoutant un peu avant et un peu après l'instant théorique de l'*anchor point*.

5.2 Window widening

La spécification introduit la notion de *window widening*, qui consiste à étendre la durée d'écoute d'un équipement donné pour compenser la potentielle dérive des horloges. Dans le cas spécifique de la couche liaison d'un *Slave* attendant le prochain *connection event*, le *window widening* peut être calculé à l'aide de la formule 4.

$$w = \frac{SCA_M + SCA_S}{1000000} \times (t_{nextAnchor} - t_{lastAnchor}) + 32\mu s \qquad (4)$$

avec :

- SCA_M : *sleep clock accuracy* de la couche liaison du *Master* (en ppm),
- SCA_S : *sleep clock accuracy* de la couche liaison du *Slave* (en ppm),
- $t_{nextAnchor}$: prochain *anchor point* théorique (en μs),
- $t_{lastAnchor}$: dernier *anchor point* observé (en μs).

Si le *Slave* transmet une trame à chaque *connection event* (soit une *Slave Latency* égale à 0), la formule peut être réécrite sous la forme 5 :

$$w = \frac{SCA_M + SCA_S}{1000000} \times d_{connInterval} + 32\mu s \qquad (5)$$

Si la *Slave latency* est supérieure à 0, l'intervalle entre le dernier *anchor point* observé et le prochain *anchor point* théorique augmente, ce qui a donc pour conséquence l'élargissement de la fenêtre de réception. Dans ce cas, l'équation 5 définit la valeur minimale du *window widening*.

En conséquence, pour un *anchor point* théorique t_{n+1}, le *Slave* acceptera le paquet du *Master* initiant le *connection event* si celui-ci a été transmis pendant la fenêtre de réception entre l'instant $t_{n+1} - w$ et l'instant $t_{n+1} + w$, comme illustré en figure 9.

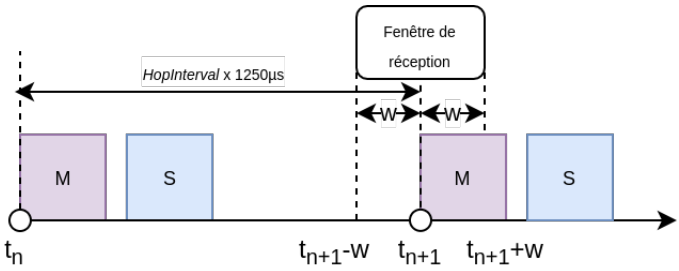

Fig. 9. Illustration du *Window widening* à la réception d'un *connection event*

5.3 Injection de paquet arbitraire

Une trame transmise dans la fenêtre de réception étant considérée comme un paquet du *Master* par le *Slave*, il est possible d'exploiter une *race condition* pour injecter une trame arbitraire dans une connexion établie en transmettant celle-ci au début de la fenêtre de réception.

Pour que cette injection puisse être réalisée, l'attaquant doit tout d'abord se synchroniser avec la connexion ciblée. Comme souligné précédemment, plusieurs approches existantes [8, 10, 17] permettent de sniffer une connexion passivement et peuvent être réutilisées dans le cadre de cette attaque. L'attaquant doit ensuite forger une trame à injecter qui soit cohérente avec l'état actuel de la connexion. Cette trame sera considérée par le *Slave* comme une nouvelle donnée si le numéro de séquence de la trame (noté SN_a) est égal au compteur *Next Expected Sequence Number* du *Slave* (noté $NESN_s$). Le bit $NESN$ dans la trame injectée (noté $NESN_a$) doit également être fixé de sorte à indiquer que la dernière trame transmise par le *Slave* (noté SN_s) a été reçue avec succès. Par conséquent, l'attaquant doit avoir observé dans le *connection event* précédant la tentative d'injection la trame transmise par le *Slave* et en avoir extrait les bits SN_s et $NESN_s$. Les bits SN_a et $NESN_a$ inclus dans la trame injectée doivent être fixés selon l'équation 6 :

$$\begin{cases} SN_a = NESN_s \\ NESN_a = (SN_s + 1) \mod 2 \end{cases} \tag{6}$$

Finalement, l'attaquant doit estimer la fenêtre de réception afin de transmettre la trame au plus tôt durant celle ci. Il peut utiliser l'équation 5 pour estimer le *window widening*. La *Sleep Clock Accuracy* du *Master* peut être extraite du paquet *CONNECT_REQ* d'initiation de connexion ou de paquets de contrôle embarquant cette information (tels que les paquets *LL_CLOCK_ACCURACY_REQ* ou *LL_CLOCK_ACCURACY_RSP*). La *Sleep Clock Accuracy* du *Slave* peut être estimée à 20ppm, ce qui constitue le pire cas de figure du point de vue de l'attaquant.

5.4 Vérification du succès de l'injection

Pour pouvoir mettre en place des attaques nécessitant l'injection de trames multiples, l'attaquant doit être en mesure d'estimer si l'injection a été réalisée avec succès ou non. Une telle vérification n'est pas triviale dans la mesure où une injection réussie ne provoque pas systématiquement un changement observable dans le comportement du *Slave* ayant reçu la trame.

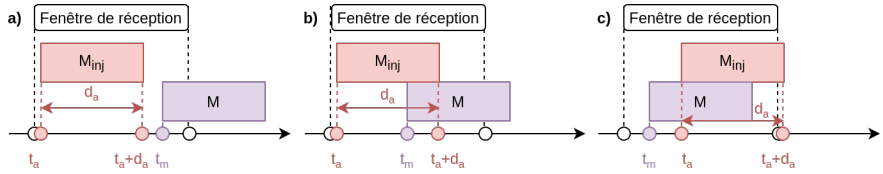

Fig. 10. Trois situations possibles pour une tentative d'injection

Ainsi, il est nécessaire de définir une heuristique, basée uniquement sur l'observation des paramètres de la couche liaison, permettant de déterminer le succès ou non de l'injection.

Une trame injectée est considérée valide par le *Slave* si :

— la trame injectée est transmise avant la trame du *Master* dans la fenêtre de réception,

— le CRC embarqué dans la trame injectée est égal au CRC calculé à la réception de la trame.

Considérons une tentative d'injection avec t_a le début de transmission de la trame injectée, d_a la durée de cette transmission et t_m le début de transmission de la trame du *Master* légitime.

Une tentative d'injection peut amener à trois situations différentes, illustrées en figure 10 :

a. La trame injectée est transmise dans la fenêtre de réception avant le début de transmission de la trame légitime ($t_a + d_a < t_m$)

b. La trame injectée est transmise dans la fenêtre de réception, mais la fin de la trame entre en collision avec la trame légitime ($t_a + d_a \geq t_m$)

c. La trame légitime est transmise avant la trame injectée ($t_a \geq t_m$)

Dans la situation a), la tentative d'injection est réussie, les deux conditions étant remplies. La situation b) peut résulter en une injection réussie si la collision ne corrompt pas la trame injectée, dans le cas contraire le CRC sera invalide et la tentative d'injection échoue. La situation c) mène à un échec de la tentative d'injection, la première condition n'étant pas remplie.

Dans la mesure où une tentative d'injection est susceptible d'échouer ou non en fonction de la situation, l'attaquant peut construire une heuristique permettant d'estimer le succès d'une injection donnée. Cette heuristique se base sur les deux conditions précédemment mentionnées :

— La trame injectée est transmise dans la fenêtre de réception avant le début de transmission de la trame légitime : une observation directe de la trame légitime transmise par le *Master* n'est généralement pas

possible car l'attaquant transmet sa propre trame en même temps. Cependant, la réponse du *Slave* peut être utilisée pour inférer indirectement cette information. En effet, si la trame injectée a été transmise avant la trame légitime, le *Slave* va considérer le début de transmission de la trame injectée comme le nouveau *anchor point*. En conséquence, il transmettra sa propre trame 150 μs après la fin de transmission de la trame injectée. Si on note t_s le début de transmission de la réponse du *Slave*, on peut exprimer cette condition ainsi :

$$t_a + d_a + 150 - 5 < t_s < t_a + d_a + 150 + 5$$

Nous avons empiriquement estimé une fenêtre de $10\mu s$ de large, résultant dans les $5\mu s$ dans la formule précédente. Cette estimation a été établie en injectant des paquets précis ayant un impact observable sur l'équipement de type *Slave* (par exemple générant une réponse du *Slave* ou une terminaison de la connexion).

— le CRC embarqué dans la trame injectée est égal au CRC calculé à la réception de la trame : pour les mêmes raisons que ci-dessus, l'attaquant ne peut pas directement vérifier si une collision s'est produite et a corrompu la trame injectée car il n'est pas en mesure d'écouter le canal durant la tentative d'injection. Cependant, la réponse du *Slave* peut également être utilisée pour inférer cette information, car si la trame a été reçue par le *Slave* avec un CRC embarqué qui ne correspond pas à celui calculé, ce dernier ne va pas incrémenter le compteur *nextExpectedSeqNum* afin d'indiquer au *Master* que la dernière trame reçue doit être retransmise. En conséquence, le champ *NESN* de la réponse du *Slave* est identique à celui utilisé de la dernière réponse précédente. Si on note SN'_s le champ *SN* et $NESN'_s$ le champ *NESN* inclus dans la réponse du *Slave*, cette condition peut être exprimée ainsi :

$$((SN_a + 1) \mod 2 = NESN'_s) \wedge (NESN_a = SN'_s)$$

Au final, l'heuristique permettant de détecter le succès de l'injection peut être exprimée par la condition 7 :

$$\begin{aligned} (t_a + d_a + 150 - 5 < t_s < t_a + d_a + 150 + 5) \wedge \\ ((SN_a + 1) \mod 2 = NESN'_s) \wedge (NESN_a = SN'_s) \end{aligned} \tag{7}$$

avec :

— t_a le début de transmission de la trame injectée,
— d_a la durée de transmission de la trame injectée,
— t_s le début de transmission de la réponse du *Slave*,
— SN'_s le champ *SN* de la réponse du *Slave*,
— $NESN'_s$ le champ *NESN* de la réponse du *Slave*.

Notons qu'il serait également possible d'utiliser un second *sniffer* afin de monitorer la communication et d'estimer le succès de l'injection par l'observation directe du trafic. Cette solution nécessiterait cependant une synchronisation temporelle des deux équipements offensifs et pourrait être limitée en cas de collision par une potentielle différence de perception du trafic reçu entre le *Slave* et le *sniffer*.

5.5 Implémentation

Nous avons développé une preuve de concept permettant de facilement mener cette attaque et de l'évaluer. Celle-ci a été implémentée sur un dongle destiné au développement IoT embarquant une puce *nRF52840* de *Nordic SemiConductors*. Cette puce a été choisie pour son support du BLE 5.0 et l'accès relativement bas niveau au composant radio qu'elle autorise, facilitant l'implémentation de l'attaque.

Le dongle communique avec le *Host* par l'intermédiaire d'un protocole USB permettant de transmettre des commandes au firmware. Un sniffer BLE léger, basé sur les travaux [8, 17] et [16], a été réimplémenté, permettant de synchroniser le dongle avec une connexion donnée. Quand une nouvelle connexion est détectée par le dongle, il se synchronise avec l'algorithme de saut de fréquence utilisée par cette dernière et transmet les paquets reçus au *Host*. A tout moment, l'utilisateur peut transmettre une série de commandes au dongle, permettant de déclencher l'injection d'une trame au sein de la communication :

a. Avant l'injection, le *window widening* utilisé est estimé à l'aide de la formule 5,
b. Le dongle réalise la tentative d'injection dès que possible dans la fenêtre de réception précédemment définie,
c. L'heuristique définie par la formule 7 est utilisée pour vérifier si la tentative d'injection a réussi ou échoué,
d. Si la tentative d'injection a échoué, une nouvelle tentative d'injection est préparée,
e. Si la tentative d'injection a réussi, une notification est transmise au *Host*, indiquant le nombre de tentatives d'injections échouées avant une injection réussie.

Le dongle expose également une API permettant de lancer les différents scénarios décrits en section 6. Une pile protocolaire *BLE* simplifiée a été implémentée, permettant d'imiter le comportement des différents rôles impliqués dans une connexion.

Avec l'autorisation du *Bluetooth SIG*, qui a été notifié de la vulnérabilité, nous mettons à disposition l'outil sous licence libre dans le dépôt [11].

6 Scénarios d'attaques

Cette section décrit quatre scénarios d'attaque réalistes que nous avons imaginés et expérimentés. Ces attaques permettent d'injecter des commandes, mais aussi d'usurper le rôle du *Slave* ou du *Master*, et enfin de réaliser des attaques du type *Man in the Middle*.

6.1 Scénario A : utiliser de façon illégitime une fonctionnalité d'un objet

Cette première attaque est une utilisation directe du mécanisme d'injection décrit dans cet article. En effet, les objets BLE implémentent en grande majorité le rôle de *Slave* et notre attaque par injection peut être utilisée simplement pour activer une fonctionnalité spécifique d'un objet, au cours d'une connexion déjà établie avec un *master* légitime. Plus précisément, nous ciblons dans notre attaque le protocole ATT et montrons que cette attaque permet d'injecter avec succès des ATT *Requests*.

Un attaquant peut par exemple injecter une *Read Request* ciblant un handle spécifique : si l'injection fonctionne, le *Slave* va générer et transmettre une *Read Response* qui contient les données correspondantes. Cette attaque peut permettre à l'attaquant d'accéder à des informations intéressantes relatives à une caractéristique donnée. En fonction du type d'équipement visé, cette attaque peut être une atteinte sérieuse à la confidentialité.

De façon similaire, un attaquant peut injecter une *Write Request* ou une *Write Command*. Ces requêtes permettent de modifier la valeur d'une caractéristique donnée de l'objet ciblé. L'attaquant peut ainsi déclencher un comportement spécifique de l'objet, ce qui peut avoir de graves conséquences sur son intégrité ou sa disponibilité.

Pour illustrer l'impact de ce scénario d'attaque, nous avons réalisé des attaques par injection ciblant trois objets connectés grand public : une ampoule, un porte-clés et une montre connectée. Nous avons réalisé une rétro-conception de ces objets afin d'identifier les principales ATT

Requests et leurs payloads qui régissent le comportement des objets. Nous avons ensuite forgé et injecté du trafic malveillant correspondant aux requêtes suivantes :

— ampoule : allumer ou éteindre, changer la couleur, changer la luminosité,
— porte-clés : le faire sonner,
— montre : envoyer un SMS forgé à la montre

6.2 Scénario B : usurpation du rôle du *Slave*

Ce second scénario permet à l'attaquant d'usurper le rôle du *Slave*. L'attaque consiste à déconnecter le *Slave* légitime et à prendre sa place sans pour autant provoquer une déconnexion du point de vue du *Master*.

L'attaque repose sur l'injection de paquets de contrôle niveau Liaison, qui sont utilisés par les objets pour contrôler les connexions sur la couche liaison. L'attaque consiste à injecter un paquet *LL_TERMINATE_IND* qui est utilisé par un équipement pour mettre fin à une connexion. Ce paquet est ignoré par le *Master* mais accepté par le *Slave* et force donc le *Slave* à quitter la connexion. Cependant, comme le *Master* ne sait pas que le *Slave* légitime n'est plus connecté, l'attaquant peut poursuivre la connexion en prenant sa place. Pour cela, l'attaquant doit attendre pendant la durée inter-trames (150 μs) après la fin de la transmision du *Master* avant d'émettre sa trame en veillant à bien positionner les champs *SN* et *NESN*. Cette attaque est illustrée en figure 11.

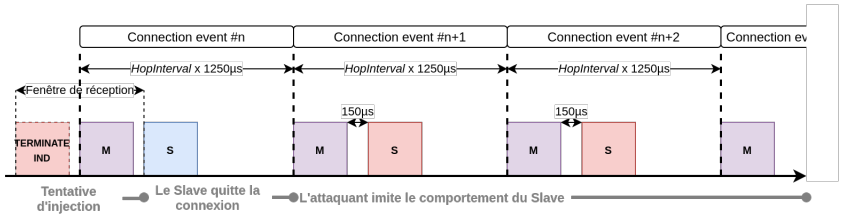

Fig. 11. Scénario d'usurpation du rôle *Slave*

Ce scénario d'attaque a été exécuté avec succès sur les trois objets mentionnés dans la section 6.1. Ces trois objets exposent une caractéristique *Device Name* pour laquelle nous avons forgé une valeur "Hacked" lorsqu'une requête en lecture pour cette caractéristique était reçue. Un tel scénario d'attaque peut avoir de sérieuses conséquences en fonction du type de l'objet considéré. Par exemple, l'attaquant pourrait attaquer des

objets tels que des pompes à insuline ou des pacemakers et envoyer des données falsifiées, mettant ainsi en danger la vie de patients.

6.3 Scénarios C et D : usurpation du rôle du *Master*, du *Slave* ou des deux simultanément (attaque de type *Man-in-the-Middle*)

Nous avons exploré deux autres scénarios, basés sur la même approche. Le scénario C consiste à usurper le rôle du *Master*. Même si ce type d'attaque a déjà été présenté, notamment dans l'outil *BTLEJack* [9], la stratégie d'attaque était basée sur du jamming et pouvait être facilement détectée par un outil de monitoring. Notre approche nécessite l'injection d'une seule trame malveillante, ce qui la rend particulièrement discrète et fiable. Le scénario D nous a permis de mener une attaque de type *Man-in-the-Middle* sans interrompre la connexion. Les approches existantes permettant de réaliser des attaques de type *Man-in-the-Middle* [7, 15] peuvent seulement être réalisées avant l'établissement de la connexion, ce qui limite grandement leur portée. En d'autres termes, en utilisant notre stratégie, un attaquant peut établir une attaque de type *Man-in-the-Middle* **à n'importe quel instant**, même si la connexion est déjà établie entre deux objets légitimes.

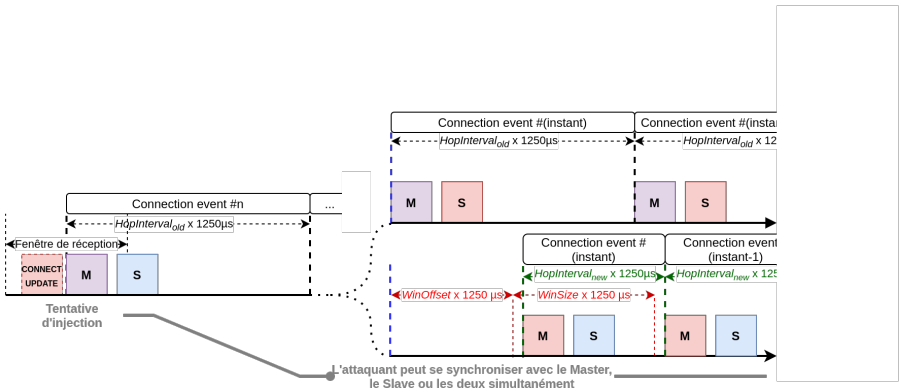

Fig. 12. Scénario de Man-in-the-Middle

Ces deux scénarios suivent une approche similaire basée sur l'injection d'un paquet de type *CONNECTION_UPDATE_IND*, comme décrit dans la section 4.2 : ce paquet peut être utilisé par le *Master* à n'importe quel instant pendant la connexion pour modifier les paramètres

de l'algorithme de sélection du canal, et particulièrement le *Hop Interval*. Cette attaque repose sur une idée simple : l'attaquant injecte un paquet *CONNECTION_UPDATE_IND* forgé contenant des paramètres arbitraires, indiquant au *Slave* que les paramètres de la connexion vont changer à un moment précis. Quand ce moment est atteint, le *Slave* attend pendant la durée *window offset* spécifiée par l'attaquant, ignorant la trame du *Master* légitime, et utilise ces nouveaux paramètres pendant que le *Master* continue à utiliser les anciens paramètres, ce qui permet à l'attaquant de se synchroniser avec le *Slave* et d'usurper le rôle de *Master* ou de se synchroniser avec les deux, réalisant ainsi un Man-in-the-Middle. Dans le premier cas (prise du rôle *Master*) le *Master* légitime ne reçoit plus aucune réponse après le moment où les paramètres ont été changés, et il quitte donc la connexion en raison d'un timeout. Notons que cette approche est particulièrement efficace parce-qu'elle peut être également utilisée pour usurper le rôle du *Slave*, comme lors du scénario B, puisque l'attaquant connaît à la fois les anciens et les nouveaux paramètres. Cette approche est illustrée sur la figure 12.

Nous avons évalué expérimentalement l'usurpation du rôle du *Master* avec les trois objets mentionnés précédemment : nous avons notamment pu obtenir les mêmes résultats que lors du scénario A. De façon similaire, le scénario D a été testé avec ces trois mêmes objets, nous permettant ainsi de modifier arbitrairement les données échangées entre les objets légitimes. Par exemple, un SMS transmis par le smartphone à la montre connectée a pu être modifié à la volée, ou les valeurs *RGB* décrivant la couleur de l'ampoule connectée ont également pu être modifiées à la volée.

7 Analyse de sensibilité

Nous avons mené une analyse de sensibilité pour valider notre approche. Notre objectif était double : tester la faisabilité de notre attaque dans un environnement réaliste et analyser les impacts de différents paramètres sur son taux de succès. En particulier, nous avons choisi trois paramètres : le *Hop Interval*, la taille du payload et la distance entre l'attaquant et la cible (le *Slave*). Nous avons dédié une expérimentation spécifique à chaque paramètre et l'impact a été évalué en calculant le nombre de tentatives d'injection pour obtenir une injection réussie.

Les différents objets testés dans cette analyse implémentent la version 4.2 de la spécification. Toutefois, le mécanisme de *window widening* est présent dans l'ensemble des versions existantes de la spécification, rendant la vulnérabilité théoriquement applicable à toutes les versions. L'ensemble

des expérimentations a été réalisé en sniffant les communications depuis leur initiation, celles-ci étant destinées à valider prioritairement la stratégie d'injection. Comme mentionné précédemment, il serait cependant tout à fait envisageable qu'un attaquant puisse inférer les paramètres de la communication pour se synchroniser avec une connexion existant [8, 17]. Notons cependant que certaines piles protocolaires compliquent considérablement l'inférence des paramètres en modifiant fréquemment le *Channel Map*, introduisant donc une difficulté technique supplémentaire pour mener l'attaque en pratique, bien que ce ne soit pas directement lié à notre stratégie d'injection.

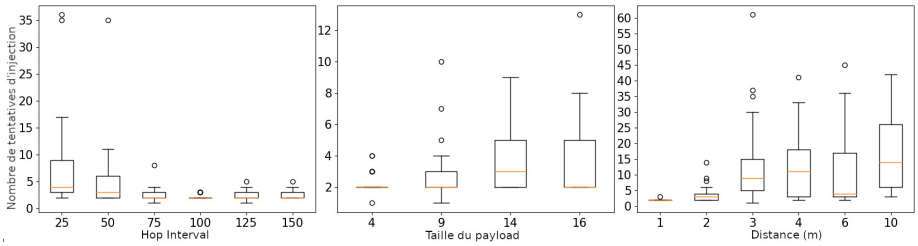

Fig. 13. Résultats de l'analyse de sensibilité

7.1 Expérimentation 1 : impact du *Hop Interval*

La première expérimentation est focalisée sur le paramètre *Hop Interval*. Ce paramètre est important puisqu'il est directement impliqué dans l'estimation du *window widening* comme indiqué dans l'équation 5. Théoriquement, comme l'attaque repose sur une *race condition* basée sur la valeur de cette fenêtre, l'injection devrait être plus difficile lorsque le *Hop Interval* diminue.

Selon les specifications, la valeur théorique du *Hop Interval* varie entre 6 et 3200. Cependant, nous avons choisi de faire varier ce paramètre entre six valeurs différentes, entre 25 et 150, pour deux raisons principales :

— Nous souhaitions volontairement nous placer dans les conditions les plus défavorables pour la tentative d'injection, ce qui correspond au cas ou la trame injectée est en collision avec la trame légitime, et ceci nous amène donc à tester des valeurs faibles pour le *Hop Interval*. La longueur de la trame injectée pendant cette expérimentation étant de 22 octets (i.e., 176 μs de durée de transmission en couche physique *LE 1M*), aucune des valeurs de *window widening* calculées

à partir des *Hop Intervals* testées ne permettent à une trame injectée d'être entièrement transmise sans collision.

— Nous souhaitions mener notre expérimentation sur de réels objets connectés du marché, et la majorité d'entre eux ne permet pas l'utilisation de valeurs élevées de *Hop Interval*, susceptibles de générer des connexions très instables. Nous avons donc choisi des valeurs de *Hop Interval* dans l'intervalle supporté par une ampoule connectée, qui se trouve être l'objet commercial supportant le plus grand intervalle de valeurs que nous ayons trouvé.

De façon à précisément positionner la valeur du *Hop Interval*, nous avons utilisé une version modifiée du framework open-source Mirage [12,13] pour simuler un objet *Central*, exploitant la capacité de l'outil à fournir un accès bas niveau à la couche *HCI*.

Nous avons réalisé une rétro-conception du protocole de communication au-dessus de la couche *GATT* utilisée par cette ampoule connectée, et sélectionné une *Write Request* permettant d'éteindre l'ampoule. Le payload correspondant a une longueur de 14 octets, générant donc une trame de 22 octets. Nous avons volontairement choisi une trame dont les effets sur l'objet sont observables, ce qui nous a permis de valider notre heuristique.

La configuration expérimentale était simple : le *Peripheral* et le *Central* légitimes ainsi que l'attaquant ont été placés aux trois sommets d'un triangle équilatéral de 2 mètres de côté. Le *Central* initie les connexions avec le *Peripheral* de façon périodique tandis que l'attaquant se synchronise avec ces connexions et déclenche l'injection lors d'un *connection event* spécifique. L'expérimentation a été menée dans un environnement réaliste, caractérisé par la présence de plusieurs autres objets BLE et plusieurs routeurs *WiFi*. Notons que synchroniser l'outil d'attaque avec une connexion n'est pas immédiat, en particulier dans un environnement bruité comme celui-ci. Pour chaque valeur de *Hop Interval*, nous avons réalisé 25 injections, et observé le nombre de tentatives nécessaires avant une injection réussie. Les résultats sont présentés dans la figure 13.

L'attaque a été concluante pour chaque connexion testée. La variance du nombre de tentatives infructueuses décroît rapidement entre 25 et 100, et se stabilise ensuite. De façon similaire, la médiane reste faible, en-dessous de 4. Ces résultats montrent tout d'abord que l'injection est toujours réalisable même avec des valeurs de *Hop Intervals* faibles, et le nombre d'injections nécessaires avant une injection réussie est généralement peu élevé. L'expérimentation confirme également que le *Hop Interval* a un impact significatif sur le succès de l'injection, l'injection étant plus fiable pour des valeurs élevées.

7.2 Expérimentation 2 : taille du *payload*

La deuxième expérimentation se concentre sur la taille du *payload* de la trame injectée, et vise à confirmer que l'injection de trames de petite taille accroît la probabilité de succès de l'injection.

Les conditions d'expérimentation sont similaires à celles de l'expérimentation précédente. Nous avons sélectionné quatre différentes tailles de *payload* : 4, 9, 14 et 16 octets, correspondant à des trames ayant un effet observable sur l'ampoule connectée (la déconnecter, l'éteindre ou changer sa couleur), permettant ainsi de vérifier le succès ou non de l'injection.

Nous avons fixé la valeur du *Hop Interval* à 75, et itéré sur les différentes tailles de *payload*. Les résultats de cette expérimentation sont présentés dans la figure 13.

De façon similaire à l'expérimentation 1, nous pouvons observer que la fiabilité de l'injection augmente quand la taille du *payload* diminue, ce qui est cohérent avec la théorie puisqu'une plus petite portion de la trame injectée entre en collision avec la trame légitime. Le nombre de tentatives d'injection avant d'obtenir une injection réussie reste très bas (valeur de la médiane inférieure à 3).

7.3 Expérimentation 3 : effet de la distance entre les objets

La dernière expérimentation nous a permis d'évaluer l'impact de la distance entre l'attaquant et le *Peripheral* légitime. Théoriquement, puisque la distance a un effet sur la puissance du signal de l'injection du point de vue du *Peripheral*, elle devrait réduire d'autant plus le succès de l'injection lors d'une collision avec la trame légitime. Nous avons utilisé l'ampoule connectée comme *Peripheral*, et un smartphone comme *Central* légitime pour être plus proche d'un scénario réel. Le smartphone a établi 25 connexions par distance testée, en utilisant 36 comme valeur du *Hop Interval*. Comme nous avons choisi d'injecter seulement des *Write Request* de 22 octets permettant d'éteindre l'ampoule, cette valeur de *Hop Interval* assure d'avoir des collisions lors des transmissions.

L'environnement expérimental était légèrement différent de celui utilisé lors des expérimentations 1 et 2 : nous avons placé l'ampoule et le smartphone à deux mètres de distance, et nous avons testé six positions différentes pour l'attaquant, entre 1 et 10 mètres, comme illustré dans la figure 14. Ceci a permis d'évaluer le succès de l'attaque quand l'attaquant est plus près du *Peripheral* que du *Central* légitime (position A), quand ils sont à la même distance (position B) et quand l'attaquant est plus éloigné (positions C,D,E and F).

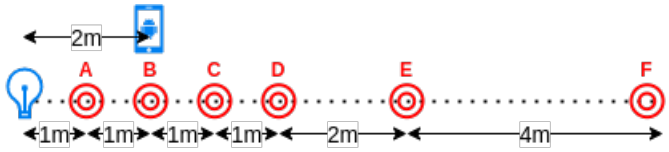

Fig. 14. Présentation de l'environnement expérimental

Les résultats sont présentés dans la figure 13. Ils montrent un impact significatif de la distance entre l'attaquant et le *Peripheral* sur la fiabilité de l'injection, puisque la variance augmente en fonction de la distance. Ceci valide notre hypothèse : l'attaquant a une plus grande probabilité de réussir rapidement une injection s'il est plus près de la cible. Cependant, notons que chaque connexion testée a mené à une injection réussie : cela signifie que l'attaquant peut réaliser une attaque réussie depuis chaque emplacement, y compris la position F qui est à 10 mètres du *Peripheral*, tandis que le *Master* légitime est seulement à 2 mètres de distance. Cette expérimentation montre que l'attaque est tout à fait utilisable, et ceci même dans des conditions défavorables, dans un environnement réaliste.

8 Contre-mesures

L'attaque *InjectaBLE* exploite une vulnérabilité inhérente aux spécifications du protocole BLE. Ainsi nous pouvons considérer toute communication BLE comme potentiellement vulnérable et les environnements incluant des objets BLE doivent être conçus et surveillés sous l'hypothèse que cette attaque pourrait potentiellement être menée contre n'importe quelle communication légitime. Plusieurs contre-mesures peuvent être envisagées pour limiter l'impact de cette attaque, pour empêcher son occurrence ou pour la détecter.

Comme expliqué dans la Section 4.2, l'implémentation de l'attaque nécessite d'injecter des trames arbitraires à des moments spécifiques. Deux solutions peuvent être proposées. Chacune nécessite plus ou moins de changements importants dans la pile protocolaire ou dans l'utilisation des puces BLE. Notons que ces changements pourraient être coûteux, notamment dans un environnement industriel, du fait du grand nombre d'objets à reprogrammer et du coût des processus de certification.

La première solution concerne les paramètres temporels de communication de la pile elle-même. Par exemple, minimiser la valeur du *window widening* réduit mécaniquement la possibilité pour un attaquant d'injecter une trame au bon moment. Plus précisément, le taux de succès de l'in-

jection va décroître puisque le taux de collision avec la trame légitime va augmenter. Cependant, on peut noter qu'une telle approche nécessite de modifier la spécification du protocole, ce qui pourrait générer des effets de bord impactant la fiabilité et la stabilité des communications.

La deuxième solution est légèrement plus restrictive. Sans aller jusqu'à modifier le standard BLE, elle nécessite d'activer systématiquement les mécanismes de chiffrement définis dans la spécification. Si toutes les trames sont correctement chiffrées, un attaquant ne sera pas en mesure de forger une trame valide. Alors que cette solution pourrait sembler évidente, il n'en est rien en réalité. On peut en effet noter que la majorité des communications BLE sont faiblement ou pas du tout chiffrées aujourd'hui (voir [22] pour une analyse quantitative du pourcentage d'équipements BLE qui activent les mécanismes cryptographiques dans différents cas d'usage). Ainsi, dans la plupart des cas, cette contre-mesure nécessite que les utilisateurs reprogramment tous leurs objets, ce qui peut être très délicat, notamment dans un contexte industriel.

Durant nos expérimentations, nous avons constaté que certains constructeurs n'utilisent pas les mécanismes natifs de chiffrement mais préfèrent implémenter leurs propres mécanismes cryptographiques au-dessus de la couche applicative *GATT*. Cette solution nous semble peu pertinente, puisque dans ce cas, les trames de contrôle de la couche Liaison ne sont pas chiffrées et nous avons déjà montré dans nos scénarios qu'un attaquant pourrait atteindre des objectifs intéressants en injectant ce type de trames. Il pourrait par exemple réaliser une attaque du type Man-in-the-Middle et ne pas faire suivre le trafic légitime pour réaliser un déni de service.

9 Conclusion

Dans cet article, nous avons démontré l'existence d'une nouvelle attaque ciblant le protocole BLE, nommée *InjectaBLE*, permettant d'injecter du trafic malveillant dans une connexion établie. Cette attaque augmente significativement la surface d'attaque des communications BLE, étant donné qu'elle exploite une vulnérabilité de la spécification elle-même indépendamment des différentes implémentations de la pile protocolaire, et peut être réalisée assez facilement à l'aide de puces BLE standards. Nous avons analysé l'impact de différents facteurs sur la réussite de l'attaque et avons montré que l'exploitation de cette vulnérabilité peut permettre à un attaquant de réaliser des scénarios d'attaque critiques qui n'étaient pas considérés réalistes jusqu'à aujourd'hui, tels que l'usurpation du rôle *Slave* ou des attaques *Man-in-the-Middle* ciblant des connexions déjà établies.

Nous avons également mené une analyse de sensibilité qui nous a permis de démontrer le réalisme de cette attaque, l'injection étant réalisée avec succès dans différentes conditions expérimentales.

Activer les mécanismes natifs de chiffrement du BLE constitue une contre-mesure efficace contre *InjectaBLE*. Cependant, en pratique, la grande majorité des objets commerciaux n'utilise pas le chiffrement, les rendant ainsi vulnérables par conception à *InjectaBLE*. Les résultats présentés dans cet article montrent clairement la faisabilité d'attaques d'injection dans les communications BLE non chiffrées. Bien que les mécanismes exploités pour mener cette attaque ne soient pas dépendants des mécanismes de sécurité du protocole, ces derniers compliquent considérablement l'attaque en théorie. De futurs travaux pourraient explorer plus en détail l'efficacité de cette contre-mesure.

Les nouvelles capacités offensives de *InjectaBLE* ouvrent des opportunités à d'autres scénarios d'attaque critiques qui doivent être considérés avec attention. Par exemple, être capable d'usurper le rôle *Slave* pourrait potentiellement permettre à un attaquant de changer la structure du serveur *ATT* pour exposer un profil de clavier et ainsi injecter des frappes clavier au *Master* en abusant du protocole *HID over GATT*, transformant ainsi une ampoule connectée inoffensive en un objet malveillant. En parallèle, il est également important de concevoir des approches défensives efficaces, tels que des systèmes de surveillance passifs permettant de détecter *InjectaBLE* en temps réel par l'analyse temporelle des communications.

Références

1. Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. Key negotiation downgrade attacks on bluetooth and bluetooth low energy. *ACM Trans. Priv. Secur.*, 23(3), June 2020.
2. Daniele Antonioli, Nils Ole Tippenhauer, Kasper Rasmussen, and Mathias Payer. Blurtooth : Exploiting cross-transport key derivation in bluetooth classic and bluetooth low energy, 2020.
3. Armis. Blueborne Technical White Paper. `https://info.armis.com/rs/645-PDC-047/images/BlueBorne%20Technical%20White%20Paper_20171130.pdf`, 2017.
4. Armis. BleedingBit Technical White Paper. `https://info.armis.com/rs/645-PDC-047/images/Armis-BLEEDINGBIT-Technical-White-Paper-WP.pdf`, 2018.
5. Bluetooth SIG. *Bluetooth Core Specification*, 12 2019.
6. S. Bräuer, A. Zubow, S. Zehl, M. Roshandel, and S. Mashhadi-Sohi. On practical selective jamming of bluetooth low energy advertising. In *2016 IEEE Conference on Standards for Communications and Networking (CSCN)*, pages 1–6, 2016.
7. Damien Cauquil. BtleJuice, un framework d'interception pour le Bluetooth Low Energy. `https://www.slideshare.net/NetSecureDay/nsd16-btle-juice-un-`

framework-dinterception-pour-le-bluetooth-low-energy-damien-cauquil, 2016.

8. Damien Cauquil. Sniffing btle with the micro :bit. *PoC or GTFO*, 17 :13–20, 2017. https://mcfp.felk.cvut.cz/publicDatasets/pocorgtfo/contents/issue17.pdf.
9. Damien Cauquil. You'd better secure your BLE devices or we'll kick your butts ! https://media.defcon.org/DEFCON26/DEFCON26presentations/DamienCauquil-Updated/DEFCON-26-Damien-Cauquil-Extras/, 2018.
10. Damien Cauquil. Defeating Bluetooth Low Energy 5 PRNG for fun and jamming. https://media.defcon.org/DEFCON27/DEFCON27presentations/DEFCON-27-Damien-Cauquil-Defeating-Bluetooth-Low-Energy-5-PRNG-for-fun-and-jamming.PDF, 2019.
11. Romain Cayre. Dépôt github injectable. https://github.com/RCayre/injectable-firmware.
12. Romain Cayre. Dépôt github mirage. https://github.com/RCayre/mirage/.
13. Romain Cayre, Vincent Nicomette, Guillaume Auriol, Eric Alata, Mohamed Kaâniche, and Geraldine Marconato. Mirage : towards a metasploit-like framework for iot. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pages 261–270. IEEE, 2019.
14. Matheus E. Garbelini, Chundong Wang, Sudipta Chattopadhyay, Sun Sumei, and Ernest Kurniawan. Sweyntooth : Unleashing mayhem over bluetooth low energy. In *USENIX ATC 20*, pages 911–925. USENIX Association, July 2020. https://www.usenix.org/conference/atc20/presentation/garbelini.
15. Sławomir Jasek. Gattacking Bluetooth Smart Devices. https://github.com/securing/docs/raw/master/whitepaper.pdf, 2017.
16. Sultan Qasim Khan. Sniffle : A sniffer for Bluetooth 5 (LE). https://hardwear.io/netherlands-2019/presentation/sniffle-talk-hardwear-io-nl-2019.pdf, 2019.
17. Mike Ryan. Bluetooth : With low energy comes low security. In *7th USENIX WOOT*, Washington, D.C., August 2013. USENIX Association. https://www.usenix.org/conference/woot13/workshop-program/presentation/ryan.
18. Mike Ryan. How Smart is Bluetooth Smart ?, 2013. https://lacklustre.net/bluetooth/how_smart_is_bluetooth_smart-mikeryan-shmoocon_2013.pdf.
19. Aellison Santos, José Filho, Avilla Silva, Vivek Nigam, and Iguatemi Fonseca. Ble injection-free attack : a novel attack on bluetooth low energy devices. *Journal of Ambient Intelligence and Humanized Computing*, 09 2019.
20. M. von Tschirschnitz, L. Peuckert, F. Franzen, and J. Grossklags. Method confusion attack on bluetooth pairing. In *S&P'21*, pages 213–228. IEEE Computer Society, 2021. https://doi.ieeecomputersociety.org/10.1109/SP40001.2021.00013.
21. Jianliang Wu, Yuhong Nan, Vireshwar Kumar, Dave (Jing) Tian, Antonio Bianchi, Mathias Payer, and Dongyan Xu. BLESA : Spoofing attacks against reconnections in bluetooth low energy. In *14th USENIX WOOT*. USENIX Association, August 2020. https://www.usenix.org/conference/woot20/presentation/wu.
22. Chaoshun Zuo, Haohuang Wen, Zhiqiang Lin, and Yinqian Zhang. Automatic fingerprinting of vulnerable ble iot devices with static uuids from mobile apps. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1469–1483, 2019.