RFC 8240 : Report from the Internet of Things (IoT) Software Update (IoTSU) Workshop 2016

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 12 septembre 2017

Date de publication du RFC : Septembre 2017

https://www.bortzmeyer.org/8240.html

La mode des objets connectés a mené à l'existence de nombreux « objets », qui sont en fait des ordinateurs (avec leurs problèmes, comme le fait que leur logiciel ait des failles de sécurité), mais qui ne sont pas considérés comme des ordinateurs par leurs propriétaires, et ne sont donc pas gérés (pas d'administrateur système, pas de mises à jour des logiciels). L'attaque contre Dyn du 21 octobre 2016, apparemment menée par ces objets, a bien illustré le risque que cette profusion irresponsable crée. Ce nouveau RFC est le compte-rendu d'un atelier sur la mise à jour des objets connectés <https://www.iab.org/activities/workshops/iotsu/>, atelier de réflexion qui s'est tenu à Dublin les 13 et 14 juin 2016. Il fait le point sur la (difficile) question.

La question de base était « que peut-on faire pour que ces foutus objets soient mis à jour, au moins pour "*patcher*" leurs failles de sécurité ? » L'atelier réuni à Trinity College n'a évidemment pas fourni une réponse parfaite, mais a au moins permis de clarifier le problème. Il me semble toutefois, d'après le compte-rendu, qu'il y avait un gros manque : les droits du propriétaire de l'objet. La plupart des solutions discutées tournaient autour de l'idée de mises à jour systématiques et automatiques du logiciel via les serveurs du vendeur, et les éventuelles conséquences néfastes pour le propriétaire (comme l'arrivée de nouvelles fonctions de surveillance, par exemple) ne sont guère mentionnées.

Aujourd'hui, un grand nombre des machines ayant accès, même indirect, à l'Internet, est composé de trucs qualifiés d'objets, qui ont en commun qu'on ne les appelle pas « ordinateurs » (alors que c'est bien cela qu'ils sont). Ces objets sont très divers, allant d'engins qui ont les capacités matérielles d'un ordinateur (une télévision connectée, par exemple) à des petits machins très contraints (CPU et mémoire limités, batterie à la capacité finie, etc). Le risque que font peser ces objets « irresponsables » (pas gérés, pas supervisés) sur l'Internet est connu depuis longtemps. Le RFC cite l'article de Schneier, « "*The Internet of Things Is Wildly Insecure And Often Unpatchable*" <https://www.schneier.com/essays/archives/2014/01/the_internet_of_thin.html> », qui tirait la sonnette d'alarme en 2014. Il notait que le logiciel de ces objets n'était déjà plus à jour quand l'objet était sorti de sa boite la

première fois (les fabricants d'objets connectés aiment bien utiliser des versions antédiluviennes des bibliothèques). Or, des objets peuvent rester branchés et actifs pendant des années, avec leur logiciel dépassé et jamais mis à jour, plein de failles de sécurité que les craqueurs ne vont pas manquer d'exploiter (cf. le logiciel Mirai).

Vers la même époque, un rapport de la FTC, « "*FTC Report on Internet of Things Urges Companies to Adopt Best Practices to Address Consumer Privacy and Security Risks*" <https://www.ftc.gov/news-events/press-releases/2015/01/ftc-report-internet-things-urges-companies-adopt-b
et un autre du groupe Article 29, « "*Opinion 8/2014 on the on Recent Developments on the Internet of Things*" <http://ec.europa.eu/justice/data-protection/article-29/documentation/opinion-recommen files/2014/wp223_en.pdf> » faisaient les mêmes remarques.

Donc, le problème est connu depuis des années. Mais il n'est pas facile à résoudre :

— Le mécanisme de mise à jour peut lui-même être une faille de sécurité. Par exemple, un certain nombre d'objets font la mise à jour de leur logiciel en HTTP, sans signature sur le code récupéré, rendant ainsi une attaque de l'homme du milieu triviale.
— Le RFC n'en parle guère, mais, même si on vérifie le code récupéré, celui-ci peut être malveillant. De nombreuses compagnies (par exemple Sony) ont déjà délibérement distribué du "*malware*" à leurs clients, et sans conséquences pénales. Une mise à jour automatique pourrait par exemple installer une porte dérobée, ou bien mettre du code DRM nouveau.
— Les problèmes opérationnels ne manquent pas. Par exemple, si les objets vérifient le code récupéré via une signature, et qu'ils comparent à une clé publique qu'ils détiennent, que faire si la clé privée correspondante est perdue ou copiée ? (Cf. « "*Winks Outage Shows Us How Frustrating Smart Homes Could Be*" <http://www.wired.com/2015/04/smart-home-headaches/>. »)
— Si les objets se connectent automatiquement au serveur pour chercher des mises à jour, cela pose un sérieux problème de vie privée. (Le RFC ne rappelle pas qu'en outre, la plupart de ces objets sont de terribles espions <https://www.bleepingcomputer.com/news/technology/roomba-maker-preparing-to-sell-maps-of-your-home-to-advertisers/> qu'on installe chez soi.)
— Pour une société à but lucratif, gérer l'infrastructure de mise à jour (les serveurs, la plate-forme de développement, les mises à jour du code, la gestion des éventuels problèmes) a un coût non nul. Il n'y a aucune motivation financière pour le faire, la sécurité ne rapportant rien. (Le RFC s'arrête à cette constatation, ne mentionnant pas la solution évidente : des règles contraignantes obligeant les entreprises à le faire. Le Dieu Marché ne sera certainement pas une solution.)
— Si l'entreprise originale défaille, qui va pouvoir et devoir assurer ses mises à jour ? (Repensons à l'ex-chouchou des médias, le lapin Nabaztag, et à son abandon par la société qui l'avait vendu.) Si le RFC ne le mentionne pas, il va de soi que le logiciel libre est la solution au problème du « pouvoir » mais pas à celui du « devoir » (dit autrement, le logiciel libre est nécessaire mais pas suffisant).

Le risque n'est donc pas seulement de l'absence de système de mise à jour. Il peut être aussi d'un système de mise à jour bogué, vulnérable, ou non documenté, donc non maintenable.

Le problème de la mise à jour automatique des logiciels est évidemment ancien, et plusieurs systèmes d'exploitation ont des solutions opérationnelles depuis un certain temps (comme pacman ou aptitude). Le RFC se focalise donc sur les objets les plus contraints, ceux limités dans leurs capacités matérielles, et donc dans les logiciels qu'ils peuvent faire tourner.

Après cette introduction, le RFC fait un peu de terminologie, car le choix des mots a suscité une discussion à l'atelier. D'abord, la notion de **classe**. Les « objets connectés » vont de systèmes qui ont les capacités matérielles et l'alimentation électrique d'un ordinateur de bureau (une télévision connectée, par exemple) et qui peuvent donc utiliser les mêmes techniques, jusqu'à des objets bien plus contraints dans leurs capacités. Pour prendre un exemple chez les systèmes populaires auprès des "*geeks*", un Raspberry Pi fait tourner un système d'exploitation « normal » et se met à jour comme un ordinateur

de bureau, un Arduino ne le peut typiquement pas. Il faudra donc sans doute développer des solutions différentes selon la classe. Ou, si on veut classer selon le type de processeur, on peut séparer les objets ayant plus ou moins l'équivalent d'un Cortex-A (comme le Pi) de ceux ayant plutôt les ressources du Cortex-M (comme l'Arduino, mais notez que cette catégorie est elle-même très variée).

Le RFC définit également dans cette section 2 la différence entre mise à jour du logiciel et mise à jour du "*firmware*". La distinction importante est que les engins les plus contraints en ressources n'ont typiquement qu'une mise à jour du "*firmware*", qui change tout, système et applications, alors que les engins de classe « supérieure » ont des mises à jour modulaires (on peut ne mettre à jour qu'une seule application).

Dernier terme à retenir, "*hitless*" (« sans impact »). C'est une propriété des mises à jour qui ne gênent pas le fonctionnement normal. Par exemple, s'il faut arrêter l'engin pour une mise à jour, elle ne sera pas "*hitless*". Évidemment, la mise à jour du logiciel d'une voiture ne sera probablement pas "*hitless*" et nécessitera donc des précautions particulières.

Maintenant, le gros morceau du RFC, la section 3, qui regroupe notamment les exigences issues de l'atelier. C'est assez dans le désordre, et il y a davantage de questions que de réponses. Pour commencer, que devraient faire les fabricants d'objets connectés (en admettant qu'ils lisent les RFC et aient le sens des responsabilités, deux paris hasardeux) ? Par exemple, le RFC note que les mises à jour globales (on remplace tout) sont dangereuses et recommande que des mises à jour partielles soient possibles, notamment pour limiter le débit utilisé sur le réseau (comme avec bsdiff <`http://www.daemonology.net/bsdiff/`> ou courgette <`https://www.chromium.org/developers/design-documents/software-updates-c`> Idéalement, on devrait pouvoir mettre à jour une seule bibliothèque, mais cela nécessiterait du liage dynamique et de code indépendant de la position, ce qui peut être trop demander pour les plus petits objets. Mais au moins un système d'exploitation plutôt conçu pour les « classe M », les objets les plus contraints (équivalents au Cortex-M), le système Contiki, a bien un lieur dynamique.

Certains dispositifs industriels ont plusieurs processeurs, dont un seul gère la connexion avec le reste du monde. Il peut être préférable d'avoir une architecture où ce processeur s'occupe de la mise à jour du logiciel de ses collègues qui ne sont pas directement connectés.

Un problème plus gênant, car non exclusivement technique, est celui des engins qui ont plusieurs responsables potentiels. Si une machine sert à plusieurs fonctions, on risque d'avoir des cas où il n'est pas évident de savoir quel département de l'entreprise a « le dernier mot » en matière de mise à jour. Pour les ordinateurs, cette question sociale a été réglée depuis longtemps dans la plupart des organisations (par exemple en laissant le service informatique seul maître) mais les objets connectés redistribuent les cartes. Si le service Communication a une caméra connectée, qui peut décider de la mettre à jour (ou pas) ?

Un cas proche est celui où un appareil industriel inclut un ordinateur acheté à un tiers (avec le système d'exploitation). Qui doit faire la mise à jour ? Qui est responsable ? Pensons par exemple au problème d'Android où les mises à jour doivent circuler depuis Google vers le constructeur du téléphone, puis (souvent) vers l'opérateur qui l'a vendu. En pratique, on constate qu'Android est mal mis à jour. Et ce sera pire pour un réfrigérateur connecté puisque sa partie informatique sera fabriquée indépendamment, et avant le frigo complet. Lorsque celui-ci démarrera pour la première fois, son logiciel aura sans doute déjà plusieurs mois de retard.

Et la sécurité ? Imaginons un objet connecté dont le fabricant soit assez incompétent et assez irresponsable pour que les mises à jour se fassent en récupérant du code en HTTP, sans aucune authentification

d'aucune sorte. Un tel objet serait très vulnérable à des attaques visant à remplacer le code authentique par un code malveillant. D'où l'exigence de DAO ("*Data Origin Authentication*"). Sans DAO, tout serait fichu. Deux solutions évidentes utilisent la cryptographie, authentifier le serveur qui distribue les mises à jour (par exemple avec HTTPS) ou bien authentifier le code téléchargé, via une signature numérique (notez que l'IETF a un format CMS pour cela, décrit dans le RFC 4108 [1], mais qui semble peu utilisé.) Mais signer et vérifier est plus facile à dire qu'à faire ! D'abord, si la cryptographie n'est pas un problème pour les engins « classe A », comme une télévision connectée ou bien une voiture connectée, elle peut être bien coûteuse pour des objets plus limités. La cryptographie symétrique est typiquement moins coûteuse, mais bien moins pratique. Le problème de fond est que la cryptographie symétrique n'authentifie pas un envoyeur mais une classe d'envoyeurs (tous ceux qui ont la clé). Il faudrait donc une clé différente par objet, ce qui semble ingérable. Faut-il lister comme exigence de base d'utiliser la cryptographie asymétrique ?

Son coût en opérations de calcul n'est pas le seul problème. Par exemple, la clé privée qui signe les mises à jour peut être volée, ou bien tout simplement devenir trop faible avec les progrès de la cryptanalyse (certains objets peuvent rester en production pendans dix ans ou plus). Il faut donc un moyen de mettre la clé publique à jour. Comment le faire sans introduire de nouvelles vulnérabilités (imaginez un attaquant qui trouverait le moyen de subvertir ce mécanisme de remplacement de clé, et changerait la clé de signature pour la sienne). Bref, les méthodes « il n'y a qu'à… » du genre « il n'y a qu'à utiliser des signatures cryptographiques » ne sont pas des solutions magiques.

Et si l'objet utilise du code venant de plusieurs sources ? Faut-il plusieurs clés, et des règles compliquées du genre « l'organisation A est autorisée à mettre à jour les composants 1, 3 et 4 du système, l'organisation B peut mettre à jour 2 et 3 » ? (Les systèmes fermés sont certainement mauvais pour l'utilisateur, mais présentent des avantages en matière de sécurité : une seule source. Pensez à tous les débats autour de l'utilisation du magasin libre F-Droid, par exemple l'opinion de Whisper <https://github.com/WhisperSystems/Signal-Android/issues/281>.)

Un problème plus fondamental est celui de la confiance : à qui un objet connecté doit-il faire confiance, et donc quelle clé publique utiliser pour vérifier les mises à jour ? Faire confiance à son propriétaire ? À son fabricant ? Au sous-traitant du fabricant ?

On n'en a pas fini avec les problèmes, loin de là. Les objets sont conçus pour être fabriqués et distribués en grande quantité. Cela va donc être un difficile problème de s'assurer que tous les objets dont on est responsable soient à jour. Imaginez une usine qui a des centaines, voire des milliers d'objets identiques, et qui veut savoir s'ils sont tous à la même version du logiciel.

On a parlé plus haut de confiance. On peut décider de faire confiance au fabricant d'un objet, parce qu'on juge l'organisation en question honnête et compétente. Mais cette évaluation ne s'étend pas forcément à la totalité de ses employés. Comment s'assurer qu'un employé méchant ne va pas compromettre le processus de mise à jour du logiciel, par exemple en se gardant une copie de la clé privée, ou bien en ajoutant aux objets produits une clé publique supplémentaire (il aurait alors une porte dérobée dans chaque objet, ce qui illustre bien le fait que la mise à jour est elle-même source de vulnérabilités).

On a parlé des risques pour la sécurité de l'objet connecté. Les précautions prises sont actuellement proches de zéro. Espérons que cela va s'améliorer. Mais, même en prenant beaucoup de précautions, des piratages se produiront parfois. Ce qui ouvre la question de la récupération : comment récupérer un objet piraté ? Si on a dix grille-pains compromis, va t-il falloir les reflasher tous les dix, manuellement ?

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc4108.txt

(Avant cela, il y a aussi l'intéressante question de savoir comment détecter qu'il y a eu piratage, sachant que la quasi-totalité des organisations ne lisent pas les messages <https://www.bortzmeyer.org/abuse-ne-repond-pas.html> leur signalant des comportements suspects de leurs machines.)

La cryptographie apporte clairement des solutions intéressantes à bien des problèmes de sécurité. Mais elle nécessite en général une horloge bien à l'heure (par exemple pour vérifier la date d'expiration d'un certificat). Avoir une horloge maintenue par une batterie pendant l'arrêt de la machine et pendant les coupures de courant n'est pas trivial :

— Cela coûte cher (la batterie et l'horloge peuvent coûter bien plus cher que le reste de l'objet),
— C'est encombrant (certains objets sont vraiment petits),
— Les batteries sont des équipements dangereux, en raison du risque d'incendie ou d'explosion, et leur présence impose donc de fortes contraintes au processus de fabrication, d'autant plus que les batteries vieillissent vite (pas question d'utiliser de vieux stocks),
— Les batteries conçues pour la maison peuvent ne pas être adaptées, car, dans un environnement industriel, les conditions sont rudes (la température peut facilement excéder les limites d'une batterie ordinaire),
— Cela ne dure pas longtemps : la présence d'une batterie peut sérieusement raccourcir la durée de vie d'un objet connecté.

Il n'est donc pas évident qu'il y ait une bonne solution à ce problème. (Le RFC cite Roughtime <https://roughtime.googlesource.com/roughtime> comme une approche prometteuse.)

Et si on a une bonne solution pour distribuer les mises à jour à des milliers ou des millions d'objets connectés, comment ceux-ci vont-ils être informés de l'existence d'une mise à jour. "*Push*" ou "*pull*" ? Envoyer les mises à jour depuis le serveur est la solution la plus rapide mais elle peut être délicate si l'objet est derrière un pare-feu qui interdit les connexions entrantes (il faudrait que l'objet contacte le serveur de mise à jour, et reste connecté). Et le cas d'objets qui ne sont pas joignables en permanence complique les choses. De l'autre côté, le cas où l'objet contacte le serveur de temps en temps pour savoir s'il y a une mise à jour pose d'autres problèmes. Par exemple, cela consomme de l'électricité « pour rien ».

Vous trouvez qu'il y a assez de problèmes ? Mais ce n'est pas fini. Imaginez le cas d'une mise à jour qui contienne une bogue, ce qui arrivera inévitablement. Comment revenir en arrière ? Refaire une mise à jour (la bogue risque d'empêcher ce processus) ? L'idéal serait que l'objet stocke deux ou trois versions antérieures de son logiciel localement, pour pouvoir basculer vers ces vieilles versions en cas de problème. Mais cela consomme de l'espace de stockage, qui est très limité pour la plupart des objets connectés.

Le consensus de l'atelier a été que les signatures des logiciels (avec vérification par l'objet), et la possibilité de mises à jour partielles, étaient importants. De même, les participants estimaient essentiels la disponibilité d'une infrastructure de mise à jour, qui puisse fonctionner pour des objets qui n'ont pas un accès complet à l'Internet. Par exemple, il faut qu'une organisation puisse avoir une copie locale du serveur de mise à jour, à la fois pour des raisons de performances, de protection de la vie privée, et pour éviter de donner un accès Internet à ses objets.

En parlant de vie privée (RFC 6973), il faut noter que des objets se connectant à un serveur de mise à jour exposent un certain nombre de choses au serveur, ce qui peut ne pas être souhaitable. Et certains objets sont étroitement associés à un utilisateur (tous les gadgets à la maison), aggravant le problème. Le RFC note qu'au minimum, il ne faut pas envoyer un identificateur unique trop facilement, surtout au dessus d'un lien non chiffré. Mais le RFC note à juste titre qu'il y a un conflit entre la vie privée et le désir des vendeurs de se faire de l'argent avec les données des utilisateurs.

On a surtout parlé jusqu'à présent des risques pour l'objet si le serveur des mises à jour est méchant ou piraté. Certains acteurs du domaine vont ajouter des risques pour le serveur : par exemple, si le

logiciel est non-libre, certains peuvent souhaiter authentifier l'objet, pour ne pas distribuer leur précieux logiciel privateur à « n'importe qui ».

On en a déjà parlé mais cela vaut la peine d'y revenir : un gros problème de fond est celui de l'autorisation (section 4 du RFC). Qui peut accepter ou refuser une mise à jour ? Oui, il y a de bonnes raisons de refuser : des mises à jour obligatoires sont une forme de porte dérobée, elles permettent d'ajouter des fonctions qui n'étaient pas là avant et qui peuvent être malveillantes. Le cas avait été cité lors de l'affaire de l'iPhone de San Bernardino (des gens proposaient une mise à jour que le téléphone appliquerait, introduisant une porte dérobée). Et le RFC rappelle le cas d'une mise à jour forcée d'imprimantes HP qui avait supprimé des fonctions utiles mais que HP ne voulait plus <http://boingboing.net/2016/09/19/hp-detonates-its-timebomb-pri.html> (la possibilité d'accepter des cartouches d'encre fournies par les concurrents). Un cas analogue révélé la veille de la publication du RFC est celui de Tesla activant une nouvelle fonction à distance <https://electrek.co/2017/09/09/tesla-extends-range-vehicles-for-free-in-florida-escape-hurricane-irma/> (ici, pour améliorer la voiture, mais il est facile de voir que cela pourrait être en sens inverse). Et il y a bien sûr le cas fameux d'Amazon détruisant des livres à distance <http://www.nytimes.com/2009/07/18/technology/companies/18amazon.html>.

Prenez un téléphone Android programmé par Google, fabriqué par LG, vendu par Orange à la société Michu qui le confie à son employé M. Dupuis-Morizeau. Qui va décider des mises à jour ?

On peut séparer les mises à jour qui corrigent une faille de sécurité et les autres : seules les premières seraient systématiquement appliquées. (Tous les fournisseurs de logiciel ne font pas cette séparation, qui les oblige à gérer deux lignes de mises à jour.) Ceci dit, en pratique, la distinction n'est pas toujours facile, et la correction d'une faille peut en entrainer d'autres.

D'autre part, certains objets connectés sont utilisés dans un environnement très régulé, où tout, matériel et logiciel, doit subir un processus de validation formel avant d'être déployé (c'est le cas des objets utilisés dans les hôpitaux, par exemple). Il y aura forcément une tension entre « faire la mise à jour en urgence car elle corrige une vulnérabilité » et « ne pas faire la mise à jour avant d'avoir tout revalidé, car cet objet est utilisé pour des fonctions vitales ».

Autre problème rigolo, et qui a de quoi inquiéter, le risque élevé d'une cessation du service de mise à jour logicielle (section 5 du RFC). Les vendeurs de logiciel ne sont pas un service public : ils peuvent décider d'arrêter de fournir un service pour des produits qui ne les intéressent plus (ou qui fait concurrence à un produit plus récent, qui offre une meilleure marge). Ou bien ils peuvent tout simplement faire faillite. Que faire face à cet échec du capitalisme à gérer les produits qu'il met sur le marché ? Le problème est d'autant plus fréquent que des objets connectés peuvent rester en service des années, une éternité pour les directions commerciales. Outre le cas du Nabaztag cité plus haut, le RFC mentionne l'exemple d'Eyefi <http://www.diyphotography.net/eyefi-drop-support-cards-will-magically-stop-work>. Un objet un peu complexe peut incorporer des parties faites par des organisations très différentes, dont certaines seront plus stables que d'autres.

Faut-il envisager un service de reprise de la maintenance, par des organisations spécialisées dans ce ramassage des orphelins ? Si ces organisations sont des entreprises à but lucratif, quel sera leur modèle d'affaires ? (Le RFC ne mentionne pas la possibilité qu'il puisse s'agir d'un service public.) Et est-ce que l'entreprise qui abandonne un produit va accepter ce transfert de responsabilité (aucune loi ne la force à gérer ses déchets logiciels) ?

Pour le logiciel, une grosse différence apparait selon qu'il s'agisse de logiciel libre ou pas. Un projet de logiciel libre peut se casser la figure (par exemple parce que l'unique développeur en a eu marre), mais

il peut, techniquement et juridiquement, être repris par quelqu'un d'autre. Ce n'est qu'une possibilité, pas une certitude, et des tas de projets utiles ont été abandonnés car personne ne les a repris.

Néanmoins, si le logiciel libre n'est pas une condition suffisante pour assurer une maintenance sur le long terme, c'est à mon avis une condition nécessaire. Parfois, un logiciel est libéré lorsqu'une société abandonne un produit (ce fut le cas avec Little Printer <http://littleprinterblog.tumblr.com/post/97047976103/the-future-of-little-printer>, qui fut un succès, la maintenance étant repris par des développeurs individuels), parfois la société ne fait rien, et le logiciel est perdu. Bref, un système de séquestre du code, pour pouvoir être transmis en cas de défaillance de l'entreprise originale, serait une bonne chose.

Comme souvent en sécurité, domaine qui ne supporte pas le « ya ka fo kon », certaines exigences sont contradictoires. Par exemple, on veut sécuriser les mises à jour (pour éviter qu'un pirate ne glisse de fausses mises à jour dans le processus), ce qui impose des contrôles, par exemple par des signatures numériques. Mais on veut aussi assurer la maintenance après la fin de la société originelle. Si un prestataire reprend la maintenance du code, mais qu'il n'a pas la clé privée permettant de signer les mises à jour, les objets refuseront celles-ci...Le séquestre obligatoire de ces clés peut être une solution, mais elle introduit d'autres risques (vol des clés chez le notaire).

Une solution possible serait que les appareils connectés soient programmés pour cesser tout service s'ils n'arrivent pas à faire de mise à jour pendant N mois, ou bien si le serveur des mises à jour leur dit que le contrat de support est terminé. De telles bombes temporelles seraient bonnes pour la sécurité, et pour le business des fabricants. Mais elles changeraient le modèle d'achat d'équipement : on ne serait plus propriétaire de sa voiture ou de son grille-pain, mais locataire temporaire. De la vraie obsolescence programmée ! De telles bombes dormantes peuvent aussi interférer avec d'autres fonctions de sécurité. Par exemple, un téléphone doit pouvoir passer des appels d'urgence, même si l'abonnement a expiré. Il serait logique qu'il soit en mesure d'appeler le 112, même si son logiciel n'a pas pu être mis à jour depuis longtemps (cf. RFC 7406 pour une discussion sur un compromis similaire.)

Dans une logique libertarienne, ce RFC parle peu de la possibilité d'obligations légales (comme il y a un contrôle technique pour les voitures, il pourrait y avoir obligation de mesures de sécurité quand on vend des objets connectés). Comme il n'y a aucune chance que des entreprises capitalistes fassent spontanément des efforts pour améliorer la sécurité de tous, si on n'utilise pas de contraintes légales, il reste la possibilité d'incitations, discutée dans la section 6. Certaines sont utopiques (« il est dans l'intérêt de tous d'améliorer la sécurité, donc il faut expliquer aux gentils capitalistes l'intérêt de la sécurité, et ils le feront gratuitement »). Une possibilité est celle d'un abonnement : ayant acheté un objet connecté, on devrait payer régulièrement pour le maintenir à jour. Du point de vue de la sécurité, c'est absurde : il est dans l'intérêt de tous que **toutes** les machines soient à jour, question sécurité. Avec un abonnement, certains propriétaires choisiront de ne pas payer, et leurs engins seront un danger pour tous. (Comparons avec la santé publiqué : il est dans l'intérêt des riches que les pauvres se soignent, sinon, les maladies contagieuses se répandront et frapperont tout le monde. Les vaccinations gratuites ne sont donc pas de la pure générosité.)

Un des problèmes du soi-disant « Internet des objets » est qu'on ne sait pas grand'chose de ce qui s'y passe. Combien de constructeurs d'objets connectés peuvent dire combien ils ont d'objets encore actifs dans la nature, et quel pourcentage est à jour, question logiciel ? La section 7 du RFC se penche sur les questions de mesures. Un des articles présentés à l'atelier <https://down.dsg.cs.tcd.ie/iotsu/subs/IoTSU_2016_paper_25.pdf> notait que, treize ans après la fin de la distribution d'un objet connecté, on en détectait encore des exemplaires actifs.

Garder trace de ces machines n'est pas trivial, puisque certaines ne seront allumées que de manière très intermittente, et que d'autres seront déployées dans des réseaux fermés. Mais ce serait pourtant

utile, par exemple pour savoir combien de temps encore il faut gérer des mises à jour pour tel modèle, savoir quels problèmes ont été rencontrés sur le terrain, savoir quelles machines ont été compromises, etc.

La solution simple serait que l'objet ait un identifiant unique et contacte son fabricant de temps en temps, en donnant des informations sur lui-même. Évidemment, cela soulève de gros problèmes de préservation de la vie privée, si le fabricant de brosses à dents connait vos habitudes d'hygiène (« sa brosse à dents n'a pas été allumée depuis six mois, envoyons-lui des pubs pour un dentiste »). Il existe des techniques statistiques qui permettent de récolter des données sans tout révéler, comme le système RAPPOR <https://github.com/google/rappor> ou EPID.

Quelle que soit l'efficacité des mises à jour, il ne fait pas de doute que, parfois, des machines seront piratées. Comment les gérer, à partir de là (section 9) ? Car ce ne sera pas seulement un ou deux objets qu'il faudra reformater avec une copie neuve du système d'exploitation. En cas de faille exploitée, ce seront au moins des centaines ou des milliers d'objets qui auront été piratés par un logiciel comme Mirai. (Notez qu'un travail - qui a échoué - avait été fait à l'IETF sur la gestion des machines terminales et leur sécurité, NEA <https://datatracker.ietf.org/wg/nea/charter/>.)

Rien que signaler à l'utilisateur, sans ouvrir une nouvelle voie d'attaque pour le hameçonnage, n'est pas évident (le RFC 6561 propose une solution, mais peu déployée). Bref, pour l'instant, pas de solution à ce problème. Quand on sait que les organisations à but lucratif ne réagissent jamais quand on leur signale un bot dans leur réseau, on imagine ce que ce sera quand on préviendra M. Michu que son grille-pain est piraté et participe à des DoS (comme celles avec SNMP <https://www.bitag.org/documents/SNMP-Reflected-Amplification-DDoS-Attack-Mitigation.pdf> ou bien celle contre Brian Krebs <http://arstechnica.com/security/2016/09/botnet-of-145k-cameras-reportedly-delive>).

Avant d'arriver à une conclusion très partielle, notre RFC, dans sa section 10, traite quelques points divers :

— Pas mal de gadgets connectés seront rapidement abandonnés, ne délivrant plus de service utile et n'intéressant plus leur propriétaire, mais dont certains seront toujours connectés et piratables. Qui s'occupera de ces orphelins dangereux ?
— Que devra faire la machine qui voit qu'elle n'arrive pas à faire ses mises à jour ? Couiner très fort comme le détecteur de fumée dont la batterie s'épuise ? Vous imaginez votre cuisine quand dix appareils hurleront en même temps « *"Software update failed. Act now !"* » ? Ou bien les appareils doivent-ils s'éteindre automatiquement et silencieusement ? (Les fabricants adoreront cette solution : pour éliminer les vieilles machines et forcer les clients à acheter les nouvelles, il suffira de stopper le serveur des mises à jour.)
— La cryptographie post-quantique est pour l'instant inutile (et les mécanismes actuels font des signatures énormes, inadaptées aux objets contraints) mais il est impossible de dire quand elle deviendra indispensable. Cinq ans ? Dix ans ? Jamais ? Le problème est que, les objets connectés restant longtemps sur le terrain, il y a un risque qu'un objet soit livré avec de la cryptographie pré-quantique qui soit cassée par les ordinateurs quantiques pendant sa vie. On sera alors bien embêtés.
— Enfin, pour ceux qui utilisent des certificats pour vérifier les signatures, il faut noter que peu testent les révocations (RFC 6961 et ses copains). Il sera donc difficile, voire impossible, de rattraper le coup si une clé privée de signature est volée.

La section 11 rassemble les conclusions de l'atelier. On notera qu'il y a peu de consensus.

— Comme c'est l'IETF, il n'est pas étonnant que les participants soient tombés d'accord sur l'utilité d'une solution normalisée pour la distribution et la signature de logiciels, afin de permettre des mises à jour fiables.
— Il n'est par contre pas clair de savoir si cette solution doit tout inclure, ou seulement certaines parties du système (la signature, par exemple).

— Des mécanismes de « transmission du pouvoir » sont nécessaires, par exemple pour traiter le cas où une société fait faillite, et passe la maintenance des objets qu'elle a vendu à une association ou un service public. Passer le mot de passe de root est un exemple (très primitif).

L'annexe B du RFC liste tous les papiers présentés à l'atelier qui sont, évidemment en ligne <https://down.dsg.cs.tcd.ie/iotsu/>. Sinon, vous pouvez regarder le site officiel de l'atelier <https://www.iab.org/activities/workshops/iotsu/>, et un bon résumé de présentation <https://www.ietf.org/blog/2016/07/patching-the-internet-of-things-iot-software-update-workshop->. Je recommande aussi cet article de l'IETF Journal <http://www.ietfjournal.org/the-internet-of-things-u>, et un très bon article critique <https://theconversation.com/the-internet-of-things-is-sending-us-k sur les objets connectés.