Internet Architecture Board (IAB)
Request for Comments: 8240
Category: Informational
ISSN: 2070-1721

H. Tschofenig
S. Farrell

September 2017

Report from the Internet of Things Software Update (IoTSU) Workshop 2016

Abstract

This document provides a summary of the Internet of Things Software Update (IoTSU) Workshop that took place at Trinity College Dublin, Ireland on the 13th and 14th of June, 2016. The main goal of the workshop was to foster a discussion on requirements, challenges, and solutions for bringing software and firmware updates to IoT devices. This report summarizes the discussions and lists recommendations to the standards community.

Note that this document is a report on the proceedings of the workshop. The views and positions documented in this report are those of the workshop participants and do not necessarily reflect IAB views and positions.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Architecture Board (IAB) and represents information that the IAB has deemed valuable to provide for permanent record. It represents the consensus of the Internet Architecture Board (IAB). Documents approved for publication by the IAB are not a candidate for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8240.

Table of Contents

1. Introduction

This document provides a summary of the Internet of Things Software Update (IoTSU) Workshop [IoTSU] that took place at Trinity College Dublin, Ireland on the 13th and 14th of June, 2016. The main goal of the workshop was to foster a discussion on requirements, challenges, and solutions for bringing software and firmware updates to IoT devices.

The views and positions in this report are those of the workshop participants and do not necessarily reflect those of their employers/sponsors, the authors of this memo, nor the Internet Architecture Board (IAB), under whose auspices the workshop was held.

The IAB holds occasional workshops designed to consider long-term issues and strategies for the Internet, and to suggest future directions for the Internet architecture. The topics investigated often require coordinated efforts of different organizations and industry bodies to improve an identified problem. One of the goals of such workshops is to assist with communication between relevant organizations, companies, and universities, especially when the topics are partly out of the scope for the Internet Engineering Task Force (IETF). This long-term planning function of the IAB is complementary to the ongoing engineering efforts performed by working groups of the IETF.

In his essay "The Internet of Things Is Wildly Insecure -- And Often Unpatchable" [BS14], Bruce Schneier expressed concerns about the status of software/firmware updates for IoT devices. IoT devices, which have a reputation for being insecure from the time they are manufactured, are often expected to stay active in the field for 10 or more years and to operate unattended with Internet connectivity.

Incorporating a software update mechanism to fix vulnerabilities, to update configuration settings and, to add new functionality as well, is recommended by security experts. However, there are challenges when using software updates, as documented in the United States Federal Trade Commission (FTC) report titled "internet of things: Privacy & Security in a Connected World" [FTC] and in the Article 29 Data Protection Working Party document "Opinion 8/2014 on the on [sic] Recent Developments on the Internet of Things"[WP29].

Among the challenges in designing a basic software/firmware update function are:

- Implementations of software update mechanisms may incorporate vulnerabilities, becoming an attractive attack target. See, for example, [OS14].

- Operational challenges, such as the case of an expired certificate in a hub device [BB14].

- Privacy issues if devices "call home" often to check for updates.

- A lack of incentives to distribute software updates along the value chain.

- Questions such as the following. Who should be able to update device software after normal support stops? When should an alternate source of software updates take over?

There are various (often proprietary) software update mechanisms in use today, and the functionality of those varies significantly with the envisioned use of the IoT devices. More powerful IoT devices, such as those running general purpose operating systems (like Linux), can make use of sophisticated software update mechanisms known from the desktop and the mobile world. This workshop focused on more constrained IoT devices that often run dedicated real-time operating systems or potentially no operating system at all.

There is a real risk that many IoT devices will continue to be shipped without a solid software/firmware update mechanism in place. Ideally, IoT software developers and product designers should be able to integrate standardized mechanisms that have experienced substantial review and where the documentation is available to the public.

Hence, the IAB decided to organize a workshop to reach out to relevant stakeholders to explore the state of the art and to identify requirements and gaps. In particular, the call for position papers asked for:

- Protocol mechanisms for distributing software updates.

- Mechanisms for securing software updates.

- Metadata about software/firmware packages.

- Implications of operating system and hardware design on the software update mechanisms.

- Installation of software updates (in context of software and hardware security of IoT devices).

- Privacy implications of software update mechanisms.

- Implications of device ownership and control for software update.

The rest of the document is organized as follows: basic terminology is provided in Section 2, followed by a longer section discussing requirements. Subsequent sections explore selected topics, such as incentives and measurements in more detail. Most of the write-up does raise more questions than it answers. Nevertheless, we tried to synthesize possible conclusions and offer a few next steps.

2. Terminology

As is typical with people from different backgrounds, workshop participants started the workshop with a discussions of terminology. This section is more intended to reflect those discussions than to present canonical definitions of terms.

Device Classes: IoT devices come in various "sizes" (such as size of RAM or size of flash memory). With these configurations, devices are limited in what they can support in terms of operating-system features, cryptographic algorithms, and protocol stacks. For this reason, the group differentiated two types of classes, namely ARM Cortex A-class/Intel Atom and Cortex M-class/Intel Quark types of devices. A-class devices are equipped with powerful processors typically found in set-top boxes and home routers. The Raspberry Pi is an example of an A-class device that is capable of running a regular desktop operating system, such as Linux. There are differences between the Intel and the ARM-based CPUs in terms of architecture, microcode, and who is allowed to update a Basic Input/Output System (BIOS) (if available). A detailed discussion of these hardware architectural differences were, however, outside the scope of the workshop. The implication is that lower-end microcontrollers have constraints that put restrictions on the amount of software that can be put on them. While it is easy to require support of a wide range of features, those may not necessarily fit on these devices.

Software Update and Firmware Update: Based on the device classes, it was observed that regular operating systems come with sophisticated software update mechanisms (such as Red Hat Package Manager (RPM) [RPM] or pacman [PACMAN]) that make use of the operating system to install and run each application in a compartmentalized fashion. Firmware updates typically do not provide such a fine-grained granularity for software updates and instead distribute the entire binary image, which consists of the (often minimalistic) operating system and all applications. While the distinction between the mechanisms that A-class and M-class devices will typically use may get more fuzzy over time, most M-class devices use firmware updates while A-class devices use a combination of firmware and software updates (with firmware updates being less frequent operations).

Hitless Update: A hitless update implies that the user experience is not "hit", i.e., it is not impacted. It is possible to impact the user experience when applying an update even when the device does not reboot (to obtain or apply said update). If the update is applied when a user is not using a product and their service is not impacted, the update is "hitless".

3. Requirements and Questions Raised

Workshop participants discussed requirements and several of these raised further questions. As with the previous section, we aim to present the discussion as it was.

- There may be a need to be support partial (differential) updates that do not require the entire firmware image to be sent. This may mean that techniques like bsdiff [BSDIFF] and courgette [COURGETTE] are used but might also mean devices supporting the download of applications and libraries alone. The latter feature may require dynamic linking and position independent code. It was unclear whether position independent code should be recommended for low-end IoT devices.

- The relative importance of dynamic linkers for low-end IoT devices is unclear. Some operating systems used with M-class devices, such as Contiki, provide support for a dynamic linker according to [OS-Support]. This could help to minimize the amount of data transmitted during updates since only the modified application or library needs to be transmitted.

- How should dependencies among various software updates be handled? These dependencies may include information about the hardware platform and configuration as well as other software components running on a system. For firmware updates, the problem of dependencies are often solved by the manufacturer or Original Equipment Manufacturer (OEM) rather than on the device itself.

- Support for devices with multiple microcontrollers may require an architecture where one microcontroller is responsible for interacting with the update service and then dispatching software images to the attached microcontrollers within its local realm. The alternative of letting each microcontroller interact with an update service appeared less practical.

- Support may be required for devices with multiple owners/ stakeholders where the question arises about who is authorized to push a firmware/software update.

- Data origin authentication (DAO) was agreed to be required for software updates. Without DAO, updates simply become a perfect vulnerability. It is, however, nontrivial to ensure that the actual trust relationships that exist are modeled by the DAO mechanism. For some devices and deployment scenarios, any DAO mechanism is onerous, possibly to the point where it may be hard to convince a device maker to include the functionality.

- Should digital signatures and encryption for software updates be recommended as a best current practice? This question particularly raises the question about the use of symmetric key cryptography since not all low-end IoT devices are currently using asymmetric crypto.

- DAO is most commonly provided via digital signature mechanisms, but symmetric schemes could also be developed, though IETF discussion of such mechanisms (for purposes less sensitive than software update) has proved significantly controversial. The main problem seems to be that simple symmetric schemes only ensure that the sender is a member of a group, and they do not fully authenticate a specific sender. And with a software update, we do not want any (possibly compromised) device to be able to authenticate new software for all other similar devices.

- What are the firmware update signing key requirements? Since devices have a rather long lifetime, there has to be a way to change the signing key during the lifetime of the device.

- Should a firmware update mechanism support multiple signatures of firmware images? Multiple signatures can come in two different flavors, namely:

  A single firmware image may be signed by multiple different parties. In this case, one could imagine an environment where an OEM signs the software it creates, but then the software is again signed by the enterprise that approves the distribution within the company. Other examples include regulatory signatures where the software for a medical device may be signed as approved by a certification body.

  A software image may contain libraries that are each signed by their developers.

  Is a device expected to verify the different types of signatures or is this a service provided by some unconstrained device? This raises questions about who the IoT device should trust for what and whether transitive trust is acceptable for some types of devices.

- Are applications from a range of sources allowed to run on a device or only those from the OEM? If the device is a "closed" device that only supports/runs software from the OEM, then a single signature may be sufficient. In a system that is more "open", third-party applications may require support of multiple signatures.

- There is a need for some form of secure storage, at least for those IoT devices that are exposed to physical attacks. This includes at least the need to protect the integrity of the public key of the update service on the device (if signature-based DAO is in use). The use of symmetric key cryptography requires improved confidentiality protection (in addition to integrity protection).

- Is there a need to allow the update infrastructure side to authenticate the IoT device before distributing an update? Questions about the identifier used for such an authentication action were raised. The idea of reusing Media Access Control (MAC) addresses lead to concerns about the significant privacy implications of such identifier reuse.

- It is important to minimize device/service downtime due to update processing and to minimize user interaction (e.g., car should not distract the driver) (see "Hitless Update" in Section 2). While it may not be possible to avoid all downtime, there was agreement that one ought to strive for "no inappropriate" device downtime. This means minimal downtime impacting the user/operation of the device. The definition of "downtime" also depends on the use case, with a smart light bulb, the device could be "up" if the light is still on, even if some advanced services are unavailable for a short time. Whether an update can be done without rebooting the device depends on the software being installed, on the OS architecture, and potentially even on the hardware architecture. The cost/benefit ratio also plays a role.

- It is desirable to minimize the time taken from the start of the update to when it is finished. In some systems with many devices (e.g., industrial lighting), this can be a challenge if updates need to be unicasted.

- In some systems with multiple devices, it can be a challenge to ensure that all devices are at the same release level, especially if some devices are sleepy. There are some systems where ensuring all relevant devices are at the same release level is a hard requirement. In other cases, it is acceptable if devices converge much more slowly to the current release level.

- It ought not be possible for a factory worker to compromise the update process (e.g., copy signing keys and install unauthorized public keys/trust anchors) during the manufacturing process. There are typically two factories involved: the first factory produces microcontrollers and other components and the second factory produces the complete product, such as a fridge. This fridge contains many of the components previously manufactured. Hence, the firmware of components produced in the first stage may be six months old when the fridge leaves the factory. One does not want to install a firmware update when the fridge boots the first time. For that time, the firmware update happens already at the end of the manufacturing process.

- Should devices have a recovery procedure when the device gets compromised? How is the compromise detected?

- There was a bit of discussion about the importance for IoT devices to know the current time for the purpose of checking certificate validity. For example, what does "real-time clock" (RTC) actually mean? And what constitutes "good enough" time? There are, however, cost, power, size, and environmental constraints that can make the addition of a real-time clock to an IoT device complex:

  - Cost: Battery- or supercap-backed RTC modules might be several times the cost of the rest of the bill of materials.

  - Size: The battery and other components are often several times larger than the rest of the material.

  - Manufacturing: Some modules require an extra assembly step, because the battery could be damaged or explode at high temperatures during the reflow process.

  - Supply chain: Devices containing fitted batteries need additional supply-chain management to account for storage temperature and to avoid shipping aged devices.

  - Environmental: Real-time-clock modules are typically not rated at industrial temperature ranges. Those that are have extremely reduced lifetime at high temperatures.

  - Lifetime: Some of these modules last only a few years at the top of their environmental range.

  While a good solution is needed, it is not clear whether there is one true solution. A recent proposal from Google called "Roughtime" [RT] may be worthwhile to explore.

- How do devices learn about a firmware update? Push or Pull? What should be required functionality for a firmware update protocol?

- There is a need to find out whether a software update was successful. In one discussed solution, the bootloader analyzes the performance of the running image to determine which image to run (rather than just verifying the integrity of the received image). One of the key criteria is that the updated system is able to make a connection to the device management/software update infrastructure. As long as it is able to talk to the update infrastructure, it can receive another update. As an alternative perspective, the argument was made that one needs to have a way to update the system without having the full system running.

- Gateway requirements. In some deployments, gateways terminate the IP-based protocol communication and use non-IP mechanisms to communicate with other microcontrollers, for example, within a car. The gateway in such a system is the endpoint of the IP communication. The group had mixed feelings about the use of gateways versus the use of IP communication to every microcontroller. Participants argued that there is a lack of awareness of IPv6 header compression (with the IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) standards) and of the possible benefits of IPv6 in those environments in terms of lowering the complexity of the overall system.

- The amount of energy consumed due to software update needs to be minimized. For example, awakening a sleepy device regularly only to check for new software would seem wasteful if the device cannot feasibly be exploited while asleep. However, the trade-off is that once the device awakens with old software, there may be a window of vulnerability if some relevant exploit has been discovered.

- The amount of storage required for update ought to be minimized and can sometimes be significant. However, there are also benefits to schemes that store two or three different software images for robustness, e.g., if one has space for separate current last-known-good and being-updated images, then devices can better survive the buggy occasional updates that are also inevitable.

Which of the features discussed in the list above are nice to have? Which are required? Not all of these are required to achieve improvement. Which are most important?

Among the participants, there was consensus that supporting signatures (for integrity and authentication) of the firmware image itself and the need for partial updates were seen as important.

However, there were also concerns regarding the performance implications, since certain device categories may not utilize public key cryptography at all; hence, only a symmetric key approach seems viable, unless some other scheme such as a hash-based signature became practical (they currently aren't, due to signature size). This aspect raised concerns and triggered a discussion around the use of device management infrastructure, similar to Kerberos, that manages keys and distributes them to the appropriate parties. As such, in this setup, there could be a unique key shared with the key distribution center; but for use with specific services (such as a software update service), a fresh and unique secret would be distributed.

In addition to the requirements for the end devices, there are also infrastructure-related requirements. The infrastructure may consist of servers in the local network and/or various servers deployed on the Internet. It may also consist of some application-layer gateways. The potential benefits of having such a local server might include:

- The local server acting for neighboring nodes. For example, in a vehicle one microcontroller can process all firmware updates and redistribute the relevant parts of those to interconnected microcontrollers.

- Local infrastructure could perform some digital signature checks on behalf of the devices, e.g., certificate-revocation checking.

- Local multicast can enable transmission of the same update to many devices.

- Local servers can hide complexity associated with Network Address Translation (NAT) and firewalls from the device.

Another point related to local infrastructure is that since many IoT devices will not be (directly) connected to the Internet, but only through a gateway, there may in any case be a need to develop a software/firmware update mechanism that works in environments where no end-to-end Internet connectivity exists.

Some current firmware update schemes need to identify devices. Different design approaches are possible.

- In an extreme form in one case, the decision about updating a device is made by the infrastructure based on the unique device identification. The operator of the firmware update infrastructure knows about the hardware and software requirements for the IoT devices, knows about the policy for updating the

device, etc. The device itself is provisioned with credentials so that it can verify a firmware update coming from an authorized device.

- In another extreme, the device has knowledge about the software and hardware configuration and possible dependencies. It consults software repositories to obtain those software packages that are most appropriate. Verifying the authenticity of the software packages/firmware images will still be required.

Hence, in some deployed software update mechanisms there is no desire for the device to be identified beyond the need to exchange information about the most recent software versions. For other devices, it is seen as important to identify the device itself in order to provide the appropriate firmware image/software packages.

Related to device identification, various privacy concerns arise, such as the need to determine what information is provided to whom and the uses to which this information is put. For IoT devices where there is a close relationship to an individual (see [RFC6973]), privacy concerns are likely higher than for devices where such a relationship does not exist (e.g., a sensor measuring concrete). The software/firmware update mechanism should, however, not make the privacy situation of IoT devices worse. The proposal from the group was to introduce a minimal requirement of not sending any new identifiers over an unencrypted channel as part of an update protocol.

However, software updates will provide yet another venue in which the tension between those advocating better privacy and those seeking to monetize information will play out. It is in the nature of software update that it requires devices to sometimes "call home" and such interactions provide fertile ground for monetization.

4. Authorizing a Software/Firmware Update

There were quite a few points revolving around authorization:

- Who can accept or reject an update? Is it the owner of the device, the user, or both? The user may not necessarily be the owner.

- With products that fall under a regulatory structure, such as healthcare, you don’t want firmware other than what has been accredited.

- In some cases, it will be very difficult for a firmware update system to communicate to users that an update is available. Doing so may require tracking the device and its status with regard to the installed firmware/software, with all the privacy downsides if such tracking is badly done.

- Not all updates are the same. Security updates are often treated differently compared to feature updates, and the authorization for these may differ.

- Some people may choose to decline updates, often on the basis that their system is currently stable, but also possibly due to concerns about unwanted changes, such as the HP printer firmware update pushed in March 2016 [HP-Firmware] that turned off features that end users liked.

5. End-of-Support

There was quite a bit of discussion about end-of-support for products/devices and how to handle that.

- How should end-of-support or end-of-features be treated? Devices are often deployed for 10+ years (or even longer in some verticals). Device makers may not want or be able to support software and services for such an extended period of time. Will these devices stop working after a certain, previously unannounced period of time, such as Eye-Fi cards [EYEFI]?

- There will be a broad range of device makers involved in IoT, who may differ substantially in terms of how well they can handle the full device life cycle. Some will be large commercial enterprises that are used to dealing with long device lifetimes, whereas others may be very small commercial entities where the device lifetime may be longer than the company lifetime. Yet other devices may be the result of open-source activities that prosper or flounder. The problem of end-of-support arises in all these cases, though feasible solutions for software update may substantially differ. In some cases, device makers may not be willing to continue to update devices, for example, due to a change in business strategies caused by a merger. In yet other cases, a company may have gone bankrupt.

- While there are many legal, ethical, and business-related questions, can we technically enable transfer of device service to another provider? Could there even be business models for entities that take over device updates for original device makers that no longer wish to handle software update?

- The release of code, as it was done with the Little Printer manufactured and developed by a company called "Berg" [LittlePrinter], could provide a useful example. While the community took over the support in that case, this can hardly be assumed in all cases. Just releasing the source code for a device will not necessarily motivate others to work on the code, to fix bugs, or to maintain a service. Nevertheless, escrowing code so that the community can take it over if a company fails is one possible option.

- The situation gets more complex when the device has security mechanisms to ensure that only selected parties are allowed to update the device (which is really a basic requirement for any secure software update). In this case, private signing keys (or similar) may need to be made available as well, which could introduce security problems for already-deployed software. In the best case, it changes assumptions made about the trust model and about who can submit updates.

- How should deployed devices behave when they are end-of-support and support ends? Many of them may still function normally, but others may fail due to the absence of cloud infrastructure services. Some products are probably expected to fail safely, similarly to a smoke alarm that makes a loud noise when the battery becomes empty. Cell phones without a contract can, in some countries, still be used for emergency services (although at the expense of society due to untraceable hoax calls), as discussed in RFC 7406 [RFC7406].

The recommendation that can be provided to device makers and users is to think about the end-of-support and end-of-support scenarios ahead of time and plan for those. While device makers rarely want to consider what happens if their business fails, it is definitely legitimate to consider scenarios where they are hugely successful and want to evolve a product line instead of supporting previously sold products forever. Maybe there is also value in subscription-based models where product and device support is only provided as long as the subscription is paid. Without a subscription, the product is deactivated and cannot pose a threat to the Internet at large.

6. Incentives

Workshop participants also discussed how to create incentives for companies to ship software updates, which is particularly important for products that will be deployed in the market for a long time. It is also further complicated by complex value chains.

- Companies shipping software updates benefit from improved security. Their devices are less likely to be abused as a vector to launch other attacks, whether on their own networks or (as part of a botnet) on other Internet hosts. This clearly creates an incentive to support and use software updates.

- On the other hand, updates can also break things. The negative customer experience can be due to service interruptions during or after the update process but can also result from bad experience from deliberate changes introduced as part of an update -- such as a feature that is not available anymore, or a "bug" that another service has relied upon being fixed.

- For most classes of device, there does not seem to be a regulatory requirement to report or fix vulnerabilities, similar to data-breach-notification laws.

- Subscription models for device management were suggested so that companies providing the service have an economic interest in keeping devices online (and updated for that).

7. Measurements and Analysis

From a security point of view, it is important to know what devices are out there and what version of software they run. One workshop paper [PLONKA] reported measurements that were initially done on buggy devices first distributed in 2003, and that were still detectable in significant numbers just before the workshop 13 years later. As such, in addition to the firmware update mechanism, companies have been offering device management solutions that allow OEMs to keep track of their devices. Tracking these devices and their status is still challenging since some devices are only connected irregularly or are only turned on when needed (such as a hockey alarm that is only turned on before a match).

Various stakeholders have a justified interest in knowing something about deployed devices. For example:

- Manufacturers and other players in the supply chain are interested to know what devices are out there, how many have been sold, and what devices are out there but have not been sold. This could help to understand which firmware versions to support and for how long.

- Device users, owners, and customers like these may want to know what devices are installed over a longer period of time, what software/firmware version is the device running, what is the uptime of each of these devices, what types of faults have occurred, etc. Forgotten devices may pose problems, particularly if they (have the potential to) behave badly.

- To an extent, network operators offering services to device owners and other actors may also need similar information, for example, to control botnets.

- Researchers doing analysis on the state of the Internet ecosystem (such as what protocols are being used, how much data IoT devices generate, etc.,) need measurements for their work.

There can easily be some invasiveness in approaches to acquiring such measurements. The challenge was put forward to find ways to create measurement infrastructures that are privacy preserving. Arnar Birgisson noted that there are privacy-preserving statistical techniques, such as RAPPOR [RAPPOR], and Ned Smith added that techniques like Intel's Enhanced Privacy ID (EPID) may play a role in maintaining some level of anonymity for the IoT device (owners) while also enabling measurement. It seemed clear that naive approaches to measurement (e.g., where devices are willing to expose a unique identifier to anyone on request) are unlikely to prove sufficient.

8. Firmware Distribution in Mesh Networks

There was some discussion of the requirements for mesh-based networks, mainly relating to industrial lighting. In these networks, software update can impose unacceptable performance burdens, especially if there are many devices, some of which may be sleepy.

The workshop discussed whether some forms of multicast (perhaps not IP multicast) would be needed to provide acceptable solutions for software update in such cases. It was not clear at which layer a multicast solution might be effective in such cases, though there did not seem to be any clearly applicable standards-based approach that was available at the time of the workshop.

9. Compromised Devices

There was recognition that there are, and perhaps always will be, large numbers of devices that can be, or have been, compromised. While updating these can mitigate problems, there will always be new devices added to networks that cannot be updated (for various reasons); so the question of what, if anything, to do about compromised devices was discussed.

- There may be value if it were possible to single out a device that shows faulty behavior or has been compromised, and to shut it down in some sense.

- Prior work in the IETF on Network Endpoint Assessment (NEA) [NEA] allowed assessing the "posture" of devices. Posture refers to the hardware or software configuration of a device and may include knowledge that the software installed is up to date. The obtained information can then be used by some network infrastructure to create a quarantined region network around the device.

- RFC 6561 [RFC6561] describes one scheme for an ISP to send "signals" to customers about hosts (usually those that are part of a botnet or generating spam) in their home network.

- Neither RFC 6561 nor NEA has found widespread deployment. Whether such mechanisms can be more successful in the IoT environment has yet to be studied.

The conclusion of the discussion at the workshop itself was that there is some interest in identifying and stopping misbehaving devices, but the actual solution mechanisms are unclear.

10. Miscellaneous Points

There were a number of points discussed at the workshop that don’t neatly fit under the above headings but that are worth recording. Those include:

- Complex questions can arise when considering the impact of the lack of updates on other devices, other persons, or the public in general. If I don’t update my device, and it is used to attack a random host on the Internet, but at no apparent cost to me, then what incentive do I have to do updates that would have protected that random host? What incentive has my device’s vendor to have provided those updates in advance? An example of such a case can be found in DDoS attacks from IoT devices, such as printers [SNMP-DDOS] and cameras [DDOS-KREBS].

- With some IoT devices, there are many stakeholders contributing to the end product (e.g., contributing different subsystems). Ensuring that vulnerabilities are fixed and software/firmware updates are communicated through the value chain is known to be difficult, as demonstrated in [OS14].

- What about forgotten devices? There are many such, and there will be more. Even though they are forgotten, such devices may be useless consumers of electricity, or they may be part of some critical system.

- Can we determine whether an update impacts other devices in the Internet? Updates to one device can have unintended impact on other devices that depend on it. This can have cascading effects if we are not careful. Changing the format of the output of a sensor could have cascading impacts, e.g., if some actuator reacts to the presence/absence of that sensor's data.

- How should a device behave when it is running out-of-date software? The example of a smoke alarm was mentioned. We don't want 100 devices in a living room to start beeping when their batteries run low or when they cannot communicate with the cloud. But are devices supposed to simply stop working?

- The IETF has published a specification that uses the Cryptographic Message Syntax (CMS) to protect firmware packages, as described in RFC 4108 [RFC4108], which also contains metadata to describe the firmware image itself. During the workshop, the question was raised whether a solution will, in the future, be needed that is post-quantum secure. A post-quantum cryptosystem is a system that is secure against quantum computers that have more than a trivial number of quantum bits. It is open to conjecture whether it is feasible to build such a machine, but current signature algorithms are known not to be post-quantum secure. This would require introducing technologies like the Hash-based Merkle Tree Signature (MTS) [HOUSLEY], which was presented and discussed at the workshop. The downsides of such solutions are their novelty and, for these use cases, the fairly large signature or key sizes involved; e.g., depending on the parameters, a signature could easily have a size of 5-10 KiB [HASHSIG] [XMSS]. While it is likely that post-quantum secure signature algorithms will be needed for software updates at some point in time, it may be the case that such algorithms will be needed sooner for services requiring long-term confidentiality, (e.g., using Transport Layer Security (TLS)), so it was not clear that this application would be a first-mover in terms of post-quantum security.

- Many devices that use certificates do not check the revocation status of certificates, even though extensions like Online Certificate Status Protocol (OCSP) stapling exists [RFC6961] and is increasingly deployed with Web browsers. The workshop participants did not reach a conclusion regarding the recommendations of certificate revocation checking, although the importance has been recognized. The reluctance regarding deploying certificate revocation deserves further investigation.

11. Tentative Conclusions and Next Steps

The workshop participants discussed some tentative conclusions and possible next steps:

- There was strong agreement that having some standardized secure (authorized and authenticated) software update would be an improvement over having none.

- It would be valuable to find agreement on the right scope for a standardized software/firmware update mechanism. It is not clear that an entire update system can or should be standardized, but there may be some aspects of such solutions where standards would be beneficial, e.g., (meta-)data formats and/or protocols for distributing firmware updates. More discussion is needed to identify which parts of the problem space could benefit from standardization.

- It will be useful to investigate solutions to install updates with no operational interruption as well as ways to distribute software updates without disrupting network operations (specifically, in low-power wireless networks), including the development of a multicast transfer mechanism (with appropriate security).

- There will almost certainly be a need for a way to transfer authority/responsibility for updates, particularly considering end-of-support cases. This is very close to calling for a standard way to "root" devices as a feature of all devices.

- We would benefit from documentation of proofs-of-concept of software/firmware updates for constrained devices on different operating system architectures. The IETF Light-Weight Implementation Guidance (lwig) Working Group may be a good venue for such documents.

12. Security Considerations

This document summarizes an IAB workshop on software/firmware updates and the entire content is, therefore, security related. Standardizing and deploying a software/firmware update mechanism for use with IoT devices could help fix security vulnerabilities faster and, in some cases, be the only via to get vulnerability patched at all.

13. IANA Considerations

This document does not require any IANA actions.

14. Informative References

[BB14] Barrett, B., "Winks Outage Shows Us How Frustrating Smart Homes Could Be", April 2014, <http://www.wired.com/2015/04/smart-home-headaches/>.

[BS14] Schneier, B., "The Internet of Things Is Wildly Insecure -- And Often Unpatchable", January 2014, <https://www.schneier.com/essays/archives/2014/01/the_internet_of_thin.html>.

[BSDIFF] Percival, C., "Matching with Mismatches and Assorted Applications", September 2016, <https://ora.ox.ac.uk/objects/uuid:4f0d53cc-fb9f-4246-a835-3c8734eba735/datastreams/THESIS01>.

[COURGETTE] Google, "Software Updates: Courgette", September 2016, <https://www.chromium.org/developers/design-documents/software-updates-courgette>.

[DDOS-KREBS] Goodin, D., "Record-breaking DDoS reportedly delivered by >145k hacked cameras", September 2016, <http://arstechnica.com/security/2016/09/botnet-of-145k-cameras-reportedly-deliver-internets-biggest-ddos-ever/>.

[EYEFI] Aldred, J., "Eye-Fi to Drop Suport for Some Cards. They Will 'Magically' Stop Working on September 16, 2016", June 2016, <http://www.diyphotography.net/eyefi-drop-support-cards-will-magically-stop-working-september-16-2016/>.

[FTC] Federal Trade Commission, "FTC Report on Internet of Things Urges Companies to Adopt Best Practices to Address Consumer Privacy and Security Risks", January 2015, <https://www.ftc.gov/system/files/documents/reports/federal-trade-commission-staff-report-november-2013-workshop-entitled-internet-things-privacy/150127iotrpt.pdf>.

[HASHSIG] Langley, A., "Hash based signatures", July 2013, <https://www.imperialviolet.org/2013/07/18/hashsig.html>.

[HOUSLEY] Housley, R., "Use of the Hash-based Merkle Tree Signature (MTS) Algorithm in the Cryptographic Message Syntax (CMS)", Work in Progress, draft-housley-cms-mts-hash-sig-07, June 2017.

[HP-Firmware] BoingBoing, "HP detonates its timebomb: printers stop accepting third party ink en masse", September 2016, <http://boingboing.net/2016/09/19/hp-detonates-its-timebomb-pri.html>.

[IoTSU] IAB, "Internet of Things Software Update Workshop (IoTSU) 2016", June 2016, <https://www.iab.org/activities/workshops/iotsu/>.

[LittlePrinter] Berg, "The future of Little Printer", September 2014, <http://littleprinterblog.tumblr.com/post/97047976103/the-future-of-little-printer>.

[NEA] IETF, "Network Endpoint Assessment (nea) Concluded WG", October 2016, <https://datatracker.ietf.org/wg/nea/charter/>.

[OS-Support] Dong, W., Chen, C., Liu, X., and J. Bu, "Providing OS Support for Wireless Sensor Networks: Challenges and Approaches", DOI 10.1109/SURV.2010.032610.00045, May 2010, <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5462978>.

[OS14] Oppenheim, L. and S. Tal, "Too Many Cooks: Exploiting the Internet of TR-069 Things", December 2014, <http://mis.fortunecook.ie/too-many-cooks-exploiting-tr069_tal-oppenheim_31c3.pdf>.

[PACMAN] "pacman", 2016, <https://www.archlinux.org/pacman/>.

[PLONKA] Plonka, D. and E. Boschi, "The Internet of Things Old and Unmanaged", June 2016, <https://down.dsg.cs.tcd.ie/iotsu/subs/IoTSU_2016_paper_25.pdf>.

[RAPPOR] Erlingsson, U., Pihur, V., and A. Korolova, "RAPPOR", DOI 10.1145/2660267.2660348, July 2014, <http://dl.acm.org/citation.cfm?doid=2660267.2660348>.

[RFC4108] Housley, R., "Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages", RFC 4108, DOI 10.17487/RFC4108, August 2005, <https://www.rfc-editor.org/info/rfc4108>.

[RFC6561] Livingood, J., Mody, N., and M. O'Reirdan, "Recommendations for the Remediation of Bots in ISP Networks", RFC 6561, DOI 10.17487/RFC6561, March 2012, <https://www.rfc-editor.org/info/rfc6561>.

[RFC6961] Pettersen, Y., "The Transport Layer Security (TLS) Multiple Certificate Status Request Extension", RFC 6961, DOI 10.17487/RFC6961, June 2013, <https://www.rfc-editor.org/info/rfc6961>.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <https://www.rfc-editor.org/info/rfc6973>.

[RFC7406] Schulzrinne, H., McCann, S., Bajko, G., Tschofenig, H., and D. Kroeselberg, "Extensions to the Emergency Services Architecture for Dealing With Unauthenticated and Unauthorized Devices", RFC 7406, DOI 10.17487/RFC7406, December 2014, <https://www.rfc-editor.org/info/rfc7406>.

[RPM] "Red Hat Package Manager (RPM)", 2016, <http://rpm.org/>.

[RT] Google, "Roughtime", September 2016, <https://roughtime.googlesource.com/roughtime>.

[SNMP-DDOS] BITAG, "SNMP Reflected Amplification DDoS Attack Mitigation", August 2012, <https://www.bitag.org/documents/SNMP-Reflected-Amplification-DDoS-Attack-Mitigation.pdf>.

[WP29] Article 29 Data Protection Working Party, "Opinion 8/2014 on the on Recent Developments on the Internet of Things", 14/EN, WP 223, September 2014, <http://ec.europa.eu/justice/data-protection/article-29/documentation/opinion-recommendation/files/2014/wp223_en.pdf>.

[XMSS] Huelsing, A., Butin, D., Gazdag, S., Rijneveld, J., and A. Mohaisen, "XMSS: Extended Hash-Based Signatures", Work in Progress, draft-irtf-cfrg-xmss-hash-based-signatures-10, July 2017.

Appendix A. Program Committee

The following individuals helped to organize the workshop: Jari Arkko, Arnar Birgisson, Carsten Bormann, Stephen Farrell, Russ Housley, Ned Smith, Robert Sparks, and Hannes Tschofenig.

Appendix B. Accepted Position Papers

The list of accepted position papers is below. Links to these, and to the workshop agenda and raw minutes are accessible at: <https://down.dsg.cs.tcd.ie/iotsu/>.

- R. Housley, "Position Paper for Internet of Things Software Update Workshop (IoTSU)"

- D. Thomas and A. Beresford, "Incentivising software updates"

- L. Zappaterra and E. Dijk, "Software Updates for Wireless Connected Lighting Systems: requirements, challenges and recommendations"

- M. Orehek and A. Zugenmaier, "Updates in IoT are more than just one iota"

- D. Plonka and E. Boschi, "The Internet of Things Old and Unmanaged"

- D. Bosschaert, "Using OSGi for an extensible, updatable and future proof IoT"

- A. Padilla, E. Baccelli, T. Eichinger, and K. Schleiser, "The Future of IoT Software Must be Updated"

- T. Hardie, "Software Update in a multi-system Internet of Things"

- R. Sparks and B. Campbell, "Avoiding the Obsolete-Thing Event Horizon"

- J. Karkov, "SW update for Long lived products"

- S. Farrell, "Some Software Update Requirements"

- S. Chakrabarti, "Internet Of Things Software Update Challenges: Ownership, Software Security & Services"

- M. Kovatsch, A. Scholz, and J. Hund, "Why Software Updates Are More Than a Security Issue: Challenges for IETF CoRE and the W3C Web of Things"

- A. Grau, "Secure Software Updates for IoT Devices"

- Birr-Pixton, "Electric Imp’s experiences of upgrading half a million embedded devices"

- Y. Zhang, J. Yin, C. Groves, and M. Patel, "oneM2M device management and software/firmware update"

- E. Smith, M. Stitt, R. Ensink, and K. Jager, "User Experience (UX) Centric IoT Software Update"

- J.-P. Fassino, E.A. Moktad, and J.-M. Brun, "Secure Firmware Update in Schneider Electric IOT-enabled offers"

- M. Orehek, "Summary of existing firmware update strategies for deeply embedded systems"

- N. Smith, "Toward A Common Modeling Standard for Software Update and IoT Objects"

- S. Schmidt, M. Tausig, M. Hudler, and G. Simhandl, "Secure Firmware Update Over the Air in the Internet of Things Focusing on Flexibility and Feasibility"

- A. Adomnicai, J. Fournier, L. Masson, and A. Tria, "How careful should we be when implementing cryptography for software update mechanisms in the IoT?"

- V. Prevelakis and H. Hamad, "Controlling Change via Policy Contracts"

- H. Birkholz, N. Cam-Winget, and C. Bormann, "IoT Software Updates need Security Automation"

- R. Bisewski, "Comparative Analysis of Distributed Repository Update Methodology and How CoAP-like Update Methods Could Alleviate Internet Strain for Devices that Constitute the Internet of Things"

- J. Arrko, "Architectural Considerations with Smart Objects and Software Updates"

- J. Jimenez and M. Ocak, "Software Update Experiences for IoT"

- H. Tschofenig, "Software and Firmware Updates with the OMA LWM2M Protocol"

Appendix C. List of Participants

- Arnar Birgisson, Google
- Alan Grau, IconLabs
- Alexandre Adomnicai, Trusted Objects
- Alf Zugenmaier, Munich University of Applied Science
- Ben Campbell, Oracle
- Carsten Bormann, TZI University Bremen
- Daniel Thomas, University of Cambridge
- David Bosschaert, Adobe
- David Malone, Maynooth University
- David Plonka, Akamai
- Doug Leith, Trinity College Dublin
- Emmanuel Baccelli, Inria
- Eric Smith, SpinDance
- Jean-Philippe Fassino, Schneider Electric
- Joergen Karkov, Grundfos
- Jonathon Dukes, Trinity College Dublin
- Joseph Birr-Pixton, Electric Imp
- Kaspar Schleiser, Freie Universitaet Berlin
- Luca Zappaterra, Philips Lighting Research
- Martin Orehek, Munich University of Applied Science
- Mathias Tausig, FH Campus Wien
- Matthias Kovatsch, Siemens
- Milan Patel, Huawei

- Ned Smith, Intel
- Robert Ensink, SpinDance
- Robert Sparks, Oracle
- Russ Housley, Vigil Security
- Samita Chakrabarti, Ericsson
- Stephen Farrell, Trinity College Dublin
- Vassilis Prevelakis, TU Braunschweig
- Hannes Tschofenig, ARM Ltd.

Acknowledgements

We would like to thank all paper authors and participants for their contributions. The IoTSU workshop is co-sponsored by the Internet Architecture Board and the Science Foundation Ireland funded CONNECT Centre for future networks and communications. The program committee would like to express their thanks to Comcast for sponsoring the social event.

Authors' Addresses

Hannes Tschofenig
ARM Limited

Email: hannes.tschofenig@gmx.net

Stephen Farrell
Trinity College Dublin
Dublin 2
Ireland

Phone: +353-1-896-2354
Email: stephen.farrell@cs.tcd.ie