Analyse des propriétés de sécurité dans les implémentations du Bluetooth Low Energy

Tristan Claverie[2], Nicolas Docq[1] et José Lopes-Esteves[2]
tristan.claverie@ssi.gouv.fr
nicolas.docq@intradef.gouv.fr
jose.lopes-esteves@ssi.gouv.fr

[1] Ministère des armées
[2] ANSSI

Résumé. Le Bluetooth Low Energy (BLE) est issu d'une norme complexe et qui laisse une grande latitude aux implémentations. Les fonctions de sécurité du BLE sont disséminées dans plusieurs couches protocolaires. Le fonctionnement du BLE et de sa sécurité ont été étudiés, avant qu'un état de l'art ne permette de comprendre les différentes attaques possibles et leurs impacts. Puis la propagation des propriétés de sécurité est étudiée sous le prisme de la norme avant d'étudier son implémentation dans les piles BLE de Linux et d'Android. Cette étude permet de conclure que la norme comporte des incohérences et de détecter des non conformités sur les implémentations étudiées.

1 Introduction

Le Bluetooth Low Energy (BLE) a été initialement créé par Nokia sous le nom de Wibree en 2006 pour compléter le Bluetooth mais sans le remplacer [10]. Le consortium Bluetooth Special Interest Group (SIG) a décidé d'intégrer cette technologie à la norme Bluetooth. Cette évolution du Bluetooth a été commercialisée en 2010 dans sa version 4.0. C'est l'iPhone 4S qui a été en 2011 le premier équipement grand public à l'utiliser. Aujourd'hui, le BLE est intégré dans un nombre incalculable d'équipements tels que smartphone, casque audio ou serrure connectée par exemple. L'Internet des Objets exploite de manière importante ce protocole dont l'intérêt est d'être léger et peu consommateur d'énergie électrique. Du point de vue de l'utilisateur, le BLE se comporte exactement de la même manière que le Bluetooth bien que les protocoles sous-jacents soient extrêmement différents. Comme le Bluetooth, le BLE peut nécessiter une phase d'appairage entre les deux équipements qui veulent communiquer ensemble. Une fois cet appairage initial réalisé ; ayant en principe nécessité une action utilisateur ; il est possible que chacun des équipements stocke un secret pour qu'à la prochaine connexion, aucune action utilisateur ne

soit nécessaire. Lors de l'appairage, l'utilisateur averti pourra estimer le niveau de sécurité de sa liaison BLE en fonction de la méthode d'appairage utilisée, mais ensuite il n'y aura aucune indication sur le niveau de sécurité de la connexion BLE entre deux équipements.

Un article [1] publié en 2019 a démontré (en partie tout du moins) que la pile BLE BlueZ n'était pas totalement conforme à la norme sur le plan de la sécurité. Les auteurs ont pu accéder au niveau de sécurité le plus élevé de la pile BLE sans que toutes les conditions édictées par la norme ne soient vérifiées (et en particulier ici, la taille de la clé LTK dérivée de l'appairage). Cet article s'intéresse donc à la confiance qui peut être placée dans le BLE ainsi qu'aux garanties de sécurité apportées lors d'une connexion BLE. Les implémentations des piles BlueZ de Linux et Fluoride d'Android sont plus particulièrement étudiées.

En Section 2, une synthèse du fonctionnement du BLE et des mécanismes intervenant dans l'établissement de la sécurité d'une connexion BLE est proposée. La Section 3 présente un état de l'art de la sécurité des appairages. Puis une étude de la propagation des propriétés de sécurité selon la norme, dans le temps et au travers des couches de la pile protocolaire, est réalisée en Section 4. Une méthode d'analyse pratique des implémentations est décrite dans la Section 5. Enfin, les résultats des tests menés seront discutés en Section 6.

2 BLE : fonctionnement et sécurité

En BLE, la sécurité est gérée par plusieurs couches protocolaires distinctes, avec un vocabulaire changeant. Quelques généralités permettront de se familiariser avec le BLE, avant de s'intéresser à chacune des couches liées à la sécurité.

2.1 Généralités

Le BLE fonctionne sur la bande Industrielle Scientifique et Médicale des 2,4 GHz. Bien que le BLE soit défini dans le même document que le Bluetooth Classique, ces deux technologies ne sont pas compatibles.

Deux sous-ensembles majeurs composent la pile BLE. Le sous-ensemble contrôleur gère la connexion radio tandis que le sous ensemble hôte est chargé de ce qui va être transmis sur le média. Une interface hôte-contrôleur (HCI) gère les interactions entre les deux sous-ensembles. La figure 1 permet de prendre connaissance de l'ensemble des couches protocolaires du BLE et de ces sous-ensembles.

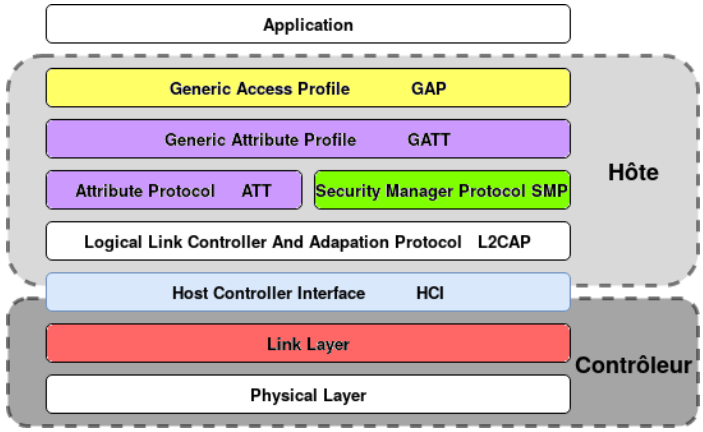

Fig. 1. La pile protocolaire du BLE

L'hôte et le contrôleur peuvent être sur une même puce électronique (cas fréquent des équipements BLE prévus pour être déployés comme objets connectés autonomes). Cette étude s'intéresse aux implémentations Linux et Android du BLE. Dans ce cadre, les systèmes d'exploitations implémentent le sous-ensemble hôte tandis que le sous-ensemble contrôleur est dévolu à la partie matérielle. Ainsi, seules les couches liées à la sécurité du sous-ensemble hôte seront étudiées, ce qui exclut la couche L2CAP qui est entièrement fonctionnelle.

La figure 2 permet de percevoir l'imbrication des couches liées à la sécurité sur un cas d'usage simple d'un utilisateur de smartphone qui veut accéder aux données d'un capteur de fréquence cardiaque. Ainsi, le cardiofréquencemètre va s'annoncer en diffusion. Le smartphone va alors vouloir accéder aux services offerts par le capteur, il y aura donc au niveau radio une connexion du smartphone vers le capteur. Le smartphone qui est alors client va émettre une requête au cardiofréquencemètre pour prendre connaissance des services délivrés et des caractéristiques exposées. Il va ensuite demander à accéder à une donnée stockée dans la base de données. Le cardiofréquencemètre va vérifier si le niveau de sécurité de la liaison est suffisant. Il peut y avoir une phase de mise en place du niveau de sécurité attendu. Finalement, le cardiofréquencemètre fournira la donnée si le niveau de sécurité de la liaison est correct. En rapprochant l'exemple précédent du modèle protocolaire du BLE, les annonces diffusées et la connexion radio incombent à la couche *Link Layer*, la demande d'accès à une donnée aux couches *Attribute Protocol* (ATT) et *Generic Attribute Profile* (GATT). Puis l'étape de vérification du niveau de sécurité est

faite par la couche *Generic Access Profile* (GAP). Selon le résultat, un appairage peut avoir lieu, qui incombe alors à la couche *Security Manager Protocol* (SMP), avant que la donnée ne soit finalement envoyée par les couches ATT et GATT si l'éventuel appairage s'est correctement terminé.

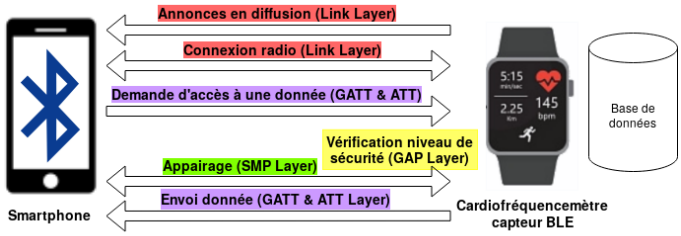

Fig. 2. Accès à la donnée d'un capteur BLE par un smartphone

2.2 La couche Link Layer

Dans cette couche, en reprenant l'exemple de la figure 2, le smartphone a le rôle de maître et le cardiofréquencemètre celui d'esclave. Le chiffrement / déchiffrement des données provenant / à destination des couches supérieures sur ordre de la partie hôte est mis en œuvre par cette couche.

2.3 Les couches ATTribute (ATT) et Generic ATTribute (GATT)

Ces deux couches sont indissociables et gèrent les données. En reprenant l'exemple de la figure 2, le smartphone a le rôle de *Client* et le cardiofréquencemètre celui de *Serveur*.

La couche ATT stocke des attributs et les permissions pour pouvoir y accéder, c'est-à-dire en fixant des restrictions dessus. Pour sa part, la couche GATT ordonnance les attributs ATT en une base de données décrivant toutes les caractéristiques d'un équipement, avec des droits d'accès associés, offrant un accès à des profils. Ces profils standardisés permettent d'accéder aux données d'un serveur GATT sans avoir besoin d'en parcourir l'ensemble pour prendre connaissance des données exposées. La figure 3 permet d'avoir un aperçu simplifié de ce que peuvent être les permissions d'accès à différentes caractéristiques.

Ces exigences - aussi appelées permissions - sont fixées par les couches supérieures à la couche ATT et ne peuvent pas être découvrables en

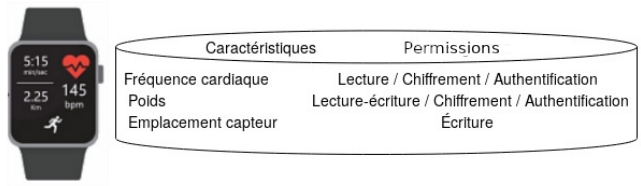

Fig. 3. Exemple sommaire de base de données GATT avec les exigences d'accès

utilisant le protocole ATT (page 1483 de la norme v5.2). La spécification définit un ensemble de base de permissions possibles qui sont décrites par le tableau 1.

Permission	d'accès	de chiffrement	d'authentification	d'autorisation
Possibilités	Lecture Écriture Lecture et Écriture	Chiffrement requis Pas de chiffrement requis	Authentification requise Pas d'authentification requise	Autorisation requise Pas d'autorisation requise

Tableau 1. Les différentes permissions ATT

De nombreuses combinaisons entre toutes ces permissions sont possibles. Les permissions d'un attribut sont donc une combinaison de permissions d'accès, de chiffrement, d'authentification et d'autorisation. La tentative d'accès à un attribut protégé alors que la liaison n'est pas dans le mode approprié déclenche un message d'erreur.

2.4 La couche Security Manager Protocol (SMP)

Cette couche gère l'appairage. Cet appairage intervient lorsque le niveau de sécurité requis est insuffisant. Cela permet à deux équipements d'échanger une clé après une authentification éventuelle. Cette clé permet d'élever le niveau de sécurité d'une connexion en la chiffrant. L'appairage commence par un échange de fonctionnalités, qui va permettre de choisir l'une des 7 méthodes d'appairage suivant les cas. Ces fonctionnalités contiennent notamment les capacité d'entrée/sortie de chaque équipement, les algorithmes supportés et les souhaits en terme de sécurité.

L'appairage peut être effectué en utilisant des méthodes d'appairages :

— *Legacy* : *Just Works*, *Passkey Entry* et *Out Of Band*.[3] Les méthodes *Legacy* sont historiques et apparues avec le Bluetooth 4.0 ;
— *Secure Connections* : *Just Works*, *Passkey Entry*, *Out Of Band* et *Numeric Comparison*. Ces méthodes sont apparues en 2014 avec le Bluetooth 4.2 pour résoudre certaines vulnérabilités des méthodes *Legacy*.

Il est à noter que le nom de la méthode d'appairage définit l'interaction utilisateur requise : par exemple les méthodes *Passkey Entry (Legacy)* et *Passkey Entry (Secure Connections)* décrivent deux protocoles distincts, donc des propriétés de sécurité distinctes.

A la suite d'un appairage peut avoir lieu une association (*bonding* dans la littérature anglaise), qui permet de ne pas avoir à refaire à chaque fois le processus d'appairage en stockant le secret partagé entre les deux équipements.

2.5 La couche Generic Access Profile (GAP)

Dans cette couche, le smartphone a le rôle de *Central* et le cardiofréquencemètre celui de *Périphérique*. La couche GAP permet de choisir un mode et niveau de sécurité qui a pour ambition de cacher toute la complexité des couches inférieures en permettant de fixer une exigence de sécurité pour avoir une vision sur la connexion en cours sans détailler élément par élément ce qui a été configuré. Le tableau 2 permet d'avoir le détail des niveaux de sécurité du mode 1 qui traite du chiffrement et de l'authentification des échanges BLE. Le mode 2 traite de la signature de certains types de messages particuliers et le mode 3 permet d'introduire de la sécurité pour de nouveaux modes de diffusion. Cet article se concentre sur la sécurité des communications, donc sur le mode 1.

Mode	Niveau	Connexion
1 - Sécurité des communications	1	Sans sécurité
	2	Chiffrement, pas d'authentification
	3	Chiffrement, authentification
	4	Chiffrement, authentification, impose *Secure Connections* et des clés de 128 bits

Tableau 2. Les 4 niveaux de sécurité du mode 1

Pour accéder à certains niveaux, il faut que l'appairage ait été effectué selon certains prérequis. Ce niveau de sécurité intervient pour vérifier

3. Out Of Band (OOB) : un autre canal de communication est utilisé pour transmettre le secret commun. Cela pourrait être un envoi du secret via NFC par exemple.

de manière globale si le niveau de sécurité exigé par une caractéristique correspond au niveau effectif de la liaison.

3 État de l'art sur la sécurité des appairages

Comme synthétisé en section 2, la spécification du BLE a prévu des moyens de sécuriser un échange. Ainsi, un appairage entre équipements est nécessaire pour atteindre un certain niveau de sécurité. Ce mécanisme est central pour la sécurité du BLE puisque la sécurité des communications dépendra de la méthode utilisée et du bon déroulé de l'appairage. Depuis la commercialisation du BLE, plusieurs vulnérabilités ont été trouvées dans différentes méthodes d'appairage. Cette section présentera le fonctionnement d'un appairage puis un état de l'art sur la sécurité des appairages.

3.1 Le fonctionnement de l'appairage

Dès lors que le niveau de sécurité d'une connexion est inférieur à ce qui est exigé pour l'accès à une caractéristique, les périphériques BLE vont tenter d'attendre le niveau requis. Pour cela, trois cas se présentent :

1. Les deux équipements possèdent déjà une clé de chiffrement commune et vont donc la réutiliser (cela signifie qu'il y a déjà eu un appairage puis une association).
2. Les deux équipements ne possèdent pas de clé de chiffrement commune et souhaitent s'associer, il sera réalisé un appairage puis une association.
3. Les deux équipements ne possèdent pas de clé de chiffrement commune et ne souhaitent pas s'associer, il sera réalisé uniquement un appairage.

Dans les deux derniers cas, une procédure d'appairage va avoir lieu afin que les deux équipements s'échangent une clé. Cette procédure débute invariablement par un échange de messages de type *PairingRequest* et *PairingResponse* qui va définir quelle méthode utiliser.

Les différences entre un appairage en *Legacy Pairing* et *Secure Connections Pairing* : malgré des noms de méthodes similaires, les appairages en *Legacy Pairing* et en *Secure Connections Pairing* sont fondamentalement différents et vont être rapidement détaillés.

L'appairage en Legacy Pairing :

- consiste à échanger entre les deux périphériques une clé temporaire nommée *Temporary Key* (TK), puis à authentifier (optionnellement) les équipements et enfin à créer une clé de chiffrement valide uniquement le temps de cette connexion nommée *Short Term Key* (STK).
- dans ce mode, l'association est la génération et l'échange d'une clé de chiffrement de long terme *Long Term Key* (LTK) après que l'appairage a eu lieu, et le stockage de cette clé pour une utilisation ultérieure.

L'appairage en Secure Connections Pairing :

- consiste à échanger une clé par le protocole *Elliptic-Curve Diffie-Hellman* (ECDH). Puis les équipements authentifient (optionnellement) cette clé et en dérivent la LTK ;
- dans ce mode, l'association consiste simplement à effectuer le stockage de la clé LTK pour une utilisation ultérieure.

Les différentes phases de l'appairage : l'appairage se fait en trois phases principales :

- phase 1 : échange des capacités des deux périphériques et de leurs souhaits / capacités de sécurité ;
- phase 2 : création de clés de chiffrement et authentification (optionnelle) via la méthode d'appairage déterminée en phase 1 ;
- phase 3 : distribution de clés.

Le lecteur curieux d'en savoir plus sur les protocoles d'appairage pourra se rapporter à [7].

3.2 Le lien entre les couches GAP et SMP

L'étude de la norme Bluetooth dans sa version 5.2 [9] permet de faire un lien - qui n'est pas explicite - entre les niveaux de sécurité de la couche GAP et les méthodes d'appairage de la couche SMP. Ce lien est décrit dans la figure 4. En effet, la norme décrit clairement les exigences de sécurité de chaque niveau et le fait que chaque méthode d'appairage permette de valider certaines des exigences.

3.3 État de l'art

La partie contrôleur offre potentiellement une très grande diversité de matériels. Bien que des failles aient été découvertes, il est plus complexe

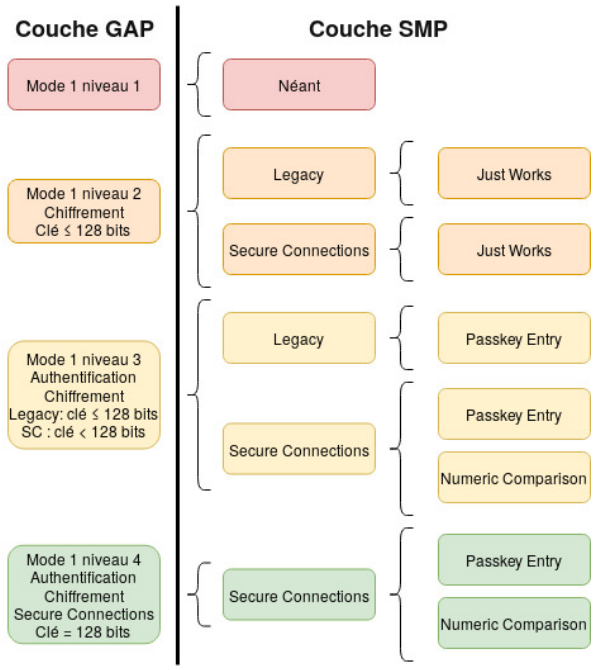

Fig. 4. Le lien entre les couches GAP et SMP

d'y mener des recherches et de les exploiter. Tandis qu'au niveau de la partie hôte, il y a un certain niveau d'abstraction qui fait que les vulnérabilités pourront être communes à de nombreuses plateformes, ou systèmes d'exploitation. De nombreuses recherches ont été faites sur cette partie, ce qui peut s'expliquer par la plus grande facilité d'instrumentation. C'est pourquoi le nombre de vulnérabilités trouvées au niveau de la partie hôte est plus important. Cette section va s'intéresser plus particulièrement à six attaques permettant de mettre en évidence les faiblesses de la norme sur certains sujets.

La récupération des secrets de *Just Works* et *Passkey Entry* en *Legacy* par force brute : le chercheur Mike Ryan a été le premier à publier [8] en 2013 une méthodologie et un outil permettant de casser les clés de chiffrement des méthodes d'appairage *Legacy Just Works* et *Legacy Passkey Entry* par force brute de manière passive. Il explique aussi que si l'échange de clé était réalisé via *OOB* avec une clé de 128 bits, il ne serait pas possible de la casser en un temps raisonnable, mais qu'il n'a jamais constaté d'implémentation réelle d'échange via cette méthode.

Cette attaque a des conséquences sur la confidentialité et l'intégrité des échanges.

La récupération des secrets de *Just Works* en *Secure Connections* : via une attaque de l'homme du milieu (MITM) [4], un attaquant actif est en mesure d'intercepter les échanges de messages d'appairage *PairingRequest* et *PairingResponse* et de les modifier pour que chaque équipement indique n'avoir aucune capacité d'entrée / sortie. De ce fait, les deux équipements légitimes vont s'appairer avec la méthode *Just Works* qui n'est pas authentifiée. L'attaquant étant toujours positionné en MITM, il peut faire une attaque active sur le protocole d'échange de clé. Une fois le processus d'appairage terminé, l'attaquant partage une clé de chiffrement avec chaque équipement et peut déchiffrer les échanges.

Attaque sur la méthode d'appairage *Passkey Entry* en *Secure Connections* : en 2008, lors du salon Black Hat, un chercheur a démontré [5] sur le Bluetooth Classique que la méthode d'appairage *Passkey Entry* en *Secure Simple Pairing* est vulnérable si le code numérique à 6 chiffres n'est pas généré aléatoirement et est donc prédictible. Or, en BLE, le mode d'appairage *Secure Connections* est similaire au mode *Secure Simple Pairing* du Bluetooth Classique. Cela signifie donc que la démonstration faite par A. Lindell de l'attaque possible sur la méthode d'appairage *Secure Simple Pairing Passkey Entry* est transposable au mode *Secure Connections* du BLE. Donc en BLE, si le code numérique à 6 chiffres est prédictible, la méthode d'appairage *Secure Connections Passkey Entry* est vulnérable à une attaque active et ne permet plus d'assurer la confidentialité et l'authenticité des échanges.

Vulnérabilité liée à une implémentation faible du protocole ECDH : pour se mettre d'accord sur la clé secrète LTK qui servira ensuite à créer des clés de session, lors d'un appairage via *Secure Connections*, les deux équipements utilisent le protocole d'échange de clé ECDH qui générera une clé *DHKey* dont la LTK sera dérivée. Des chercheurs ont montré qu'il est possible de modifier une partie de la clé publique sans modifier le résultat de l'appairage [2]. Dans certaines implémentations, il n'était pas vérifié que la clé publique reçue soit sur la bonne courbe elliptique. Il faut que l'attaquant soit actif lors de la séquence d'appairage pour tenter l'attaque proposée, qui fonctionne dans 25 à 50% des cas. En cas d'exploitation réussie, l'attaquant positionné en MITM peut déchiffrer tous les échanges entre les interlocuteurs légitimes ainsi que faire de

l'injection de paquets. En outre, une méthodologie de test a été publiée lors du SSTIC 2020 [3] pour vérifier si un équipement cible est vulnérable à une implémentation faible du protocole ECDH.

Attaques sur la gestion d'erreurs en reprise de connexion en BLE : dans [11], des scénarios où un attaquant exploite une reprise de connexion entre deux équipements qui ont déjà été appairés et associés par le passé sont mis en œuvre. Un attaquant actif au moment de la reprise de connexion parvient, quel que soit l'appairage qui a eu lieu précédemment, en envoyant des messages d'erreur indiquant qu'il ne connaît pas les secrets cryptographiques, à établir une session BLE en clair avec le pair. Différentes approches sont proposées, selon que le maître ou l'esclave est ciblé par l'attaquant. En complément, une perte d'information entre les couches basses et la couche applicative est identifiée comme étant un problème important pour la mise en place du mode *Secure Connections Only*.

KNOB attack ou la réduction de la taille de clé dans le mode 1 niveau 4 GAP : en 2019, des chercheurs ont documenté une attaque [1] basée sur la réduction de la taille de la clé de chiffrement. Ils ont nommé cette attaque : KNOB (Key Negotiation Of Bluetooth). Ils proposent deux variantes, une en Bluetooth Classique et une en BLE. Contrairement à ce qu'annonce l'étude, l'attaque décrite ne fonctionnera qu'avec l'utilisation d'un appairage en *Secure Connections* en BLE. En BLE, cette attaque permet de réduire la taille de la clé de chiffrement de 16 à 7 octets par la modification des paquets de la phase d'identification de l'appairage. Il est ensuite possible de casser par force brute la clé de chiffrement dans un temps raisonnable. Cette attaque sur la négociation de la taille de clé met en exergue trois vulnérabilités différentes, liées :

- à la norme Bluetooth qui prévoit que la taille de la clé puisse être négociée en BLE entre 7 et 16 octets. Or, cette réduction de la taille de clé n'apporte pas de gain de ressource pour un équipement ;
- au protocole : lors d'un appairage en *Secure Connection*, le protocole n'authentifie pas la taille de clé. Cela permet à l'attaquant de jouer une attaque de type MITM et d'usurper chacun des interlocuteurs à tour de rôle ;
- à l'implémentation Linux : sous Linux, en BLE, le mode de sécurité 1, niveau 4 doit permettre d'assurer que la taille de la clé est bien de 16 octets. Or, les chercheurs ayant réalisé cette attaque ont démontré qu'un Linux où le niveau de sécurité requis est le mode

1 niveau 4 acceptait des connexions avec une clé dont l'entropie avait été diminuée à 7 octets. La taille de la LTK n'est donc pas vérifiée correctement dans la pile BLE Linux, en particulier dans un mode de sécurité qui suppose d'en forcer la taille à 16 octets.

3.4 Bilan après l'état de l'art

Les différentes attaques décrites précédemment permettent de déduire les impacts en termes de confidentialité, d'intégrité ou d'authenticité selon les cas. Il est aussi à distinguer si l'attaquant est passif ou actif, ces capacités n'étant alors pas les mêmes. Le tableau 3 synthétise pour chaque attaque les impacts et précise si l'attaquant doit être passif ou actif.

Attaque	Impact sur	Attaquant	Réf.
1 ▶	Confidentialité et intégrité	Passif	[8]
2 ▶	Confidentialité et intégrité	Actif	[4]
3 ▶	Confidentialité, intégrité et authenticité	Actif	[5]
4 ▶	Confidentialité et intégrité	Actif	[2]
5 ▶	Confidentialité et intégrité	Actif	[11]
6 ▶	Confidentialité et intégrité	Actif	[1]

Tableau 3. Les différentes attaques et leurs impacts sur la sécurité des communications

Il apparaît donc que l'appairage est en BLE un moment critique de la mise en place de la sécurité. Par ailleurs, la connaissance de l'utilisation d'une méthode d'appairage sûre n'est pas suffisante non plus puisqu'il est possible à la reprise de connexion d'en modifier la sécurité en injectant des erreurs pour utiliser une méthode non sûre, comme l'a montré la cinquième attaque décrite.

4 La propagation des propriétés de sécurité

Avant de pouvoir s'intéresser à la propagation des propriétés de sécurité dans les implémentations cibles, il est nécessaire de comprendre ce qu'édicte la norme sur ce sujet. Il convient donc d'étudier la manière dont la norme préconise de gérer cette propagation.

4.1 L'absence de définition des interactions entre les couches ATT/GATT, GAP et SMP dans la norme

La norme Bluetooth (version 5.2) détaille le comportement de chaque couche, dont les couches GAP, SMP et ATT/GATT qui concentrent les mécanismes de sécurité d'intérêt pour cette étude. Les dialogues d'un équipement à un autre, en restant au niveau d'une couche précise, sont bien détaillés, y compris concernant les messages ou codes d'erreur qui doivent être envoyés. C'est ce que synthétise la figure 5. Par contre, la norme ne donne aucun détail concernant le dialogue entre les couches. Ce dialogue est pourtant nécessaire et obligatoire, puisqu'au moment de la première étape de la figure 5, il faut que le serveur GATT vérifie le niveau de sécurité effectif de la liaison, d'où les pointillés à tous les endroits où logiquement se trouve un dialogue inter-couches. Mais la norme est encore plus subtile, ce qui apparaît dans le tableau 2. En effet, pour qu'une liaison BLE puisse être dans le niveau de sécurité 4 de la couche GAP, il faut qu'une méthode d'appairage *Secure Connections* et une clé de chiffrement de 128 bits aient été utilisées (outre l'authentification et l'activation du chiffrement). Or, rien dans la norme n'indique comment gérer ces exigences au niveau ATT et GATT. Chaque implémentation BLE de la norme va donc potentiellement gérer ce cas d'une manière différente. Il est donc important de pouvoir étudier la manière dont les différentes implémentations ont répondu à cette problématique.

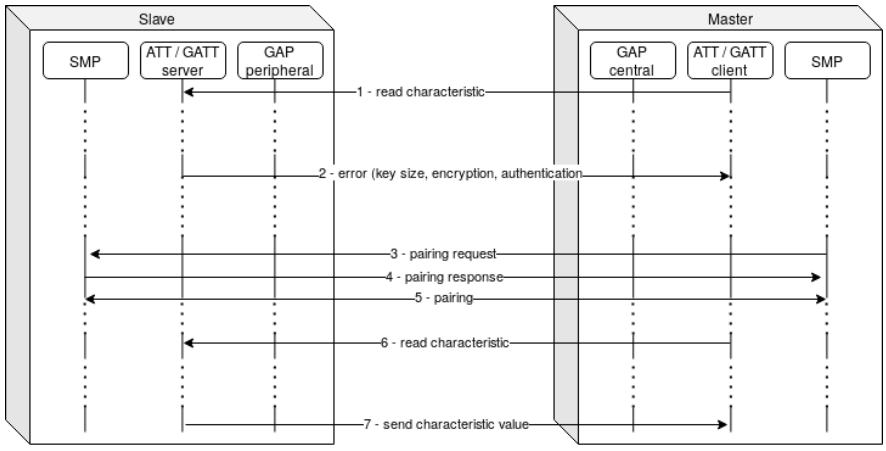

Fig. 5. Les interactions entre les couches ATT/GATT, GAP et SMP

4.2 L'absence de possibilité de décrire au niveau ATT certaines exigences de la couche GAP

Après qu'au niveau GAP, un périphérique a fait des annonces et qu'un central GAP veut accéder à une ou des caractéristiques proposées, tout le mécanisme de propagation des propriétés de sécurité part de la tentative d'accès du client GATT vers une caractéristique mise à disposition par le serveur GATT. Si la liaison BLE est dans un niveau de sécurité adéquat, l'accès est autorisé et donc la réponse est transmise. Comme expliqué en section 2.5 par le tableau 2, la norme définit les modes et niveaux de sécurité opérés par la couche GAP. De même, la section 2.3 tire de la norme le fait qu'au niveau ATT, il est possible de configurer des exigences (lecture et/ou écriture, chiffrement, authentification) sur les caractéristiques. La norme précise aussi que la taille de clé doit être respectée, si une taille minimale est exigée.

Or, au niveau ATT, il n'est pas indiqué comment prendre en compte la taille de clé dans la configuration des exigences de sécurité d'une caractéristique (ni même comment imposer une liaison *Secure Connections*). Bien que la norme laisse la porte ouverte à ces implémentations (page 1483 de la norme v5.2), rien n'est défini. Alors qu'il y a un lien naturel et clair entre les couches GAP et ATT sur la propagation des propriétés de sécurité sur le chiffrement et l'authentification qui sont clairement définies dans les deux couches, la propriété de taille de clé n'est définie que dans l'une de ces couches. Bien que la norme ne définisse clairement aucun champ, la configuration est cependant bien faite au niveau GAP puisqu'on peut avoir une clé de 128 bits ou moins (ce qui définit si l'on se trouve en niveau 4 ou niveau 3 notamment), alors que la norme explique que la taille de clé compte au niveau ATT mais sans définir comment.

De même, il est possible au niveau GAP de forcer l'utilisation lors de l'appairage de méthodes *Secure Connections* (page 1375 de la norme v5.2), mais il n'est pas prévu au niveau ATT/GATT d'exiger cette configuration (bien qu'une fois encore la norme laisse la porte ouverte à toute implémentation spécifique [4]).

Par ailleurs, une connexion peut avoir été établie dans un niveau de sécurité élevé, mais une caractéristique peut n'exiger aucune sécurité, il n'y

4. La fin de page 1843 de la norme v5.2 spécifie ainsi : *«Encryption, authentication, and authorization permissions can have different possibilities ; for example, a specific attribute could require a particular kind of authentication or a certain minimum encryption key length.»*. Toute la subtilité est dans le *«for example, a specific attribute could require. . . »*.

a pas de lien d'obligation entre le niveau de sécurité établi par la couche GAP et par l'exigence sur une caractéristique au niveau ATT/GATT.

De fait, au niveau norme, plutôt que d'évoquer la propagation de propriétés de sécurité, il pourrait être plus juste de parler de concordance entre une exigence au niveau de la couche ATT/GATT et un état actuel garanti par la couche GAP.

Finalement, cette étude fait apparaître deux manques distincts au niveau des couches ATT / GATT qui sont l'impossibilité de définir :

— la taille de clé ;
— l'exigence d'utilisation de méthodes d'appairage *Secure Connections*.

4.3 L'impossible connaissance du niveau de sécurité en reconnaissant une méthode d'appairage

En reprenant la figure 4 et en y ajoutant les attaques décrites en section 3.3 et leurs conséquences (tableau 3), il est obtenu la figure 6. Il apparaît clairement que les méthodes d'appairage *Just Works* et *Passkey Entry* en *Legacy* sont attaquables par un attaquant passif. En *Secure Connections*, la sécurité est plus élevée puisque aucune méthode d'appairage n'est vulnérable à une attaque passive.

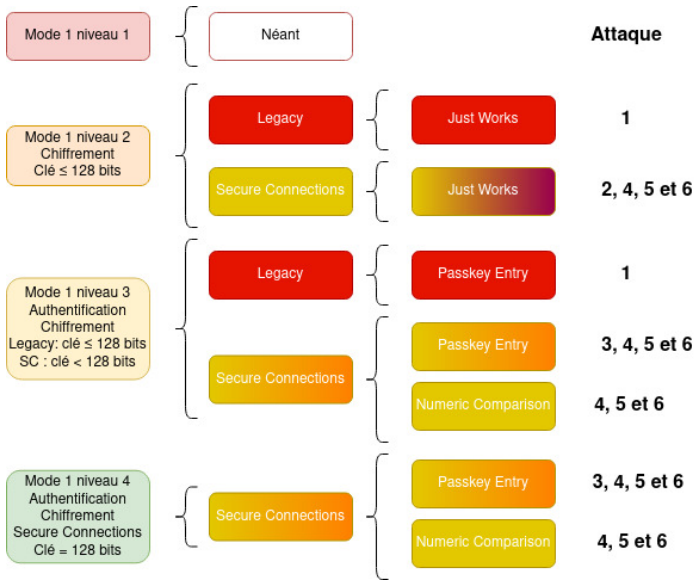

Fig. 6. Le lien entre les couches GAP et SMP après état de l'art

La figure 6 (ou la figure 4) permet déjà de visualiser un souci majeur. Si un utilisateur averti reconnaît lors d'un appairage l'utilisation de la méthode *Passkey Entry* par exemple, il lui sera impossible de savoir si au niveau de la couche GAP, il se trouve dans le mode 1 niveau 4, ou s'il se trouve dans le mode 1 niveau 3. Pire encore, dans le cas où il serait en mode 1 niveau 3, il ne lui est pas possible de savoir si le mode d'appairage utilisé est *Legacy* ou *Secure Connections*. Et la nuance est de taille, comme la section 3.3 l'a montré. L'impact en terme de sécurité est très différent suivant si l'appairage a utilisé une méthode *Legacy* ou *Secure Connections*, laissant l'utilisateur dans le flou concernant la confiance à accorder à son appairage. La figure 6 synthétise le lien entre couche GAP et SMP, après prise en compte de l'état de l'art.

4.4 Bilan après l'étude de la norme

Bien que complexe, la norme décrit relativement bien la manière dont doit être gérée la sécurité et la définition des propriétés de sécurité. Cependant, il est clair que pour un même niveau de sécurité affiché, la sécurité réelle peut être très différente, selon qu'un appairage ait été réalisé par une méthode *Legacy* ou *Secure Connections*. D'autant que la séquence d'appairage est un moment critique dans la sécurité d'une connexion BLE et qu'il peut être compliqué (cela dépend de l'attaque et des moyens d'observation) de savoir si un attaquant a pu mener à bien une attaque lors de cette phase. Comme la figure 6 le montre, la plupart des méthodes d'appairage sont sensibles à des attaques. Même le mode d'appairage *Secure Connections* est potentiellement vulnérable, bien que la plupart des implémentations récentes aient pu corriger le problème de la faiblesse d'implémentation du protocole ECDH.

Le mode 1 niveau 4 GAP est le plus sûr à tous égards puisqu'il n'y a que des méthodes d'appairage *Secure Connections* authentifiées. *A contrario* les niveaux de sécurité GAP 2 et 3 mélangent des méthodes *Secure Connections* et *Legacy* dont la robustesse très différente ne permet pas de garantir un niveau de sécurité réel de bon niveau.

Ces faits - issus de l'étude de la norme et des attaques ou vulnérabilités connues sur le BLE - posés, le cœur de l'étude ne concernera pas la sécurité des méthodes d'appairage mais l'adéquation entre les paramètres de sécurité exigés par la couche GATT, le niveau de sécurité configuré par la couche GAP et l'utilisation de la méthode d'appairage appropriée lorsqu'un appairage est déclenché. Par ailleurs, il sera étudié la manière dont les implémentations vont traiter les problématiques soulevées par la section 4.1, puisque la norme ne permet absolument pas de savoir comment

faire pour être dans le niveau GAP 4 avec les propriétés de sécurité définies par la couche GATT.

5 La méthode d'étude des implémentations

L'analyse de la norme permet de constater que le fonctionnement interne de chaque couche est relativement clair, alors qu'*a contrario* le dialogue inter-couches n'est pas décrit et que dans la norme, les couches GATT et ATT ne disposent pas de mécanismes permettant de stocker les exigences du niveau de sécurité 4 du mode 1 introduites par la couche GAP. Pour étudier les implémentations cibles (dont BlueZ de Linux sera la cible principale et Android la cible secondaire), la démarche d'analyse s'est faite sur un cycle itératif basé sur :

1. L'analyse statique du code source de BlueZ.
2. L'analyse dynamique de comportement de la pile BlueZ.
3. Des tests sur une plateforme comportementale.

Le fonctionnement de la plateforme comportementale sera détaillé, avant de s'intéresser pour BlueZ aux apports de l'analyse statique de code puis de l'analyse dynamique de la pile.

5.1 La plateforme comportementale

Afin d'étudier les interactions entre les différentes couches protocolaires, en particulier sur les points identifiés dans la section 4.4, il apparaît nécessaire de disposer d'outils permettant de générer de multiples cas d'études, en faisant varier les exigences de sécurité au niveau GATT et les capacités de sécurité au niveau SMP. Ce sont les objectifs fixés pour la plateforme comportementale, qui a été construite avec un système Linux comme serveur GATT et un système Android comme client GATT.

Le serveur GATT Linux : il est basé sur un système d'exploitation Linux Raspbian d'un ordinateur Raspberry Pi Zéro W qui dispose de capacités BLE. La pile BlueZ est en version 5.50 [5] et n'a initialement pas été modifiée. Le serveur GATT proprement dit est constitué d'un script Python nommé *uart-peripheral.py* (le code source permettant de créer le serveur GATT provient d'Internet [6]). Il permet de configurer les exigences de sécurité de caractéristiques exposées par le serveur GATT, qui

5. La commande `bluetoothctl -v` permet de connaître la version de BlueZ

seront ensuite accédées en lecture (il s'agit de la caractéristique nommée *UART_TX_CHARACTERISTIC_UUID*) et en écriture (il s'agit de la caractéristique nommée *UART_RX_CHARACTERISTIC_UUID*) depuis un client GATT via une interface UART.

Le client GATT Android : il est basé sur un système d'exploitation Android, sur un téléphone Asus Zenfone X00HD en version Android 7.1.1, rooté.[6] Les tests qui seront présentés dans la suite de cet article ont tous été faits avec le smartphone Asus Zenfone (mais l'utilisation d'un smartphone Nokia 7.2 en version 10 non rooté et d'un Samsung Galaxy S5 mini en version Android 9 Lineage OS rooté à permis de valider et confirmer la pertinence et la validité des tests). Pour le client GATT, deux applications Android réalisées par *Nordic Semiconductor* sont utilisées.

1. Pour l'écriture : l'application *nRF Toolbox for BLE* (gratuite dont le code source est téléchargeable sur GitHub, modifiable et compilable) servira pour l'écriture dans une caractéristique du serveur GATT (ces tests seront détaillés ultérieurement). Cette application permet d'établir une connexion vers le serveur GATT et de mettre en place un lien UART (mais aussi de se connecter à de nombreux capteurs BLE prédéfinis : fréquence cardiaque, humidité. . .) afin d'écrire dans certaines caractéristiques exposées par le serveur GATT.
2. Pour la lecture : c'est l'application *nRF Connect* (gratuite mais dont le code est propriétaire) qui sera utilisée. Cette application permet d'interroger un serveur GATT en temps que client, mais aussi de tenir un rôle de serveur GATT. L'écriture d'une caractéristique exposée par le serveur GATT est aussi possible mais moins simplement qu'avec *nRF Toolbox*).

La configuration de la plateforme pour chaque test : Linux et Android.

Sous Linux :

— au niveau de la couche Application, il faut configurer les exigences de sécurité des caractéristiques ATT dans le fichier *uart-peripheral.py*, comme le montre le listing 1. Le champ à modifier figure en jaune, les différentes valeurs possibles figurent dans la colonne *Couche Application* du tableau 4 ;

6. Un téléphone est dit *rooté* lorsque l'utilisateur a modifié son équipement Android pour avoir les droits administrateurs (donc root en terminologie Linux).

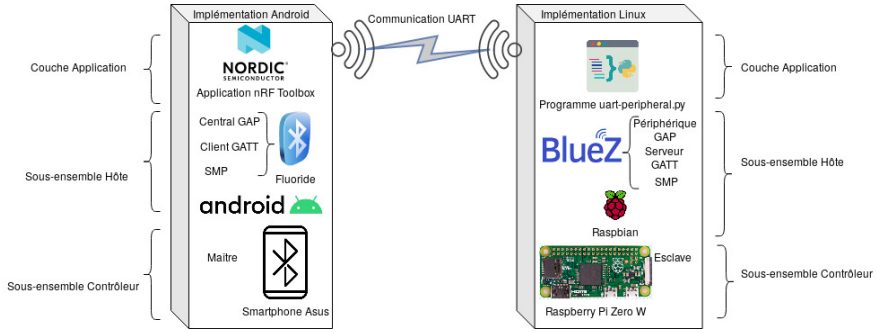

Fig. 7. Le schéma de principe de la plateforme comportementale

— au niveau de la couche SMP, il faut choisir la capacité [7] qui sera annoncée lors d'un appairage, avec l'utilitaire *btmgmt* comme le montre le listing 2.

```
class TxCharacteristic(Characteristic):
    def __init__(self, bus, index, service):
        Characteristic.__init__(self, bus, index,
            UART_TX_CHARACTERISTIC_UUID,
                                ['encrypt-read', 'notify'], service)

class RxCharacteristic(Characteristic):
    def __init__(self, bus, index, service):
        Characteristic.__init__(self, bus, index,
            UART_RX_CHARACTERISTIC_UUID,
                                ['encrypt-write'], service)
```

Listing 1. La configuration des propriétés de sécurité en lecture et écriture de la couche Application

```
user@debian:~$ sudo btmgmt
[mgmt]# io-cap KeyboardDisplay
IO Capabilities successfully set
```

Listing 2. La configuration de la capacité *KeyboardDisplay*

Sous Android : la capacité annoncée est fixe dans tous les cas et est *KeyboardDisplay*, ce qui correspond au maximum des capacités dont peut disposer un équipement.

Donc, en utilisant d'un côté les applications *nRF Toolbox* et *nRF Connect* sur un smartphone Android comme client GATT et de l'autre côté sur l'équipement Linux le script Python *uart-peripheral.py* comme

7. Il s'agit de la capacité d'entrée/sortie de l'équipement, qui est normalement liée au matériel (présence d'un écran, de boutons, d'un clavier...).

serveur GATT et en utilisant l'utilitaire *mtmgmt* ou *bt-agent* pour faire varier la capacité annoncée sous Linux, il est possible de créer une multitude de cas permettant de confronter ce que prévoit la norme avec les réactions des implémentations cibles. La figure 7 synthétise cette plateforme.

5.2 L'étude statique du code source de BlueZ

L'étude statique du code source de la pile BLE de Linux, BlueZ, a consisté à identifier l'ensemble des descripteurs de sécurité et à les corréler avec ceux mentionné par la norme. Puis, cet article s'intéresse à la manière dont un serveur GATT initialise les paramètres de sécurité. Cette rubrique permettra de répondre à la problématique exposée dans la section 4.2. Enfin, il sera décrit comment est gérée la sécurité lors d'une tentative d'accès à une caractéristique GATT avant de conclure.

Les descripteurs de sécurité de la pile BlueZ : l'étude du code source a permis de voir qu'un développeur d'application a accès dans la pile BLE, au niveau de la couche Application, à toutes les propriétés définies par la norme pour le niveau ATT, comme l'indique la colonne *Couche Application* du tableau 4. Il y a donc un lien fort avec la déclinaison explicite entre les propriétés de la couche Application et celles des couches ATT/GATT. Cependant, l'implémentation BlueZ nomme certaines variables avec *BT_GATT* en préfixe dans le nom, là où la norme indique que ce sont des propriétés *ATT* introduisant de l'ambiguïté.

Par ailleurs, au niveau Application et au niveau ATT (il s'agit des colonnes *Couches ATT & GATT* et *Couche Application* du tableau 4) se trouvent des descripteurs de sécurité qui n'existent pas dans la norme, avec dans chaque cas la dénomination *secure* dans le nom de la variable. Il semble que BlueZ introduise des descripteurs non prévus par la norme pour répondre à la problématique exposée dans la section 4.2.

Enfin, il y a deux jeux de descripteurs de sécurité dont les noms sont proches. L'analyse a montré qu'un de ces jeux permet d'avoir le niveau de sécurité global exigé lors de l'accès à une caractéristique (il s'agit de la colonne *Niveau ATT global* du tableau 4 avec `BT_ATT_SECURITY_HIGH` par exemple), tandis que le deuxième permet d'avoir le niveau de sécurité actuel d'une connexion BLE (il s'agit de la colonne *Socket Linux* du tableau 4 avec par exemple `BT_SECURITY_HIGH`). L'intérêt de ces deux jeux nominativement très proches sera explicité à la fin de cette section.

L'initialisation des paramètres de sécurité d'un serveur GATT : le cœur de l'initialisation des paramètres de sécurité d'un serveur GATT et

de la base de données ATT se trouve dans la fonction `parse_chrc_flags` du fichier `src/gatt-database.c`. Cette fonction se charge suivant les cas d'initialiser pour chaque caractéristique les propriétés GATT et/ou les propriétés étendues GATT et/ou les permissions ATT. Le lien entre les descripteurs de sécurité de la couche Application et ceux des couches ATT et GATT est clairement fait dans cette fonction. Par ailleurs, il y a bien une filiation nette entre les descripteurs `secure-write` et `secure-read` de la couche Application et les descripteurs `BT_ATT_PERM_WRITE_SECURE` et `BT_ATT_PERM_READ_SECURE` de la couche ATT.

Ces descripteurs sont introduits par BlueZ et n'existent pas dans la norme. Cet ajout par rapport à la norme est le biais décidé par les développeurs de la pile BlueZ pour répondre à la problématique introduite par le mode 1 niveau 4 de la couche GAP qui impose, outre le chiffrement et l'authentification, l'utilisation d'un appairage avec *Secure Connections* et d'une clé de chiffrement de 128 bits. La pile BlueZ a été écrite alors que cette exigence n'existait pas, et il était probablement plus simple d'introduire de nouvelles constantes pour gérer cette exigence de sécurité.

La sécurité lors d'une tentative d'accès à une caractéristique GATT : les objets importants dans ce modèle client / serveur GATT pour la vérification des permissions d'accès aux caractéristiques sont :

- un socket Linux permettant d'obtenir via un objet `bt_security` et la fonction `bt_att_get _security_sec` le niveau de sécurité en cours (*low, medium, high, fips* comme décrit dans `bluetooth.h`) ;
- un serveur GATT qui contient notamment la taille de clé (`server->enc_size`) ;
- un ou plusieurs attributs `ATT`, chacun possédant une ou plusieurs permissions récupérées via la fonction `gatt_db_attribute_get_permissions` ;
- une fonction `check_permissions` qui vérifie si l'accès demandé est une lecture ou une écriture, puis qui récupère le niveau de sécurité de la liaison et la taille de clé, puis qui vérifie l'adéquation entre les permissions de l'attribut et les caractéristiques de la liaison.

C'est la fonction **`check_permissions`** qui est au cœur de la vérification de l'adéquation du niveau de sécurité de la connexion en cours avec l'exigence de sécurité lors de la tentative d'accès à une caractéristique. Cette fonction fait le lien dans l'implémentation BlueZ entre :

- ce que la norme définit pour la couche GAP ;
- des variables décrites dans BlueZ au niveau GATT et au niveau du *socket* Linux. C'est la comparaison de ces deux variables (des

colonnes *Socket Linux* et *Niveau ATT global* du tableau 4) qui permet de voir si le niveau de sécurité est suffisant pour permettre de donner accès à la caractéristique ou non.

Par ailleurs, cette analyse statique du code source de *BlueZ* a permis de déterminer que l'adéquation entre les permissions exigées par chaque attribut et la sécurité de la connexion GATT en cours n'est pas correctement faite par la fonction `check_permissions` en raison de deux non conformités qui touchent le mode 1 niveau 4 GAP par rapport à la norme. Ces deux non conformités sont décrites dans la section 6.3.

Enfin, c'est bien aussi dans cette fonction critique qu'est fait le lien entre les descripteurs de sécurité qui n'existent pas dans la norme mais introduits par les développeurs de BlueZ. Ces descripteurs permettent de gérer le cas du mode 1 niveau 4 de la norme qui mélange des exigences décrites par la couche ATT (chiffrement et authentification) et des exigences décrites par la couche SMP (taille de clé, utilisation du mode *Secure Connections*).

Conclusion sur l'analyse statique du code de BlueZ : le tableau 4 montre bien la cohérence entre l'implémentation faite par BlueZ et la norme, bien que la place de la couche GAP puisse paraître curieuse lorsque le code de BlueZ est analysé. En effet, en mode 1, les niveaux de sécurité de 1 à 4 sont déclinés à deux endroits : dans la couche ATT, et au niveau du *socket* Linux qui est le descripteur de la connexion BLE actuelle. C'est le test entre ces deux différents types de descripteurs qui permet de savoir si l'accès est permis ou non. Après avoir analysé de manière statique l'implémentation par la pile BlueZ de la norme Bluetooth, il est possible de conclure que la pile BlueZ n'implémente pas de manière juste la norme BLE en raison des non conformités qui sont décrites dans la section 6.

5.3 L'analyse dynamique de la pile BlueZ

Dans l'analyse dynamique de la pile BlueZ, ce qui est recherché c'est de pouvoir observer le niveau de sécurité configuré dans les différentes couches, en temps réel, lors de l'utilisation du BLE. La méthode choisie est d'ajouter des `printf` dans certaines fonctions du code source de BlueZ identifiées lors de l'analyse statique de code puis de recompiler le code source en espace utilisateur et enfin relancer le démon Bluetooth en mode débogage en le conservant au premier plan. Cela permet ainsi d'afficher les variables intéressantes et de vérifier à quels moments sont utilisées les fonctions instrumentées. Cette manière de faire peut paraître triviale, mais il n'a pas paru possible de déterminer plus simplement le niveau de sécurité atteint par une connexion BLE.

Niveau de sécurité	Couche selon la norme Couche selon BlueZ	Couche GAP Socket Linux	 Niveau ATT	Couches ATT & GATT	Couche Application
Mode 1 niveau 1	Pas de sécurité (pas d'authentification ni de chiffrement)	BT_ SECURITY_ LOW	BT_ATT_ SECURITY_ LOW	BT_GATT_CHRC_PROP_READ BT_ATT_PERM_READ	read
				BT_GATT_CHRC_PROP_WRITE BT_ATT_PERM_WRITE	write
				BT_GATT_CHRC_PROP_WRITE_WITHOUT_RESP BT_ATT_PERM_WRITE	write-without-response
				BT_GATT_CHRC_EXT_PROP_RELIABLE_WRITE BT_ATT_PERM_WRITE	write-reliable
				BT_GATT_CHRC_EXT_PROP_BROADCAST	broadcast
				BT_GATT_CHRC_EXT_PROP_NOTIFY	notify
				BT_GATT_CHRC_EXT_PROP_INDICATE	indicate
				BT_GATT_CHRC_EXT_PROP_WRITABLE_AUX	writable-auxiliaries
				BT_GATT_CHRC_PROP_EXT_PROP	extended-properties
				req_prep_authorization = true	authorize
Mode 1 niveau 2	Appairage non authentifié	BT_ SECURITY_ MEDIUM	BT_ATT_ SECURITY_ MEDIUM	BT_GATT_CHRC_PROP_READ BT_GATT_CHRC_EXT_PROP_ENC_READ BT_ATT_PERM_READ BT_ATT_PERM_READ_ENCRYPT	encrypt-read
	Données chiffrées			BT_GATT_CHRC_PROP_WRITE BT_GATT_CHRC_EXT_PROP_ENC_WRITE BT_ATT_PERM_WRITE BT_ATT_PERM_WRITE_ENCRYPT	encrypt-write
Mode 1 niveau 3	Appairage authentifié	BT_ SECURITY_ HIGH	BT_ATT_ SECURITY_ HIGH	BT_GATT_CHRC_PROP_READ BT_GATT_CHRC_EXT_PROP_AUTH_READ BT_ATT_PERM_READ BT_ATT_PERM_READ_AUTHEN	encrypt-authenticated-read
	Données chiffrées			BT_GATT_CHRC_PROP_WRITE BT_GATT_CHRC_EXT_PROP_AUTH_WRITE BT_ATT_PERM_WRITE BT_ATT_PERM_WRITE_AUTHEN	encrypt-authenticated-write
Mode 1 niveau 4	Appairage authentifié LE Secure Connections	BT_ SECURITY_ FIPS	BT_ATT_ SECURITY_ FIPS	BT_GATT_CHRC_PROP_READ BT_GATT_CHRC_EXT_PROP_AUTH_READ BT_ATT_PERM_READ BT_ATT_PERM_READ_SECURE	secure-read
	Données chiffrées			BT_GATT_CHRC_PROP_WRITE BT_GATT_CHRC_EXT_PROP_AUTH_WRITE BT_ATT_PERM_WRITE BT_ATT_PERM_WRITE_SECURE	secure-write
Mode 2 niveau 1	Appairage non authentifié, signature des données	•	•	•	Non implémenté
Mode 2 niveau 2	Appairage authentifié, signature des données	•	•	BT_GATT_CHRC_PROP_AUTH BT_ATT_PERM_WRITE	authenticated-signed-writes

Tableau 4. Les descripteurs des niveaux de sécurité de la pile BlueZ suivant la couche d'abstraction

En procédant ainsi, l'analyse dynamique de la pile BlueZ a permis de :
— visualiser le niveau de sécurité de la liaison BLE représenté par un nombre entier pouvant aller de 1 à 4. Pour la valeur *1*, cet entier correspond au mode 1 niveau 1 (BT_SECURITY_LOW) jusqu'au mode 1 niveau 4 (BT_SECURITY_FIPS) qui vaut *4* ;
— savoir dans quel état est une liaison BLE, à savoir s'il y a appairage, association et si l'opération est en cours, réalisée ou réussie ;
— voir la tentative d'élévation du niveau de sécurité lorsqu'un accès à une caractéristique nécessite un niveau de sécurité supérieur ;
— valider l'importance des fonctions obtenues lors de l'analyse statique ainsi que leur utilisation par la pile BlueZ ;
— confirmer les conséquences de la non conformité 2 décrite à la section 6.3 ;
— permettre la récupération du niveau de sécurité atteint lors de jeux de tests avec la plateforme comportementale.

5.4 L'implémentation de la sécurité dans la pile BlueZ

La mise en concordance, dans le tableau 4, des niveaux de sécurité édictés par la norme et de ce qui est implémenté aux couches ATT et GAP par la pile BlueZ est logique mais déduit du code source. A aucun moment il n'a été possible d'avoir de certitude sur ce sujet, le code étant faiblement commenté. L'instrumentation dynamique a permis de confirmer la compréhension acquise par le biais de l'analyse statique.

Les descripteurs de sécurité introduits par BlueZ alors qu'ils n'existent pas dans la norme permettent de répondre à la problématique introduite par la norme avec le mode 1 niveau 4 de la couche GAP qui impose des conditions de taille de clé et de méthode d'appairage sans fournir de descripteur au niveau ATT. Mais les deux non conformités (décrites dans la section 6.3 dont la présence de la deuxième a été confirmée par l'analyse dynamique) montrent que tout n'a pas été parfaitement implémenté, et que c'est le mode 1 niveau 4 de la couche GAP, censément le plus sécurisé, qui cumule les non conformités. Bien qu'il faille étudier les résultats des tests menés à l'aide de la plateforme comportementale, les différentes méthodes d'analyses montrent déjà que l'implémentation BlueZ ne respecte pas parfaitement et totalement la norme Bluetooth version 5.2.

Il faut noter que sans l'instrumentation dynamique, il n'aurait pas été possible de trouver une méthode pour savoir quel est le niveau de sécurité réel d'une connexion BLE, que l'on soit utilisateur ou administrateur de l'équipement Linux. Le développeur d'application peut exiger un niveau

de sécurité, mais de toute manière, la couche Application n'a pas de moyen simple au premier abord de vérifier si ce niveau est bien configuré.

Il est dommage que de manière simple et intuitive, la pile BlueZ ne permette pas de savoir quel est le niveau de sécurité configuré entre deux équipements appairés, quel que soit le rôle tenu par celui qui recherche cette information (administrateur, utilisateur, développeur).

6 Les résultats de tests

Cette section expose tout d'abord la manière dont les résultats attendus ont été déterminés. Puis les résultats obtenus en ayant étudié les implémentations cibles seront décrits, avec une première partie sur les non conformités ayant un impact fonctionnel ou modéré sur la sécurité et une deuxième partie sur les non conformités ayant un impact fort sur la sécurité.

6.1 Les résultats prévus

Avant de pouvoir réaliser des tests à l'aide de la plateforme comportementale présentée dans la section 5.1, il convient d'identifier les résultats qui devraient se produire en croisant :

— au niveau de la couche SMP, les capacités annoncées par les deux équipements et en sélectionnant un appairage utilisant les méthodes *Secure Connections* ou *Legacy* ;
— au niveau de la couche GATT les exigences de sécurité positionnées sur une caractéristique en lecture et une caractéristique en écriture en configurant le serveur GATT via la couche Application.

En l'état actuel de la plateforme, concernant le client GATT qui s'exécute sur un smartphone Android, il n'est pas possible de faire varier les capacités annoncées. La couche SMP du smartphone annonce systématiquement lors d'un appairage les capacités *KeyboardDisplay*, qui sont les capacités maximales que peut avoir un équipement. Par ailleurs, il n'est pas non plus possible de forcer l'utilisation de méthodes d'appairage *Legacy* ou *Secure Connections*, le smartphone utilisé pour les tests annonce en permanence dans les *PairingRequest* qu'il dispose de méthodes *Secure Connections*.

C'est donc l'équipement utilisant Linux et la pile BlueZ qui fera varier l'ensemble des paramètres, permettant ainsi de réaliser l'ensemble des tests souhaités. En interagissant avec BlueZ, il sera possible de modifier :

— l'exigence de sécurité des caractéristiques du serveur GATT, telle que décrite dans le tableau 4 via la couche Application ;

— le choix global des méthodes d'appairage, c'est-à-dire soit en imposant le mode *Legacy*, soit en indiquant que le mode *Secure Connections* est possible.[8] Le choix sera fait en fonction de la négociation entre le maître et l'esclave ;
— les capacités[9] annoncées par la couche SMP de l'ordinateur Linux, entre *NoInputNoOutput*, *DisplayOnly*, *KeyboardOnly*, *DisplayYesNo* et *KeyboardDisplay*.

Pour identifier les résultats attendus, les éléments suivants ont été mis en concordance :

— le tableau 2 qui indique pour les 4 niveaux de sécurité du mode 1 de la couche GAP quels éléments doivent être respectés pour prétendre à ce niveau ;
— le tableau situé pages 1641 et 1642 de la norme v5.2 [9] qui décrit précisément quelle méthode d'appairage doit être utilisée suivant les capacités annoncées par chacun des équipements ;
— la figure 4 qui recense quel niveau de sécurité peut être atteint suivant la méthode d'appairage utilisée.

Il est important de noter que les deux équipements, s'ils doivent s'appairer, vont échanger leurs fonctionnalités (*Secure Connections*, MITM, OOB, capacités d'entrée/sortie. . .). C'est à partir de cet échange de fonctionnalités lié aux messages *PairingRequest* et *PairingResponse* qu'une méthode d'appairage sera décidée. Cela veut dire que si les deux équipements peuvent s'appairer avec la méthode la plus sécurisée, ils le feront, même si ensuite les caractéristiques GATT peuvent n'exiger qu'un mode 1 niveau 2 par exemple. Il faut donc être conscient que le niveau de sécurité retenu après l'appairage pourra être supérieur au niveau qui sera exigé par certaines caractéristiques protégées. Cela n'aurait pas de sens de négocier le niveau de sécurité strictement suffisant pour la caractéristique ayant certaines exigences, puisque ensuite, une autre caractéristique pourrait exiger un niveau de sécurité supérieur. La négociation aboutit donc en

8. Dans l'utilitaire *btmgmt* fourni avec BlueZ, il est possible d'intervenir sur la capacité à utiliser les méthodes *Secure Connections*. Indiquer "yes" signifie que les appairages *Legacy* et *Secure Connections* sont possibles. Choisir "no" signifie que seul les appairages *Legacy* seront disponibles. Le choix "only" signifie que seul les appairages *Secure Connections* seront disponibles.

9. Rappelons que la capacité d'un équipement dépend théoriquement de son matériel, à savoir s'il dispose de boutons ou d'un clavier, ou bien d'un écran par exemple. Il découle des différents cas possibles une matrice dont les entrées sont les capacités décrites juste après.

permanence au niveau de sécurité maximal atteignable compte tenu des fonctionnalités échangées.

6.2 Les non conformités avec un impact modéré sur la sécurité

L'utilisation de la plateforme comportementale a permis de relever les problématiques suivantes actuellement non résolues, qui ont un impact fonctionnel ou modéré sur la sécurité :

1. La configuration des agents Bluetooth de la pile BlueZ ne permet pas de configurer en BLE la capacité *KeyboardDisplay*, empêchant de réaliser les tests avec cette capacité. Cette impossibilité est inexplicable.
2. La configuration du *Secure Connections only mode* ne permet plus de faire fonctionner normalement la pile BlueZ. En effet, il est possible de forcer la pile BlueZ dans le mode *Secure Connections only* à l'aide de l'utilitaire *btmgmt*, la pile BlueZ respectant ainsi bien la norme sur ce sujet là. Cependant après avoir activé le *Secure Connections only mode*, les tentatives d'appairage ont curieusement systématiquement échouées. Le fait de forcer sous Linux le *Secure Connection only mode* entraîne une impossibilité de connexion en utilisant la plateforme comportementale. Il n'a pas été trouvé les raisons de ce dysfonctionnement illogique, dans le sens où les méthodes d'appairage *Secure Connections* fonctionnent parfaitement lorsque *btmgmt* est configuré en *sc yes* mais plus en *sc only*. Il a été procédé à l'analyse comparative des fichiers journaux HCI Linux et Android lorsqu'un appairage en *Secure Connections only mode* échoue et lorsqu'un appairage réussi en utilisant une méthode *Secure Connections* (mais sans l'imposer par la consigne *Secure Connections only mode*). Ce dysfonctionnement n'a pas d'impact sur les tests menés mais est gênant puisqu'il ne permet pas d'empêcher l'utilisation d'appairages *Legacy* le cas échéant.
3. Lorsque la méthode d'appairage *PasskeyEntry* est utilisée, avec BlueZ affichant le code numérique à 6 chiffres et la saisie s'effectuant sur Android, la saisie d'un code erroné sur Android ne donne pas lieu à une deuxième tentative (la norme prescrit 3 tentatives).
4. En console sous Linux, lorsque le code numérique à 6 chiffres commence par un zéro, celui-ci n'est pas affiché, occasionnant des erreurs de saisie pour l'utilisateur insuffisamment averti.

5. Sur Android, lors d'une séquence d'appairage, l'action nécessitant une action utilisateur est dans certains cas au premier plan, dans d'autres cas elle s'affiche simplement dans la barre de notifications, pouvant amener l'utilisateur à ne pas voir qu'une action de sa part est nécessaire. Il n'a pas été trouvé de logique dans les cas.

6.3 Les non conformités avec un impact fort sur la sécurité

Non conformité concernant le mode 1 niveau 4 de la couche GAP : L'attaque *KNOB* décrite dans la section 3.3 est toujours exploitable avec les piles BlueZ en version 5.50 jusqu'à 5.54. En effet, l'attaque décrite par l'article [1] consiste à négocier la taille de clé minimale lors de la phase d'appairage, pour ensuite pouvoir casser la clé par force brute, alors que la connexion a été établie en mode 1 niveau 4 de la couche GAP. C'est une non conformité par rapport à la norme puisque pour être en mode 1 niveau 4, il faut avoir utilisé un appairage *Secure Connections*, activé le chiffrement avec une clé de 128 bits et réalisé de l'authentification. Il convient de comparer au bon endroit la taille de clé de la connexion actuelle avec 128 bits, et pas avec la taille de la clé stockée dans le serveur GATT, qui peut être inférieure à 128 bits.

Non conformité concernant le mode 1 niveau 4 de la couche GAP : Une non-conformité de la pile BlueZ par rapport à la norme a été relevée, liée aux vérifications de l'adéquation entre le niveau de sécurité demandé et celui configuré. Dans la fonction `check_permissions`, la permission *secure* n'est pas vérifiée correctement. Autrement dit, lorsque le mode 1 niveau 4 est exigé pour une caractéristique, la connexion BLE reste dans son état antérieur sans tenir compte de cette exigence. Une connexion BLE débutant toujours en mode 1 niveau 1 (sans confidentialité ni authentification) restera donc non protégée même si le développeur du service GATT sur BlueZ en configure les caractéristiques pour le plus haut niveau de sécurité possible. Dans ce cas, un attaquant passif peut accéder aux informations échangées lors d'un accès légitime en écriture ou en lecture des caractéristiques et un attaquant actif peut accéder aux caractéristiques sans avoir à effectuer un appairage. Ce comportement a été vérifié sur la plateforme comportementale.

Le correctif a été soumis aux développeurs de la pile BlueZ et la CVE-2021-0129 a été attribuée.[10] Il convient de noter qu'une fois le correctif appliqué, les tests menés sont devenus conformes à ce qui était attendu.

7 Conclusion

La norme décrivant le Bluetooth Low Energy introduit parfois des exigences de sécurité qui ne sont pas explicitement propagées au travers des couches de la pile protocolaire. Un exemple significatif est le cas de la couche GAP qui définit un mode 1 niveau 4 imposant des mécanismes de sécurité (méthode d'appairage *Secure Connections*, authentification et chiffrement avec une taille de clé de 128 bits) sans que la couche GATT ne dispose de moyen pour stocker certaines de ces informations. Cela se traduit, entre autre, par une certaine latitude laissée à certains endroits aux développeurs des implémentations.

Une propagation complète et cohérente des propriétés de sécurité entre les couches du protocole est indispensable pour assurer le niveau de sécurité attendu. Cependant, peu de moyens permettent d'en vérifier la complétude et la cohérence. Par ailleurs, plusieurs attaques sur les appairages ont été démontrées récemment, remettant en cause les garanties de sécurité devant être obtenues en théorie d'après le standard. Par conséquent, la seule connaissance de la méthode d'appairage mise en œuvre n'est pas suffisante. La question de la détermination du niveau de sécurité réellement mis en place lors d'une session BLE en cours se pose alors naturellement.

Afin de répondre à cette problématique, une approche hybride combinant analyse statique de code source, analyse dynamique et analyse fonctionnelle a été mise en oeuvre sous la forme d'une plateforme d'analyse comportementale, ciblant BlueZ et Fluoride.

Cette plateforme comportementale a permis d'étudier de manière rapide et efficace les comportements des implémentations cibles face aux changements des exigences de sécurité. Il a également été possible d'analyser la façon dont les imprécisions normatives se sont répercutées dans les implémentations. Notre approche a permis d'identifier plusieurs non conformités qui ont été signalées aux développeurs de la pile BlueZ. Un reproche majeur peut être fait aux deux implémentations étudiées : il n'est pas possible de connaître le niveau de sécurité GAP en cours d'une connexion BLE, que celui qui se pose la question soit administrateur, utilisateur ou développeur d'application.

10. Le correctif se trouve ici : `https://git.kernel.org/pub/scm/bluetooth/bluez.git/commit/?id=00da0fb4972cf59e1c075f313da81ea549cb8738`.

Enfin, cette approche pourrait ouvrir la voie à de plus amples expérimentations, en multipliant les scénarios de test et en approfondissant l'exploration de la propagation des propriétés de sécurité, ce qui pourra à terme aboutir à la possibilité de fournir et de superviser les garanties de sécurité d'une connexion BLE en cours.

Références

1. Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. Key negotiation downgrade attacks on bluetooth and bluetooth low energy. volume 23, New York, NY, USA, June 2020. Association for Computing Machinery.
2. Eli Biham and Lior Neumann. Breaking the bluetooth pairing - fixed coordinate invalid curve attack. 2018. https://www.cs.technion.ac.il/~biham/BT/bt-fixed-coordinate-invalid-curve-attack.pdf.
3. Tristan Claverie and José Lopes-Esteves. Testing for weak key management in bluetooth low energy implementations. SSTIC 2020, 2020. https://www.sstic.org/media/SSTIC2020/SSTIC-actes/testing_for_weak_key_management_in_bluetooth_low_e/SSTIC2020-Article-testing_for_weak_key_management_in_bluetooth_low_energy_implementations-lopes-esteves_claverie.pdf.
4. Keijo Haataja and Pekka Toivanen. Practical man-in-the-middle attack against bluetooth secure simple pairing. 4th International Conference on Wireless Communications, Networking and Mobile Computing, 2008.
5. Andrew Y. Lindell. Attacks on the pairing protocol of bluetooth v2.1. BlackHat 2008 Symposium, 2008. https://www.blackhat.com/presentations/bh-usa-08/Lindell/BH_US_08_Lindell_Bluetooth_2.1_New_Vulnerabilities.pdf.
6. Max. https://scribles.net/creating-ble-gatt-server-uart-service-on-raspberry-pi/.
7. Kay Ren. Bluetooth pairing part 1 – pairing feature exchange, 2016. https://www.bluetooth.com/blog/bluetooth-pairing-part-1-pairing-feature-exchange/.
8. Mike Ryan. With low energy comes low security. 7th USENIX Workshop on Offensive Technologies, WOOT '13, Washington, D.C., USA, August 13, 2013, 2013.
9. Bluetooth SIG. *Bluetooth Core Specification 5.2*, 2019. https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=478726.
10. Wikipedia Team. https://en.wikipedia.org/wiki/Bluetooth_Low_Energy.
11. Yue Zhang, Jian Weng, Rajib Dey, Yier Jin, Zhiqiang Lin, and Xinwen Fu. Breaking secure pairing of bluetooth low energy using downgrade attacks. 29th Usenix Security Symposium, 06 2020. http://jin.ece.ufl.edu/papers/USENIX2020-BLE.PDF.