Ingénierie inverse d'une brosse à dents connectée

Axelle Apvrille
aapvrille@fortinet.com

Fortinet

Résumé. L'internet des objets (IoT) ou objets connectés n'est pas réputée pour sa forte sécurisation et sa protection de la vie privée. Dans cet article, j'étudie la sécurité d'un objet connecté particulièrement limité : une brosse à dents.
Bien qu'une brosse à dents soit un objet relativement simple, la tâche est probablement plus ardue que l'ingénierie inverse d'une caméra sur IP, par exemple, notamment parce qu'il n'existe aucune documentation technique à son sujet, encore moins des outils.
Pour arriver à mes fins, d'une part, je dé-compile le code de l'application mobile associée. D'autre part, j'implémente plusieurs outils : `talk2brush` pour communiquer en Bluetooth Low Energy avec la brosse à dents et par exemple modifier la vitesse du moteur, `talk2api` pour interagir avec le serveur applicatif distant et aussi un clone de brosse à dents, virtuel, basé sur un serveur Node.js avec le module Bleno.
Au cours de cette ingénierie inverse, plusieurs vulnérabilités sont identifiées. Exploitées, elles peuvent être monétisées, conduire à des fraudes à l'assurance, alimenter des bases de courriels à spammer, servir de dispositif de suivi d'un utilisateur. Cela démontre que même une brosse à dents - objet bénin qui parait dénué d'intérêt en termes de sécurité - *doit être sécurisé* sous peine d'intéresser un attaquant un peu plus original et déboucher à de bien réels délits.

1 Introduction

Cet article traite de la sécurité d'une brosse à dents connectée, la *Beam Brush*, commercialisée aux États-Unis par l'assurance dentaire *Beam Dental*.

Mais pourquoi parler d'une *brosse à dents* ? N'est-ce pas un peu... stupide ? Il y a deux raisons fondamentales. La première est que cela marque les esprits - par son originalité, et parce que c'est un objet qui touche tout le monde[1]. La deuxième raison est technique. Nous avons déjà vu via l'infection massive par le ver Mirai [6] qu'il n'était pas bon de faire l'impasse sur la sécurité d'objets connectés tels que des caméras sur IP

[1] Le titre par défaut du template LaTeXde SSTIC est la sécurité des trottinettes : la sécurité d'une brosse à dents, c'est encore mieux, non ?

ou des enregistreurs vidéo. Mais une brosse à dents connectée est bien plus limitée qu'une caméra sur IP : il n'y a pas, par exemple, de Linux embarqué. Si le manque de sécurisation d'une brosse à dents mène à la compromission d'équipements, de données ou d'individus, cela serait bien plus fort, car alors on pourrait prouver aux yeux de tous que *tous les objets doivent être sécurisés quels qu'ils soient*. Certes, il s'agit là d'un fait que la plupart des ingénieurs et chercheurs en sécurité ressentent intuitivement, mais ce n'est pas le cas des développeurs d'objets connectés, concepteurs ou hommes d'affaires.

Pour atteindre cet objectif, il faut étudier en détail la sécurité de l'objet connecté. Et pour cela, compte tenu de la proche inexistence de toute documentation technique (voir section 2), il n'y a guère d'autre choix que de débuter par de l'ingénierie inverse. C'est ce à quoi cet article s'attelle. Dans cet article, je présente une ingénierie inverse de type « boite noire » quasi complète de la *Beam Brush* et de l'application mobile qui lui est associée (section 3). D'un point de vue méthodologique, j'explique les outils que j'ai utilisés ou développés pour aboutir à mes fins (section 4). Au final, je peux contrôler la brosse à dents à distance (changer la vitesse de rotation de la tête par exemple), tricher et augmenter mes scores virtuels, cloner une brosse à dents (section 5). Certaines de ces attaques sont bénignes, d'autres peuvent être monétisées ou dériver vers de la fraude à l'assurance. En termes de vie privée, on arrive à parcourir la base des utilisateurs de la *Beam Brush*, récupérer leurs emails, photos - y compris celles d'enfants. Dans la section 6, je décris les grands axes qu'il me reste à explorer, comme faire véhiculer un virus à la brosse à dents. Enfin, je conclus sur la nécessité de sécuriser les objets connectés.

2 État de l'art

Il n'y a pas de documentation, spécifications, ni de forum utilisateur pour la *Beam Brush*. Les seules informations disponibles sont celles du site web de *Beam Dental* (`https://beam.dental`) et les articles de presse. On y apprend que la *Beam Brush* transmet ses informations via Bluetooth Low Energy (BLE), que les diverses parties de son enveloppe extérieure sont imprimés par une imprimante 3D et qu'elle est assemblée en salle blanche [10].

Sur la *Beam Brush*, les seules études de sécurité dont j'ai connaissance sont les miennes [1,2]. Elles portent sur l'intérêt d'inspecter les applications mobiles pour effectuer l'ingénierie inverse d'objets connectés. Un autre hacker, Arnab Nandi, s'est penché sur la *Beam Brush* mais pas sur l'angle

de la sécurité : il a développé un outil qui l'avertit si la brosse à dents est actuellement utilisée [7] et un autre qui affiche des couleurs différentes à l'écran d'un ordinateur portable en fonction de la couleur de *Beam Brush* utilisée [8] (la brosse à dents existe en couleur bleue, rose et verte). Ces outils montrent qu'il est possible de récupérer l'état de la brosse à dents et sa couleur. Dans cet article, outre l'aspect sécurité, nous allons bien plus loin et récupérons tous les paramètres connus de la brosse à dents.

Toujours sur les brosses à dents, mais chez Oral B, une vulnérabilité a été identifiée sur le système de monitoring sans fil *SmartGuide Professional Care 9900* [12] pour la brosse à dents *Oral B Triumph Toothbrush*. Le système de monitoring révèle des données personnelles comme la fréquence de brossage des dents. Il peut également être l'objet d'un déni de service.

Enfin, si l'on étend les recherches de l'état de l'art à la sécurité du protocole Bluetooth Low Energy, la bibliographie est plus garnie tant sur la sécurité [5, 9] que sur les outils [4, 11, 13].

3 Ingénierie inverse

3.1 Brosse à dents

La brosse à dents *Beam Brush* est un objet connecté relativement simple, qui ne fait qu'agir / réagir à des commandes Bluetooth Low Energy (BLE). Elle n'est pas directement reliée à Internet, mais faite pour converser avec une application mobile sur un téléphone ou tablette supportant le BLE.

Comme tout équipement supportant le BLE, la brosse à dents s'annonce au reste du monde en envoyant des paquets dits d'*advertising* ou en répondant aux demandes de scan. Ces paquets contiennent son adresse Bluetooth, indispensable à la poursuite de la communication, mais également son nom, « Beam Brush » (voir figure 1).

Pour interagir avec les fonctionnalités de la brosse à dents, il faut s'y connecter[2]. En BLE, cette connexion se fait par l'envoi d'une requête `CONNECT_REQ` du demandeur (par exemple le téléphone portable) vers l'adresse Bluetooth précédemment fournie. Cette connexion est transparente pour l'utilisateur, à ne pas confondre avec le *pairing* Bluetooth qui établit une relation de confiance entre deux entités. Une fois la connexion établie, l'ensemble des fonctionnalités de la brosse à dents devient disponible. Dans le monde BLE, les fonctions d'un objet sont décrites par son profil GATT (Generic Attribute Profile) [3], c'est-à-dire qu'elles sont

[2] NB. Ceci est une particularité de la *Beam Brush*. Notons que tous les périphériques BLE ne supportent pas de connexion.

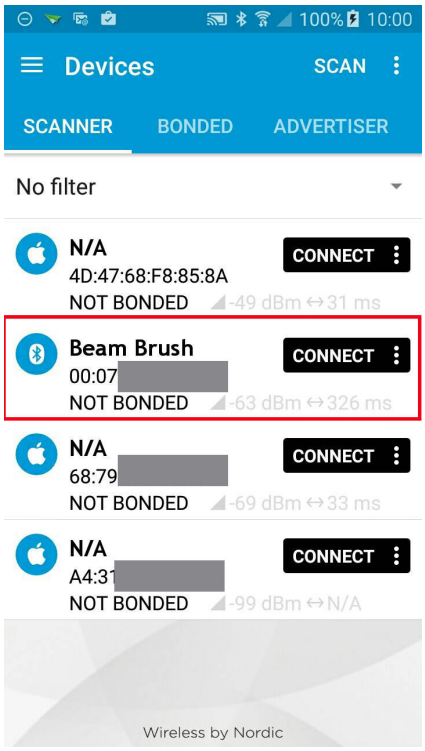

Fig. 1. Une brosse à dents *Beam Brush* vue par un scanneur BLE.

vues sous forme de *services* où un service est constitué d'un ensemble de *caractéristiques*.

La *Beam Brush* possède 6 services. Les 3 premiers sont standards, tandis que les 3 derniers sont spécifiques à la *Beam Brush*.

- **Generic Access**. Ce service standard contient dans notre cas précis une caractéristique pour le nom de l'objet *(Beam Brush)*, et son apparence (la couleur de la brosse à dents : bleu, rose, vert, inconnue).
- **Generic Attribute**. Ce service contient une seule caractéristique, « Service Changed », faite pour indiquer si un service a été ajouté, modifié ou enlevé.
- **Device Information**. Contient les caractéristiques suivantes : nom du vendeur (« Beam Technologies »), modèle (« BB-0002 »), numéro de série, version du firmware (« V0.28 »).
- **Firmware OTA**. Ce service gère les mises à jour de firmware *over-the-air* (OTA). Il révèle deux caractéristiques : l'une pour gérer l'état de la mémoire Flash (par exemple préparer la brosse à dents à recevoir un nouveau firmware), l'autre pour envoyer la mise à jour.
- **Beam**. Ce service est central à l'utilisation de la brosse puisque c'est lui qui gère presque tous les paramètres de la brosse à dents. Ces caractéristiques sont listées au tableau 1.

– **3D**. Ce service ne gère que les paramètres relatifs à l'accéléromètre, au gyroscope et au bouton marche/arrêt.

Avec BLE, les caractéristiques sont accessibles via un handle, dont la valeur peut éventuellement changer d'une brosse à dents à l'autre. On trouve ce handle en envoyant des requêtes `Find XXX`, par exemple `Find By Type Value Request`, ou de manière concrète via des outils tels que `gatttool`. Les caractéristiques sont accessibles en lecture, en écriture ou via des notifications spontanées de la part de l'équipement.

Fonction	UUID	R	W	N
Brossage actif en cours	a8902afd-4937-4346-a4f1-b7e71616a383	X		X
Durée	91c78eb7-a9c5-47d9-a51f-f0e55dfb42cc	X		X
Etat du moteur (éteint/allumé)	267b09fd-fb8e-4bb9-85cc-ade55975431b	X		X
Date et heure actuelle	3530b2ca-94f8-4a1d-96be-aa76d808c131	X	X	
Évènement de brossage	fddd3028-7292-4069-9db1-26e9d8d850d6			X
Index d'événement	9964e098-96c9-446e-b241-db8ac344eb4a	X	X	X
Vitesse du moteur	833da694-51c5-4418-b4a9-3482de840aa8	X	X	
Mise en veille automatique et avertissement pour changer de secteur buccal	19dc94fa-7bb3-4248-9b2d-1a0cc6437af5	X	X	
Niveau de la batterie	6dac0185-e4b7-4afd-ac6b-515eb9603c4c	X		
Couleur de la brosse à dents	0971ed14-e929-49f9-925f-81f638952193	X		
Version matérielle	cf16681d-78fd-431f-965e-c7b7b5d1d147	X		
Buzz (vibration courte)	ca048932-cec7-4b82-a086-f384c3270df5	X		

Tableau 1. Caractéristiques du service Beam avec leur handle, identifiant unique et propriétés d'accès : R(ead), W(rite), N(otify).

Voici quelques exemples concrets :

– **Faire vibrer courtement la brosse à dents** : trouver le handle de la caractéristique `ca048932-cec7-4b82-a086-f384c3270df5`, envoyer un paquet BLE `Write Command` à la brosse à dents, pour ce handle et avec la valeur 1.
– **Lire le niveau de la batterie** : chercher le handle de la caractéristique `6dac0185-e4b7-4afd-ac6b-515eb9603c4c`, envoyer un PDU `Read Request` à ce handle. La brosse à dents renvoie deux octets. En désassemblant l'application mobile (voir listing 1), on comprend que le niveau de la batterie est mesuré par un convertisseur analogique vers digital sur 12 bits et fonctionnant en 5V. Pour effectuer la conversion des deux octets vers un pourcentage de batterie, il ne faut conserver que les 12 bits du poids le plus fort, multiplier par 0.001221 (ce qui correspond à $5/2^{12}$) et ramener la valeur entre 0 et 100.

```
public static float getBatteryLevel(BluetoothGattCharacteristic
    charac){
  return ByteSerialize.somemax(ByteSerialize.a(1.1f,
    1.5f,
    (((float)(ByteSerialize.getUint16(charac, 0) >> 4))) *
        0.001221f,
    0f,
    1f));
}

private static float a(float arg2, float arg3, float arg4, float
     arg5, float arg6) {
  return (arg4 - arg2) / (arg3 - arg2) * (arg6 - arg5) + arg5;
}
```

Listing 1. Portion de code de l'application mobile lisant le niveau de la batterie.

- **Modifier la vitesse de vibration de la brosse à dents**. De même, on récupère le handle pour `833da694-51c5-4418-b4a9-3482de840aa8`, puis on envoie un PDU `Write Command` avec pour valeur une vitesse sur un octet entre `0xD0` (lent) et `0x45` (très rapide). Très exactement, la valeur est calculée à partir de la formule suivante : $(1 - pourcentage/100) * 8B_{16} + 45_{16}$.
- **Couper l'avertisseur de secteur buccal**. Récupérer le handle pour `19dc94fa-7bb3-4248-9b2d-1a0cc6437af5`, envoyer un PDU Write Command à ce handle, avec pour valeur un octet dont le bit de poids le plus faible est à 0.

Un autre service prometteur du point de vue de l'ingénierie inverse est le service des *mises à jour du firmware*. L'application mobile suit les étapes suivantes :

1. Télécharger les firmwares les plus récents et les stocker sur le téléphone mobile.
2. Si le firmware de la brosse à dents est antérieur, le mettre à jour. Pour cela, il faut « activer » la mémoire Flash en envoyant la valeur 4 à la caractéristique d'état du service. Puis, il faut progressivement amener la mémoire Flash dans un état où elle accepte l'écriture. Cela se fait apparemment en passant dans l'état ERASE0 (0), ERASE1 (1), INITIALIZE (2) puis WRITE. Enfin, envoyer le firmware sur l'autre caractéristique du service, puis terminer en rebootant la brosse à dents fait en envoyant la valeur 3 à la caractéristique d'état du service.

À l'heure actuelle, le firmware et le matériel restent des zones d'ombre à élucider (voir section 6). L'ingénierie inverse du logiciel mobile fournit

cependant plusieurs indications sur le matériel. On sait qu'il y a forcément une puce BLE, un vibreur, de la mémoire Flash, une horloge, un moteur, un gyroscope et un accéléromètre. Le gyroscope et l'accéléromètre ne sont pas utilisés à l'heure actuelle par le logiciel. Sur la brosse à dents en elle-même, ils servent probablement à générer l'indicateur de brossage actif. D'après mes tests, ils ne servent pas pour l'avertisseur de zone buccale, qui ne serait qu'un pur timer.

3.2 Service web

Tout le reste des informations de l'application mobile, comme le nom de l'utilisateur de la brosse à dents ou les étoiles et coeurs gagnés, ne sont que purement logiciels : la brosse à dents ne les connaît pas.

Ces informations sont gérées par un service distant, `https://app.beam.dental`, via une API de type REST. On peut créer de nouveaux comptes utilisateurs (depuis une même application mobile, on peut contrôler plusieurs brosses à dents, typiquement celles de tous les membres de la famille), ajouter des brosses à dents, changer de photo de profil, modifier son email, compléter des défis (petits jeux présents dans l'application - pour inciter au bon brossage et/ou passer le temps), etc.

L'ingénierie inverse de l'application mobile fournit toutes les URLs à utiliser dans chaque cas, et les arguments à y joindre. L'interface `BeamAPI` fournit une excellente vue d'ensemble (voir listing 2).

Dans la plupart des cas, la première chose à faire consiste à se logger. Il faut envoyer une requête HTTP POST à `http(s)://app.beam.dental/api/v1/users/token` avec les éléments suivants :

- client_id : valeur codée en dur, trouvée en dé-compilant l'application.
- client_secret : idem, codé en dur.
- grant_type : idem, codé en dur : `password`.
- username : mettre ici son identifiant.
- password : mettre ici son mot de passe.

Le serveur distant renvoie alors un objet au format JSON (voir listing 3).

Le `access_token` est à fournir comme preuve d'authentification dans toutes les requêtes qui le nécessitent. À noter : la durée de vie du token est très longue (plus de 2 jours).

Une fois authentifié, on peut notamment demander les dernières versions de firmware : on envoie un HTTP GET à `https://app.beam.dental/api/v1/firmwares/current?android_version=38` en incluant

```
package dental.beam.beamapp.beamapi;

import AUserGameInfo;
import AnImage;
import dde;
import ddf;
...
import retrofit.http.Query;

public interface BeamAPI {
    @FormUrlEncoded @POST(value="/users/token") void authenticate(
        @Field(value="username") String arg1, @Field(value="password
        ") String arg2, @Field(value="client_id") String arg3,
        @Field(value="client_secret") String arg4, @Field(value="
        grant_type") String arg5, Callback arg6);

    @POST(value="/devices") BeamAPIResponse createBrush(@Body dde
        arg1);

    @POST(value="/client_sessions") Response createClientSession(
        @Body ddg arg1);
```

Listing 2. Portion de code dé-compilé pour la Beam API.

```
{"meta":{},
 "payload":[{"access_token":"405ba6...",
                    "token_type":"bearer",
                    "expires_in":189345600}]}
```

Listing 3. Réponse à un login réussi.

```
...
 "payload":[{"id":"3389ca80-a666-0133-0db3-0242ac110197",
             "version":"0.28",
             "url":"https://xxxx/firmwares/Beam2-0.28-Release-Blue.
                 ota",
             "current":true,
             "color":"#00C9F0",
             "color_int":1,
             "crc32":3383421848,
             "updated_at":"2016-01-26T14:22:49.000Z",
             "created_at":"2016-01-26T14:22:49.000Z"},
            {"id":"338ceb20-a666-0133-0db5-0242ac110197",
              "version":"0.28",
              "url":"https://xxxx/firmwares/Beam2-0.28-Release-Pink.
                  ota",
...
```

Listing 4. Réponse du serveur listant les firmwares les plus récents.

le token d'authentification dans une entête de la requête (`Authorization: Bearer xxxx`). Le serveur répond avec les URLs des différents modèles de firmware (voir listing 4).

De nombreuses autres actions sont possibles, je détaillerai les plus intéressantes d'un point de vue sécurité à la section 5.

4 Outils

La majeure partie du travail d'ingénierie inverse s'effectue comme chacun le sait par la patiente relecture du code désassemblé.

Sur iOS, les applications étant chiffrées, j'ai d'abord dû utiliser *Clutch* (`https://github.com/KJCracks/Clutch`) pour récupérer un paquet que je désassemble ensuite avec IDA Pro. Le segment *objc* du binaire de l'application est particulièrement intéressant à lire car, pour chaque classe, il liste les méthodes (avec leur prototype) et variables. Cela donne une idée précise de l'implémentation de l'application.

Sous Android, de nombreux outils sont disponibles comme `apktool`, `baksmali`. J'ai personnellement utilisé le décompileur JEB (`https://www.pnfsoftware.com/`).

Pour lever certains doutes, j'ai aussi procédé à un peu d'analyse dynamique. Notamment, j'ai écouté les paquets BLE à l'aide d'un petit dongle USB peu onéreux : le *Bluefruit* d'Adafruit (`https://www.adafruit.com/products/2269`). Ce dongle possède une puce BLE de Nordic Semiconductor et un sniffer Python a été développé pour. On insère le dongle, lance le sniffer, ce dernier liste les équipements BLE qu'il voit, on choisit celui qu'on veut écouter et le sniffer constitue un fichier de capture réseau (.pcap). Cette capture réseau peut ensuite être lue à l'aide de Wireshark (voir figure 2).

Cependant, un sniffer ne permet que d'écouter les trames, pas de les émettre. Je me suis donc penchée sur BtleJuice [4], un logiciel pour intercepter et faire du man-in-the-middle sur les paquets BLE. Je l'ai ensuite abandonné car je peinais à le mettre en oeuvre (il faut deux instances notamment) et surtout parce qu'en réalité je n'avais pas vraiment besoin de faire de l'interception de paquets, juste de lire et émettre.

J'ai alors développé mes propres outils, dont l'architecture générale est illustrée à la figure 3.

Notamment, j'ai écrit un programme appelé `talk2brush` (figures 3 et 4) qui peut communiquer avec toutes les caractéristiques BLE de la brosse à dents, y compris de la faire envoyer des messages en Morse en variant l'intensité du moteur pour coder les points et les ti-

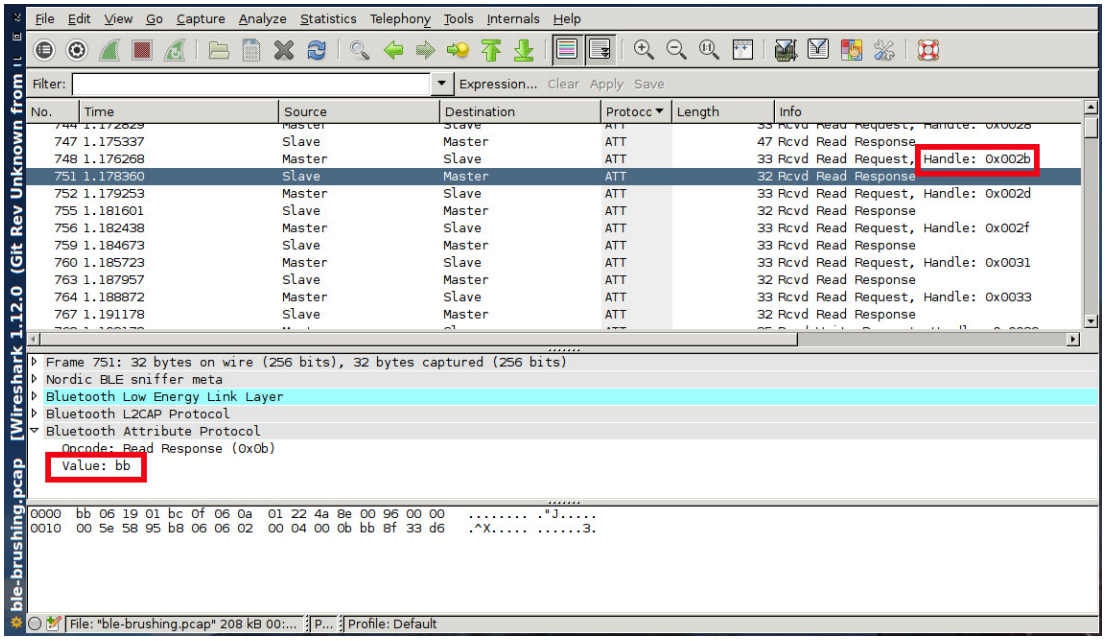

Fig. 2. Capture de paquets BLE pour ou de la brosse à dents. En particulier, on a demandé l'intensité actuelle du moteur, et la brosse à dents a répondu 0xbb ce qui correspond environ à 15%.

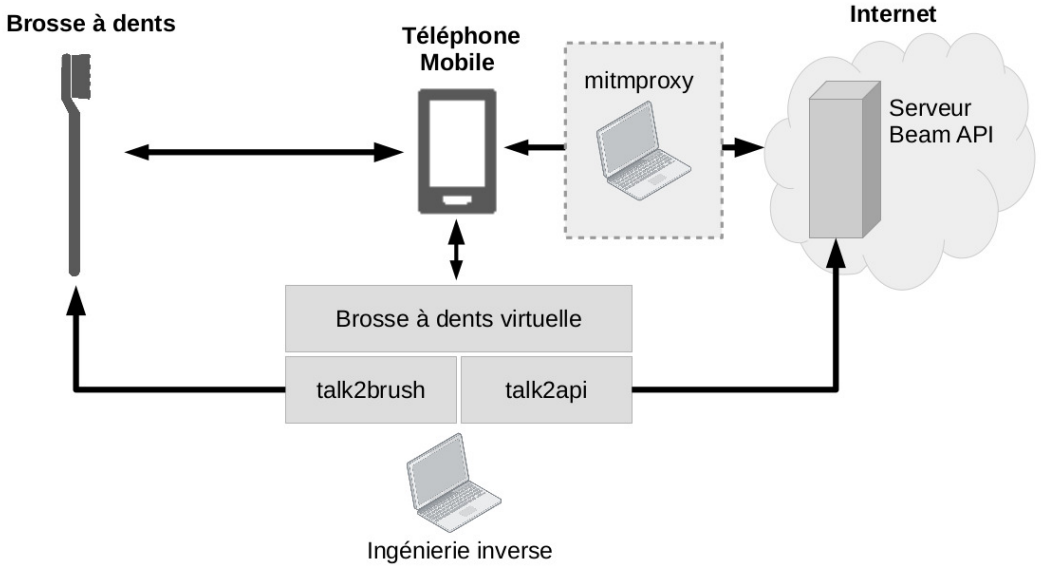

Fig. 3. Architecture générale d'étude de la *Beam Brush*. `talk2brush`, `talk2api` et le programme de brosse à dents virtuelle sont chacun indépendants les uns des autres et peuvent fonctionner individuellement. `itmproxy` est à mettre en place si l'on a besoin d'écouter les requêtes HTTPS.

rets. Ce programme repose sur des librairies existantes gattlib (`https://github.com/labapart/gattlib`) et pygattlib (`https://bitbucket.org/OscarAcena/pygattlib`) qui m'évitent d'entrer dans les détails des paquets Bluetooth. Par exemple, au listing 5, j'active les notifications du bouton de la brosse à dents. Chaque fois que le bouton est enfoncé ou relâché, une notification est émise, qui appelle une *callback* spécifique.

```
def enable_button_notif(self, verbose=False):
    hexstring = str(bytearray([01, 00]))
    notif_handle = 0x003f
    self.req.write_by_handle(notif_handle, hexstring)
    if verbose:
        print "[+] Will receive button state notifications"
```

Listing 5. Activation des notifications à l'aide de pygattlib.

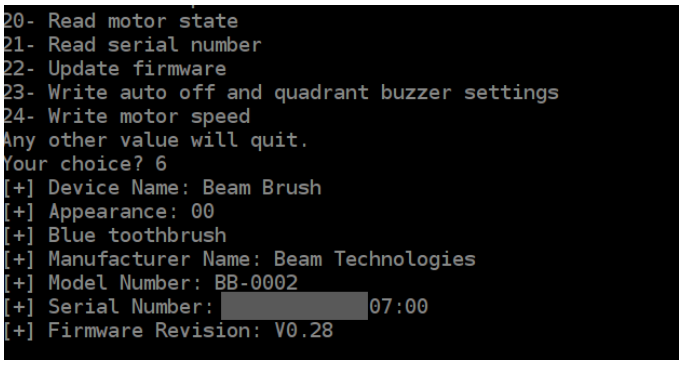

Fig. 4. `talk2brush` offre un menu de plusieurs options. On le voit ici qui a récupéré le nom de la brosse, modèle, numéro de série, version.

Il est également utile de faire l'inverse : se faire passer pour une brosse à dents. Cela permet de bien comprendre les réactions de l'application mobile. J'ai donc commencé à implémenter une brosse à dents virtuelle. Cette dernière n'est pas complète à l'heure actuelle, mais je peux d'ores et déjà m'annoncer comme une *Beam Brush*, être vu et ajouté par l'application mobile comme une *Beam Brush* d'une couleur donnée et faire virtuellement varier le moteur de la brosse à dents virtuelle (figures 5 et 6).

L'implémentation de cette brosse à dents virtuelle repose sur Bleno (`https://www.npmjs.com/package/bleno`), un module Node.js pour créer des périphériques BLE. Pour une meilleure portabilité, le serveur Node.js (ainsi que Bleno et mes fichiers Javascript pour la brosse à dents) tourne dans un container Docker. L'émission physique des paquets BLE est effectuée par un simple dongle USB BLE (coût : entre 5 et 10 euros). D'une certaine manière, ce dongle devient la brosse à dents virtuelle.

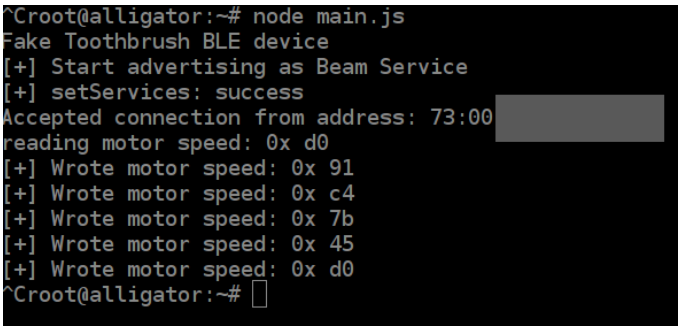

Fig. 5. Ce shell représente une brosse à dents factice. En parallèle, avec l'application mobile, j'ai modifié l'intensité du moteur à plusieurs reprises. On voit que la brosse à dents factice reçoit bien les requêtes, et modifie l'intensité (factice) en conséquence : `0x91`, `0xc4`, `0x7b`, `0x45` (intensité max), `0xd0` (min).

Fig. 6. La brosse à dents factice est vue comme une vraie depuis l'application mobile. Notamment, l'application mobile détecte bien sa couleur (rose).

Enfin, il reste l'aspect communication entre l'application mobile et le serveur distant de Beam. Pour cela, j'ai procédé de deux manières complémentaires.

1. La communication vers le serveur distant `https://app.beam.dental/api/v1` est automatiquement redirigée sur HTTPS si bien que les requêtes envoyées ou reçues ne sont pas facilement lisibles puisque chiffrées. L'outil `mitmproxy` (`http://mitmproxy.org`) est une solution à ce problème. Il s'agit d'une console interactive qui effectue un man-in-the-middle pour HTTPS. D'un côté, on fait tourner le proxy `mitmproxy` sur une machine. De l'autre, on connecte le téléphone portable sur le même réseau wifi, et on le configure pour utiliser l'autre machine comme proxy pour toutes les requêtes. Sur le téléphone, on ajoute également en certificat de confiance un certificat généré par `mitmproxy`. Toutes les requêtes HTTPS du téléphone vers l'extérieur transitent alors d'abord vers le proxy, qui se pose en « homme du milieu », lit et affiche les requêtes dans la console (éventuellement peut aussi les filtrer et les modifier) et les envoie vers leur destination finale (voir figure 7).

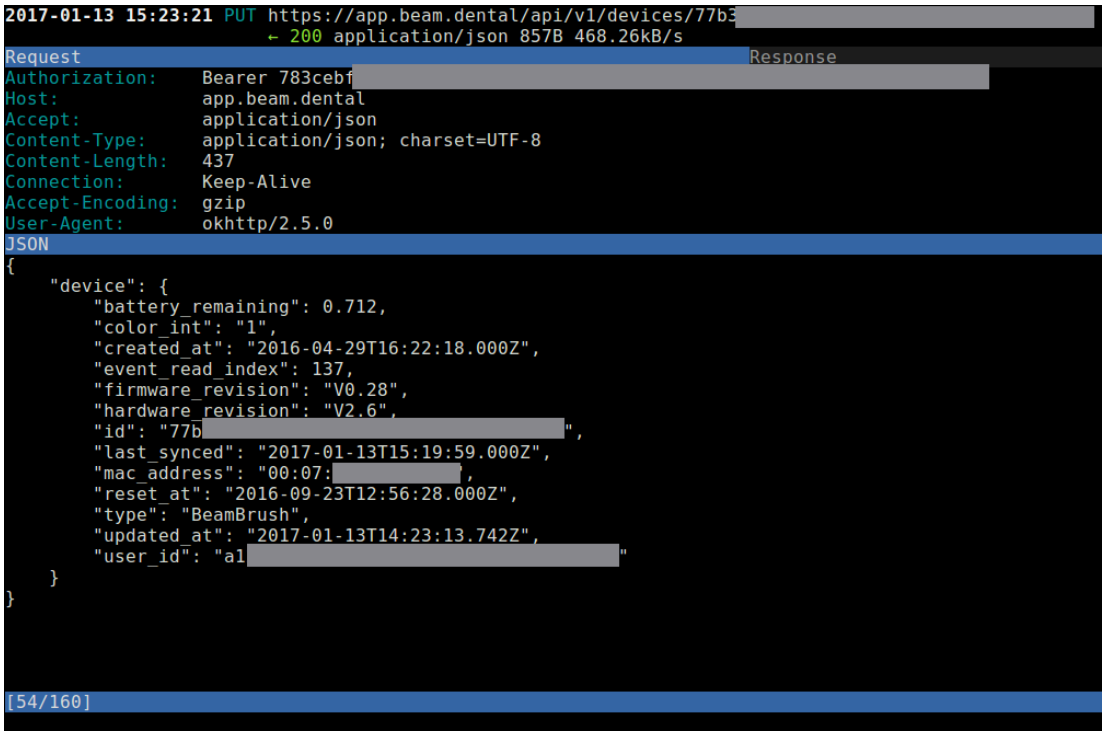

Fig. 7. `mitmproxy` capture une requête HTTP PUT qui contient des informations sur la brosse à dents : adresse Bluetooth, version, batterie restante, etc.

2. J'ai écrit un programme `talk2api` qui envoie et reçoit des requêtes HTTP vers `https://app.beam.dental/api/v1`. Par exemple, je peux me logger, délogger, modifier mon nom mais aussi effectuer quelques attaques qui seront détaillées à la section 5.

L'utilisation de ces deux outils, `mitmproxy` et `talk2api`, a grandement facilité l'ingénierie inverse de la *Beam API*. Par exemple, dans le code dé-compilé, je peinais à voir le format exact des objets JSON à envoyer au serveur distant : `mitmproxy` les a affichés (voir figure 7).

Les 3 outils que j'ai développés, `talk2brush`, `talk2api` et la brosse à dents factice[3], peuvent servir de base à l'ingénierie inverse d'autres objets connectés. Il suffira d'adapter les requêtes BLE et HTTP au cas.

5 Résultats

L'ingénierie inverse de la *Beam Brush* révèle un certain nombre de choses :

- **Les requêtes BLE ne sont aucunement sécurisées** (alors que le protocole BLE prévoit plusieurs mécanismes). Il n'y a pas d'authentification, pas de chiffrement, pas de contrôle d'intégrité et le constructeur ne prévoit pas non plus de génération aléatoire de l'adresse Bluetooth de la brosse à dents pour éviter le suivi de son propriétaire.
- **Certaines requêtes vers l'API distante ne nécessitent pas d'authentification**. Il résulte notamment une **fuite d'information importante** sur les utilisateurs de *Beam Brush* - voir le tableau 2. *Beam Dental* a été averti de ces vulnérabilités, a fini par répondre après plusieurs relances. À l'heure actuelle, il n'est pas clair s'ils vont corriger les failles, et sous quels délais.
- **Aucune donnée dentaire**. À aucun moment l'application mobile ou la brosse à dents n'enregistre le nombre de dents, quelles dents exactement sont brossées ou pas, l'état des dents, etc. Les seules données qui sont conservées sont le nom de l'utilisateur, son email, sa police d'assurance, ses ayant-droits, les âges, adresses, etc. C'est un peu surprenant pour un service qui se veut améliorer l'hygiène dentaire, mais au moins cela fait des données sensibles en moins...

Le tableau 2 présente les actions possibles sur la brosse à dents ou l'application mobile, ainsi que les conditions nécessaires et l'intérêt éventuel pour un attaquant.

[3] Ces outils seront rendus disponibles dans la mesure du possible par rapport à l'avancée des corrections de *Beam Dental*.

Condition : être à proximité (150m maximum) de la brosse à dents	
Action	**Intérêt pour un attaquant**
Faire une vibration courte (buzz)	Très faible
Activer ou désactiver la mise en veille automatique	Très faible
Activer ou désactiver l'avertisseur de changement de secteur buccal	Très faible
Communiquer en Morse	Très faible (mais amusant)
Modifier la vitesse du moteur	Faible. Rançon, déni de service en vidant la batterie
Surveiller le brossage de dents d'un individu	Faible
Modifier la durée du brossage de dents ou créer de nouveaux événements de brossage	Moyenne. **Fraude à l'assurance envisageable**
Suivi physique d'un détenteur de *Beam Brush*	Moyenne à forte. **Atteinte à la vie privée**

Réalisable à distance, sans condition particulière	
Action	**Intérêt pour un attaquant**
Demande de ré-initialisation de mot de passe (envoie un email de confirmation à l'utilisateur)	Faible. Surcharge éventuelle du serveur de Beam si ré-initialisation faites en masse
Récupération du nom et de la photo d'un utilisateur Beam à partir de son email	Moyenne. **Atteinte à la vie privée**.
Récupération de noms d'utilisateur Beam, emails, photos, adresses Bluetooth, couleur de brosse à dents, date de création du compte, date de modification du compte	Moyenne à forte. **Atteinte à la vie privée, spam, rançons, espionnage, établissement de profil pour des publicités...**

Réalisable à distance à condition de posséder un compte *Beam Brush*	
Action	**Intérêt pour un attaquant**
Création, modification, suppression de nouveaux utilisateurs	Très faible, car on ne peut créer que des comptes dépendants du compte principal
Modifier le nombre de coeurs virtuels, ou le niveau de jeu, ou la progression ou les étoiles de jeu de son compte ou de ses ayant-droits (exemple à la figure 8)	Moyenne. **Monétisation envisageable** via des services tels que Pact, Achievemint, Higi, FitStudio. Ou **fraude à l'assurance**.
Futur : Infection d'une brosse à dents	Moyen. Déni de service, rançon, fraude à l'assurance...
Futur : Faire véhiculer un virus à une brosse à dents	Élevé. Participation éventuelle à un botnet, à des campagnes de dénis de service, spam, etc.

Tableau 2. Attaques identifiées sur la *Beam Brush*. En gras, les plus importantes.

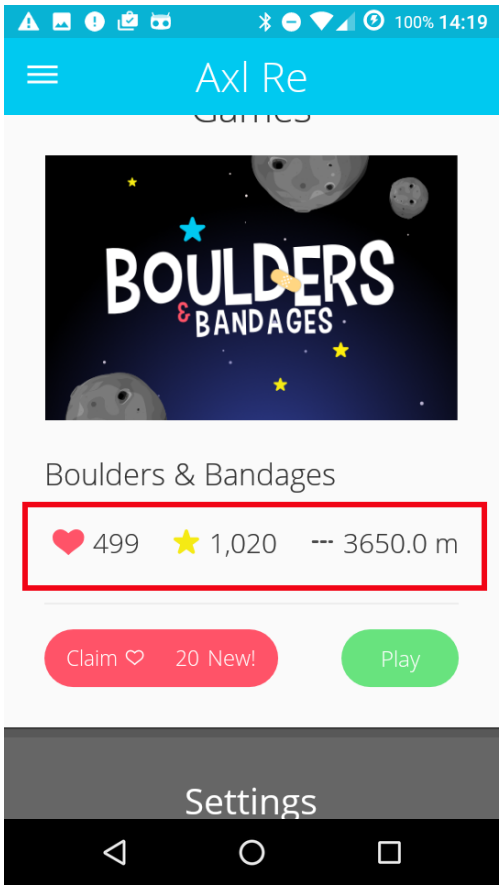

Fig. 8. Exemple de compte utilisateur où j'ai modifié à souhait le nombre de coeurs, étoiles et distances. Il serait difficile d'arriver exactement à 3650 mètres...

Certaines attaques n'ont que peu d'intérêt du point de vue de la sécurité, d'autres ont plus d'impact. Je ne détaillerai que les 3 les plus intéressantes :

1. À cause d'une erreur de conception, identifiée grâce à `talk2api`, il est possible de **brute forcer le serveur distant et lister les utilisateurs de la *Beam Brush*** ainsi que de multiples autres informations : email, photo, adresses MAC, couleur de brosse à dents, date de création du compte... Cette attaque n'étant pas corrigée à l'heure actuelle, je m'abstiens d'expliquer comment elle fonctionne, mais elle est **réalisable à distance et sans être affilié à *Beam Dental***. Je l'ai testée quelques secondes pour vérifier et prouver son fonctionnement et ai récupéré 9 profils d'utilisateur, certains étant des *enfants*.
2. L'adresse Bluetooth de la brosse à dents est fixe. Par conséquent, un espion qui ne fait que scanner les appareils BLE aux alentours sait qu'une *Beam Brush* est à proximité, et il peut le suivre dans ses

déplacements. Pire, **le suivi peut se faire que la brosse à dents soit allumée ou pas**. Le seul moyen d'empêcher la brosse à dents de répondre au scan BLE est de lui ôter sa pile... **La brosse à dents devient donc, à l'insu de son propriétaire, un équipement de surveillance**. On peut imaginer par exemple des scanneurs BLE postés dans des gares ou aéroports : on saura par conséquent où le propriétaire s'est déplacé et à quelle heure. Cette faille est connue depuis longtemps, et les spécifications Bluetooth supportent même une génération aléatoire d'adresse pour éviter le suivi de l'appareil, mais ceci n'est pas implémenté dans la *Beam Brush*. [5] détaille très bien le problème et propose une solution, appelée BLE-Guardian, pour les équipements qui ont « oublié » d'implémenter la génération aléatoire d'adresse prévue dans les spécifications.

3. Il est possible de **modifier la durée de son propre brossage de dents, voire de créer des événements de brossage totalement fictifs**. Sachant que les *Beam Brush* sont fournies par une assurance dentaire, le propriétaire peu éthique pourrait vouloir faire croire à son assurance qu'il se brosse scrupuleusement les dents (alors que ce n'est pas le cas). On peut supposer que l'assurance dentaire - ou une autre assurance médicale affiliée - *serait tentée de diminuer le tarif de son assurance*. Initialement, *Beam Dental* assurait le contraire, mais la dérive était tentante et la firme semble maintenant avoir changé d'avis et affiche en gros la devise suivante sur son site web *« C'est simple, le mieux vous vous brossez les dents, le moins vous payez »*[4].

6 Futur

Le travail sur la *Beam Brush* est très avancé, mais il reste tout de même quelques zones d'ombres sur lesquelles je compte me pencher prochainement :

– Ingénierie inverse du matériel. Je n'ai pour l'instant pas voulu ouvrir la brosse à dents n'en ayant qu'une seule. Les *Beam Brush* ne sont en théorie pas achetable publiquement, il faut souscrire à une assurance dentaire via son entreprise aux États-Unis. J'ai acquis ma première *Beam Brush* par chance sur *eBay*, et guette un deuxième modèle depuis plusieurs mois.

[4] Traduit de l'anglais : *"It's easy - the better you brush the less you pay"* - rapatrié le 17 mars 2017 de `https://beam.dental/plans`.

- Firmware. Actuellement l'analyse du firmware n'a rien donné. Il est possible que les fichiers que j'ai ne soient pas des firmwares complets, mais seulement des mises à jour. Dès que j'aurai une deuxième *Beam Brush*, je compte essayer de changer son firmware, charger un firmware modifié et j'espère ainsi mieux le comprendre.
- Infection. Dans le tableau 2, j'ai noté deux attaques intéressantes sur lesquelles il me reste à travailler : il s'agit d'infecter la brosse à dents voire de lui faire véhiculer un malware. Pour envoyer du code malicieux à la brosse à dents, il faut des requêtes et réponses BLE pouvant contenir des données arbitraires. Je me suis penchée sur les requêtes d'advertising et de scan BLE, mais hélas chaque octet des requêtes est utilisé de manière fixe, seule la réponse `SCAN_RSP` comporte une zone pour des données arbitraires. Ceci n'est pas suffisant et donc pas utilisable. Les requêtes pour les caractéristiques de date et index d'événement sont éventuellement utilisables, mais elles sont de tailles limitées, il me faudra investiguer pour voir si cela est suffisant, si je peux tronquer le code malicieux dans plusieurs paquets. Enfin, une autre option que j'envisage est l'infection de la brosse à dents par une mise à jour vérolée. Cet axe est prometteur mais nécessite une meilleure compréhension du firmware de la brosse à dents.

7 Conclusion

Sans grande surprise, la brosse à dents *Beam Brush* n'est pas correctement sécurisée et met à mal la vie privée de son propriétaire.

Les données issues de la brosse à dents sont aisément falsifiables (manque de contrôle d'authenticité), ses communications Bluetooth Low Energy sont en clair, et quiconque est à proximité peut interagir avec elle, ou suivre son propriétaire. La seule faible attention à la sécurité se trouve sur le serveur applicatif distant, mais, naïve, elle se limite à l'utilisation d'HTTPS et à une authentification à base de jeton requise pour certaines actions (mais pas toutes).

Les attaques qui en découlent sont loin d'être dénuées d'intérêt pour un attaquant. Par effet de bord parfois surprenant, certaines peuvent être monétisées (coeurs, étoiles virtuelles revendues ou transformées en biens), déboucher sur des fraudes à l'assurance (je me brosse bien les dents, donc vous pouvez réduire ma cotisation), servir de dispositif de suivi (à l'insu du propriétaire et non désactivable, sauf en enlevant la pile de la brosse) ou encore alimenter des bases de données d'emails à spammer. Ces résultats, obtenus sans même ouvrir la brosse à dents, démontrent son faible niveau

de sécurité et devrait inciter les constructeurs à mieux tester leur produit. L'exemple de cette brosse à dents montre également qu'il faut **penser à la sécurité dès la conception de l'objet** : c'est beaucoup plus difficile - et plus couteux - de rattraper un mauvais design. Dans le cas de la *Beam Brush* par exemple, le manque de contrôle d'authenticité des données est difficile à résoudre sans changer profondément le matériel et les protocoles de communication.

Que faire pour sécuriser le monde des objets de demain ? Mon prochain axe d'étude portera sur la compromission / infection de la brosse à dents, mais je ne manquerai pas d'étudier la sécurité d'autres objets connectés également. Idéalement, tous les objets connectés devraient être scrutés, sans oublier les plus anodins qui sont parfois trompeurs. Les vulnérabilités remontées servent directement à améliorer la sécurité d'un modèle donné, mais elles renforcent également les connaissances en sécurité des fabricants. **Certains d'entre eux sont actuellement très novices** dans ce domaine, ne savent pas ce qu'est une notification de vulnérabilité, la méprennent pour du spam, n'y répondent jamais, ou acceptent mal qu'on teste *leur* produit.

En tant que chercheurs, nous pouvons également travailler sur des mécanismes qui facilitent le développement de la sécurité des objets connectés. Comment prouver l'authenticité de mesures effectuées par du matériel connecté ? Comment restreindre l'accès à des services Bluetooth ? etc. Si les objets connectés sont mal sécurisés à l'heure actuelle, c'est aussi parce que c'est difficile à faire et que les constructeurs ne savent pas comment s'y prendre.

Enfin, à plus haut niveau, une charte ou un label sécurité pour les objets connectés seraient la bienvenue, tant pour pour les chercheurs du domaine que le consommateur. Cette charte pourrait garantir l'utilisation d'une proportion du budget à la sécurité, définir les contacts privilégiés pour les soucis de sécurité, le délai pour la correction de failles, etc.

Références

1. Axelle Apvrille. Mobile Applications : a Backdoor into Internet of Things ? In *Virus Bulletin Conference*, pages 201–208, Octobre 2016. Denver, CO, USA.
2. Axelle Apvrille. Reversing Internet of Things from mobile applications. `http://insomnihack.ch/wp-content/uploads/2016/04/inso16_mobileiot.pdf`, Mars 2016. Insomni'hack.
3. Specifications of the Bluetooth System, Juin 2010. Core Package Version 4.0.
4. Damien Cauquil. BtleJuice : the Bluetooth Smart Man In The Middle Framework. In *Hack.lu conference*, Octobre 2016. Luxembourg.

5. Kassem Fawaz, Kyu-Han Kim, and Kang G. Shin. Protecting privacy of BLE device users. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 1205–1221, 2016.
6. Jerry Gamblin. Leaked Mirai Source Code for Research/IoC Development Purposes, Octobre 2016.
7. Arnab Nandi. Beambrush brushing status reporter. `https://gist.github.com/arnabdotorg/8324701ef96282509783`, 2015.
8. Arnab Nandi. Interactive Art using Node.Js and Socket.io. `https://twitter.com/arnabdotorg/status/800289667694424064`, Novembre 2016. HackOhio.
9. Mike Ryan. Bluetooth : With low energy comes low security. In *Proceedings of the 7th USENIX Conference on Offensive Technologies*, WOOT'13, pages 4–4, Berkeley, CA, USA, 2013. USENIX Association.
10. Katy Smith. Slideshow : How Bluetooth-enabled toothbrushes get made. `http://www.bizjournals.com/columbus/blog/2015/05/slideshow-how-bluetooth-enabled-toothbrushes-get.html`, Mai 2015.
11. Stian. BLE Sniffer in Linux using Wireshark. `https://devzone.nordicsemi.com/blogs/750/ble-sniffer-in-linux-using-wireshark/`, Septembre 2015.
12. Craig Wright. CSW Security Advisory 0002 : Oral B SmartMonitor Information Disclosure Vulnerability and DoS. `http://www.securityfocus.com/archive/1/493475/30/0/threaded`.
13. Candid Wueest. Attack points in health apps & wearable devices - how safe is your quantified self ? In *Virus Bulletin conference*, Septembre 2014. Seattle, USA.