Network Working Group
Internet-Draft
Intended status: Informational
Expires: November 21, 2016

S. Farrell
Trinity College Dublin

May 20, 2016

Some Software Update Requirements
draft-farrell-iotsu-00

Abstract

The importance of software update as a mitigation for vulnerabilities discovered after deployment is widely recognised for both desktop and data centre applications and infrastructure. However, in the case of smaller devices, whether running on challenged networks or platforms or not, the situation is much worse, perhaps a decade or more behind that in better developed contexts. This memo proposes requirements for software update in situations where none is currently deployed, argues that that is the right target. In doing this, and perhaps somewhat in contrast to a vendor-driven approach, the interests of the individual device owner are emphasised.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 21, 2016.

Table of Contents

1. Introduction

This is a contribution to the IoTSU workshop. [1] It is not expected to ever become an RFC, but who knows?

The context for this memo is software update for devices that today lack a workable software update mechanism. That includes many or perhaps most very small (sometimes so-called "IoT") devices such as sensors and actuators, but also larger devices, such as printers, hubs and switches, WiFi access points and home routers. While there will be some implementation and deployment differences between different classes of device (e.g. very tiny devices might struggle with more onerous cryptography or network connectivity requirements), we believe many similar requirements apply in all these cases so considering common solutions and frameworks should be technically tractable.

All code has bugs that will need fixing, and all systems of whatever size will run code that includes bugs. (The author would welcome a better reference than this blog. [2]) And some bugs will make a system vulnerable to failure. So the base requirement is to be able to update code to fix bugs and make systems less vulnerable. In addition to inadvertent bugs however, there are also bad actors on the Internet, who will search for and exploit vulnerabilities wherever those are found. Devices deployed for any extended duration will inevitably be subject to such attack.

It is important to recognise that the searching-for part of that is a solved problem for the public IPv4 Internet [zmap] and is likely to be solved for the public IPv6 Internet. [RFC7707] And even if some devices are not directly accessible from the public Internet, many are, [secconsult] and transitioning "sideways" inside a breached network is a standard mode of operation for attackers and can be automated.

One might object that some smaller devices are not really sensitive, e.g. that a bad actor taking over a single thermostat, while possible, could be harmless. Such bad actors may however be attempting to leverage one vulnerabilitiy as a lauch-pad to attack other systems, as was the case with the Heating, Ventillation and Air Conditioning (HVAC) system used to start a major recent breach, [target] or as part of a pervasive monitoring [RFC7258] attack.

If we do manage to get a software update mechanism deployed, that system may itself include vulnerabilities. So if, as seems likely, cryptography is used to authenticate updates, then the signing keys used will be a very tempting target to attack. [update-msc] Such attacks are also more likely to succeed if there are very many such signing keys maintained by many vendors, some of whom will not be particularly well qualified to do that particular job. Some other vendors might even try to use a software update mechanism as part of a strategy to lock-in their customers. While no software update mechanism is likely to be able to counter such business choices, from the device owner perspective a vendor acting in that way is yet another bad-actor, and ought be treated as such. Both of these issues point towards considering a set of requirements and solutions where it is possible for some third parties to be involved in software update for many developers of devices.

One conclusion to reach here is that all devices are currently vulnerable to attack. If we select a random device, it may be that nobody knows the currently viable attacks that would work today, but we certainly know that some vulnerabilities are known, or will be found, for that device, once effort is expended on finding those. It is therefore, in the author's opinion, irresponsible to deploy devices for extended durations that we know have vulnberabilities, even if those are currently unknown, without some workable form of software update. It follows that all devices require software update.

2. Arguments for some infrastructure

Earlier, we described the context here as relating to those devices deployed on the Internet for which no working software update mechanism is deployed. That is a little different from the scope of

the workshop call for papers, which focused only on the so-called "IoT" space. In this section, we provide arguments for why that broader scope is the better one.

So long as there are any widely deployed devices with no software update, those will be exploited. It therefore makes sense to at this stage consider a broader range of devices than only those that are most-constrained. "Solving" the problem for only a specific class of device or network does not solve the problem.

The existence of open-source software and hardware, as well as "mixed" products (e.g. closed-hardware running some variety of almost-all GPL'd linux code), together with the number and size of the vendors of such equipment, means that such vendors and specific device deployments will not be of sufficient scale to by themselves support a specific software update infrastructure. Some sharing of infrastructure is therefore required, and that means that it's very unlikely that such infrastructure will be completely specific to one class of device or network.

Even where a vendor has found or been informed of a vulnerability, and has developed a patch or update, there are still many issues around the distribution of that. (Here's [3] a timely case in point.) Arguably, announcing but then failing to effectively distribute updates is worse than doing nothing, as patches themselves can be a source for malware developers. There is also a major difficulty with locating patches and with typing of update containers (tarball, zip, OS-specific executable, etc) that needs to be addresses in a more scalable manner over many vendors.

Devices in homes that benefit from software update will likely need to connect to some service on the public Internet in order to poll for updates. Let's assume all such traffic is encrypted as is proper, typically via use of TLS with server authentication. We need ways in which we can distinguish that traffic from potentially nefarious spying being done by devices. If many devices poll some pieces of shared infrastructure that only support software update then the probability that we engineer that well, and that workable control features are developed for end-user or home networks increases.

Vendors buy and sell one another, and cease to exist and products are end-of-life'd. Open-source projects can wither away over time. All of those mean that we need some form of infrastructure that will still be present despite such changes.

Device owners cannot be expected to know much or anything about software update, so any measures of quality or attempts to detect

misbehaviours will require some form of observatory or audit. That means that we need there to be an auditable number of sources for distributing software updates. Wth O(10^3) or fewer sources, that seems quite doable. With O(10^6) sources, that would appear to be significantly harder.

3. Requirements

In this section we posit some requirements for software update. We do not intend this to be an exhaustive list, but rather to be a list that captures key requirements for systems that do not currently have a working software-update mechanism.

We also make no claim that this is an authoritative list, rather this is a list to try start a discussion with those who implement and deploy devices and the software update infrastructure those will need.

In the author's opinion, the actual deployment of some software update scheme is far more important than meeting any specific subset of these requirements - once we can update a system then we can fix anything, given enough time and interest. (That seems to indicate there will be value in trying to construct a minimal list of requirements.)

Requirements posited below are numbered for ease of reference.

R1 All devices that have software or firmware that can be updated should be able to use a software update infrastructure developed to meet these requirements, regardless of whether that device is open- or closed-source, is made by a for-profit vendor or a community or academic projecct.

R2 One device can require software update from multiple sources, e.g., there can be different chips on a single board from different vendors. Or a larger device might have a "native" operating system inside which various containers are used to run applications, possibly with entirely different operating systems.

R3 Those update systems might need to co-operate, e.g. so that only one object needs to be downloaded to update multiple subsystems within the device.

R4 That co-operation might simply involve some kind of packaging of different otherwise indpendent sofware update objects so that co-ordination is mostly around transport and some form of meta-packaging.

R5 Co-operation might require that one system under the device owner's control act as a local server for software update. In the case of a device with a "native" and several "guest" OSes, one of these might be seen by the others as the source of software updates.

R6 In other cases, the device can take part in multiple independent software update schemes, with no co-ordination being needed.

R7 Data origin authentication is required, and overwhelmingly likely to be provided via a digital signature mechanism.

R8 There may be some systems where signature verification does not happen on the device to be updated but instead on some other upstream device - this can be because of algorithm implementation issues, or due to issues with Public Key Infrastructure (PKI) if say signature verification requires certificate status checking, which requires support for HTTP, which can again be too onerous for some systems.

R9 More than one PKI may need to be supported in updating even one device. There are widely deployed systems based on an X.509 PKI, others based on OpenPGP and perhaps others based on home-grown concepts similar to a PKI. [update-msc]

R10 It is very likely that multiple signers may need to be part of a solution, e.g. authenticating the orgin of the software separately from the download repository or perhaps separately authenticating the origin of a collection of updates from that for each member of a collection.

R11 It needs to be possible for sources of software to change, under the control of the device owner. The relevant software sources may or may not be co-operating with such changes.

R12 In the case of end-of-life devices, if there is a (typically open-source) community who could manage future updates then it is important to enable this without the device owner having to hack into ("root") their own device. (That last should be considered an anti-pattern in this context.)

R13 There may be devices where a choice of software sources exist. While there may be challenges in how to offer a user interface to allow the device owner such a choice, it must be possible to offer such choices where they exist.

R14 Devices will need to poll for software update. It seems unlikely that most devices can act as a server listening for incoming connections from software sources. That means that devices will need to establish a way to connect to a software source, or some entity who can distribute updates.

R15 When the device is in the home, or carried on a person, then there are significant privacy issues that arise. While in those cases, encrypting the interactions may be required, it may also be required that it is clear to the person (or their home network) that all that is happening is software update. To give a concrete example, the author would be happy if a television called home to check for a software update, but very unhappy if the same device called home to tell someone what channel changes are occurring.

4. Security Considerations

While software update does mitigate vulnerabilities, it also creates new vulnerabilities. First, software signing private kes are a hugely attractive target, and have subject to real attacks in the past. [flame]

The software updates that are distributed are also usable to re-create the vulnerabilities that are being fixed, so that unpatched systems are put at further risk once the patch details reach a bad actor, which is impossible to prevent. Automation of attack-generation code is also possible, [auto] though still difficult. That could allow bad actors to exploit the time window during which updates propogate to the deployed base. There are therefore good reasons to try to minimise this duration.

Patches can also include new vulnerabilities. While this is true, that is out of scope for this document which considers the software update mechanisms and not the quality of the software being updated.

5. Privacy Considerations

If the software update mechanism exposes any correlatable identifier then it becomes a useful way to track people who carry devices that poll for updates.

Devices within homes or vehicles or other sensitive locations can also expose privacy sensitive information via the software update mechanism. For example, if the device polls the update server on power-on, and if power-on is associated with some event such as a homeowner arriving home, then the existence of the packets

(regardless of encryption) leaks some possibly privacy sensitive information.

We will likely want to use a confidentiality service in order to not expose identifiers to network attackers. That creates the potential for the device to use the software update channel (or any other indistinguishable channel) as a covert channel. Devices have been known to exfiltrate privacy sensitive data for the benefit of the device maker [refs] so if we can provide some way to increase confidence that only software update is happening, then that can be beneficial for all concereed (except bad actors). It may help if many devices poll some well-known services/hosts on the Internet and do not call-home to the device maker's network.

6. IANA Considerations

This document makes no requests for IANA action. This section would be removed except it won't be as we're not aiming for publication as an RFC.

7. Acknowledgements

TBD - your name here for comments or beer! But that assume there'll be another revision of this which may not happen.

8. References

8.1. Informative References

[auto] Brumley, D., Poosankam, P., Song, D., and J. Zheng, "Automatic patch-based exploit generation is possible: techniques and implications", 2008, <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4531150>.

[flame] Sotirov, A., "Analyzing the md5 collision in flame", 2012, <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.675.2256>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <http://www.rfc-editor.org/info/rfc7258>.

[RFC7707] Gont, F. and T. Chown, "Network Reconnaissance in IPv6 Networks", RFC 7707, DOI 10.17487/RFC7707, March 2016, <http://www.rfc-editor.org/info/rfc7707>.

[secconsult] Viehbock, S., "House of Keys: Industry-Wide HTTPS Certificate and SSH Key Reuse Endangers Millions of Devices Worldwide.", 2015, <http://blog.sec-consult.com/2015/11/house-of-keys-industry-wide-https.html>.

[target] Weiss, N. and R. Miller, "The target and other financial data breaches: frequently asked questions", 2015.

[update-msc] Ruissen, P. and R. Vloothuis, "Insecurities within automatic update systems v1. 16", 2007, <https://staff.science.uva.nl/c.t.a.m.delaat/rp/2006-2007/p32/report.pdf>.

[zmap] Durumeric, Z., Wustrow, E., and J. Halderman, "Zmap: fast internet-wide scanning and its security applications.", 2013.

8.2. URIs

[1] https://down.dsg.cs.tcd.ie/iotsu/

[2] https://amartester.blogspot.ie/2007/04/bugs-per-lines-of-code.html

[3] http://arstechnica.com/security/2016/05/foul-mouthed-worm-takes-control-of-wireless-isps-around-the-globe/

Author's Address

Stephen Farrell
Trinity College Dublin
Dublin 2
Ireland

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