RFC 9019 : A Firmware Update Architecture for Internet of Things

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 1 mai 2021

Date de publication du RFC : Avril 2021

https://www.bortzmeyer.org/9019.html

On le sait, le « S » dans « IoT » veut dire « sécurité ». Cette plaisanterie traditionnelle rappelle une vérité importante sur les brosses à dents connectées, les télés connectées, les voitures connectées et les sextoys connectés : leur sécurité est catastrophique. Les vendeurs de ces objets ont vingt ou trente ans de retard en matière de sécurité et c'est tous les jours qu'un nouveau piratage spectaculaire d'un objet connecté est annoncé. Une partie des vulnérabilités pourrait être bouchée en mettant à jour plus fréquemment les objets connectés. Mais cela ne va pas sans mal. Ce RFC fait partie d'un projet qui vise à construire certaines briques qui seront utiles pour permettre cette mise à jour fréquente. Il définit l'architecture générale.

Une bonne analyse du problème est décrite dans le RFC 8240[1], qui faisait le compte-rendu d'un atelier de réflexion sur la question. L'atelier avait bien montré l'ampleur et la complexité du problème ! Notre nouveau RFC est issu du groupe de travail SUIT <https://datatracker.ietf.org/wg/suit/> de l'IETF, créé suite à cet atelier. La mise à jour du logiciel des objets connectés compte beaucoup d'aspects, et l'IETF se focalise sur un format de manifeste (en CBOR) qui permettra de décrire les mises à jour. (Le format effectif sera dans un futur RFC.)

Le problème est compliqué : idéalement, on voudrait que les mises à jour logicielles de ces objets connectés se passent automatiquement, et sans casse. Ces objets sont nombreux, et beaucoup d'objets connectés sont installés dans des endroits peu accessibles ou, en tout cas, ne sont pas forcément bien suivis. Et ils n'ont souvent pas d'interface utilisateur du tout. Une solution réaliste doit donc être automatique (on sait que les injonctions aux humains « mettez à jour vos brosses à dents connectées » ne seront pas suivies). D'un autre côté, cette exigence d'automaticité va mal avec celle de consentement de

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc8240.txt

l'utilisateur, d'autant plus que des mises à jour peuvent éliminer certaines fonctions de l'objet, ou en ajouter des peu souhaitables (cf. RFC 8240). En moins grave, elles peuvent aussi annuler des réglages ou modifications faits par les utilisateurs.

Le format de manifeste sur lequel travaille le groupe SUIT vise à authentifier l'image (le code), par exemple en la signant. Un autre point du cahier des charges, mais qui n'est qu'optionnel [et que je trouve peu réaliste dans la plupart des environnements] est d'assurer la confidentialité de l'image, afin d'empêcher la rétro-ingénierie du code. Ce format de manifeste vise avant tout les objets de classe 1 (selon la terminologie du RFC 7228). Il est prévu pour du logiciel mais peut être utilisé pour décrire d'autres ressources, comme des clés cryptographiques.

On l'a dit, le travail du groupe SUIT <https://datatracker.ietf.org/wg/suit/> se concentre sur le format du manifeste. Mais une solution complète de mise à jour nécessite évidemment bien plus que cela, le protocole de transfert de fichiers pour récupérer l'image, un protocole pour découvrir qu'il existe des mises à jour (cf. le "*status tracker*" en section 2.3), un pour trouver le serveur pertinent (par exemple avec DNS-SD, RFC 6763). Le protocole LwM2M <http://www.openmobilealliance.org/release/LightweightM2M/V1_0_2-20180209-A/OMA-TS-LightweightM2M-V1_0_2-20180209-A.pdf> ("*Lightweight M2M*") peut par exemple être utilisé.

Et puis ce n'est pas tout de trouver la mise à jour, la récupérer, la vérifier cryptographiquement. Une solution complète doit aussi fournir :

- La garantie que la mise à jour ne va pas tout casser (si beaucoup d'utilisateurs débrayent les mises à jour automatiques, ce n'est pas par pure paranoïa), ce qui implique des mises à jour bien testées, et une stratégie de repli si la mise à jour échoue,
- des mises à jour rapides, puisque les failles de sécurité sont exploitées dès le premier jour, alors que les mises à jour nécessitent souvent un long processus d'approbation, parfois pour de bonnes raisons (les tests dont je parlais au paragraphe précédent), parfois pour des mauvaises (processus bureaucratiques), et qu'elles doivent souvent passer en cascade par plusieurs acteurs (bibliothèque intégrée dans un système d'exploitation intégré dans un composant matériel intégré dans un objet, tous les quatre par des acteurs différents),
- des mises à jour économes en énergie, l'objet pouvant être contraint dans ce domaine,
- des mises à jour qui fonctionnent même après que les actionnaires de l'entreprise aient décidé de mettre leur argent ailleurs et aient abandonné les objets vendus (on peut toujours rêver).

Bon, maintenant, après cette liste au Père Noël, au travail. Un peu de vocabulaire pour commencer :

- Image (ou "*firmware*") : le code qui sera téléchargé sur l'objet et installé (je n'ai jamais compris pourquoi ça s'appelait « ROM » dans le monde Android). L'image peut être juste le système d'exploitation (ou une partie de celui-ci) ou bien un système de fichiers complet.
- Manifeste : les métadonnées sur l'image, par exemple date, numéro de version, signature, etc.
- Point de départ de la validation ("*trust anchor*", RFC 5914 et RFC 6024) : la clé publique à partir de laquelle se feront les validations cryptographiques.
- REE ("*Rich Execution Environment*") : un environnement logiciel général, pas forcément très sécurisé. Par exemple, Alpine Linux est un REE.
- TEE ("*Trusted Execution Environment*") : un environnement logiciel sécurisé, typiquement plus petit que le REE et doté de moins de fonctions.

Le monde de ces objets connectés se traduit par une grande dispersion des parties prenantes. On y trouve par exemple :

- Les auteurs de logiciel, qui sont souvent très loin du déploiement et de la maintenance des objets,
- les fabricants de l'objet,
- les opérateurs des objets qui ne sont **pas** les propriétaires de l'objet : la plupart du temps, pour piloter **votre** objet, celui que vous avez acheté, vous devez passer par cet opérateur, en général le fabricant, s'il n'a pas fait faillite ou abandonné ces objets,
- les gestionnaires des clés (souvent le fabricant),
- et le pauvre utilisateur tout en bas.

[Résultat, on a des chaînes d'approvisionnement longues, compliquées, opaques et vulnérables. Une grande partie des problèmes de sécurité des objets connectés vient de là, et pas de la technique. Par exemple, il est fréquent que les auteurs des logiciels utilisés se moquent de la sécurité et ne fournissent pas de mise à jour pour combler les failles dans les programmes.]

La section 3 du RFC présente l'architecture générale de la solution. Elle doit évidemment reposer sur IP (ce RFC étant écrit par l'IETF, un choix contraire aurait été étonnant), et, comme les images sont souvent de taille significative (des dizaines, des centaines de kilo-octets, au minimum), il faut utiliser en prime un protocole qui assurera la fiabilité du transfert et qui ne déclenchera pas de congestion, par exemple TCP, soit directement, soit par l'intermédiaire d'un protocole applicatif comme MQTT ou CoAP. En dessous d'IP, on pourra utiliser tout ce qu'on veut, USB, BLE, etc. Outre IP et TCP (pas évident pour un objet contraint, cf. RFC 9006), l'objet devra avoir le moyen de consulter le "*status tracker*" pour savoir s'il y a du nouveau, la capacité de stocker la nouvelle image avant de l'installer (si la mise à jour échoue, il faut pouvoir revenir en arrière, donc stocker l'ancienne image et la nouvelle), la capacité de déballer, voire de déchiffrer l'image et bien sûr, puisque c'est le cœur du projet SUIT, la capacité de lire et de comprendre le manifeste. Comme le manifeste comprend de la cryptographie (typiquement une signature), il faudra que l'objet ait du logiciel pour des opérations cryptographiques typiques, et un magasin de clés cryptographiques. Notez que la sécurité étant assurée via le manifeste, et non pas via le mécanisme de transport (sécurité des données et pas du canal), le moyen de récupération de l'image utilisé n'a pas de conséquences pour la sécurité. Ce sera notamment pratique pour le "*multicast*".

Le manifeste peut être inclus dans l'image ou bien séparé. Dans le second cas, l'objet peut décider de charger l'image ou pas, en fonction du manifeste.

La mise à jour du code d'un objet contraint pose des défis particuliers. Ainsi, ces objets ne peuvent souvent pas utiliser de code indépendant de sa position. La nouvelle image ne peut donc pas être mise n'importe où. Cela veut dire que, lorsque le code est exécuté directement à partir du moyen de stockage (et donc pas chargé en mémoire), il faut échanger l'ancienne et la nouvelle image, ce qui complique un éventuel retour en arrière, en cas de problème.

Avec tout ça, je n'ai pas encore tellement parlé du manifeste lui-même. C'est parce que notre RFC ne décrit que l'architecture, donc les généralités. Le format du manifeste sera dans deux futurs RFC, un pour décrire le format exact (fondé sur CBOR, cf. RFC 8949) et un autre pour le modèle d'information (en gros, la liste des éléments qui peuvent être mis dans le manifeste).

La section 7 de notre RFC fait la synthèse des questions de sécurité liées à la mise à jour. Si une mise à jour fréquente du logiciel est cruciale pour la sécurité, en permettant de fermer rapidement des failles découvertes dans le logiciel de l'objet, la mise à jour peut elle-même présenter des risques. Après tout, mettre à jour son logiciel, c'est exécuter du code récupéré via l'Internet...L'architecture présentée dans ce RFC fournit une sécurité de bout en bout, par exemple en permettant de vérifier l'authenticité de l'image récupérée (et, bien sûr, son intégrité). Pas question d'exécuter un code non authentifié, ce qui est pourtant couramment le cas aujourd'hui avec les mises à jour de beaucoup d'équipements informatiques.

Cette même section en profite pour noter que la cryptographie nécessite de pouvoir changer les algorithmes utilisés. Le RFC cite même l'importance de la cryptographie post-quantique. (C'est un des tous premiers RFC publiés à en parler, après le RFC 8773.) D'un côté, ce genre de préoccupations est assez décalée : la réalité de l'insécurité des objets connectés est tellement abyssale qu'on aura bien des problèmes avant que les calculateurs quantiques ne deviennent une menace réelle. De l'autre, certains objets connectés peuvent rester en service pendant longtemps. Qui sait quelles seront les menaces dans dix ans? (Un article comme « "*Quantum Annealing for Prime Factorization*" `<https://www.nature.com/articles/s41598-018-36058-z>` » estimait que, dans le pire des cas, les algorithmes pré-quantiques commenceraient à être cassés vers 2030, mais il est très difficile d'estimer la

fiabilité de ces prédictions, cf. mon exposé à Pas Sage En Seine <https://www.bortzmeyer.org/pas-sage-en-seine-quantique.html>.)

J'insiste, mais je vous recommande de lire le RFC 8240 pour bien comprendre tous les aspects du problème. D'autre part, vous serez peut-être intéressés par l'évaluation faite du projet SUIT <https://mailarchive.ietf.org/arch/msg/suit/vH6PL5czghj5eLohdZgLysCwElc> sous l'angle des droits humains. Cette évaluation suggérait (ce qui n'a pas été retenu) que le chiffrement des images soit obligatoire.