Software Update in a multi-system Internet of Things

Ted Hardie, Google
Position paper Submitted to the IoTSU Workshop

In many common deployments for the Internet of Things there is a vertically integrated stack, one aspect of which presents the management interfaces for all the devices within the stack. In those systems, software update systems present the usual problems of trustworthiness, effectiveness, and timeliness, along with some specific issues around the constraints of the devices. Those constraints may limit the possibilities for running local checks of software signatures, may eliminate the possibility of maintaining a cache of software suitable for rollback, and may hinder the ability to audit or even establish the current version of software. All of those are serious challenges, but in systems where there is no vertically integrated stack, things are a good bit worse.

The most obvious issue is that there will be no single interface for managing the devices in a specific context (e.g. home, business, or other facility). There is also no way to tell within a specific management interface that a low-level device is being used in multiple systems. An example here is a motion detector that is simultaneously used both by a lighting system and by a burglar alarm system. In one context, it may light room lights when motion is detected. In the other, motion that occurs after the alarm system is armed may cause a silent alarm, trigger locks, or wake other sensors. In that multi-system context, a software update intended to improve functionality in one system might have serious consequences for a different system. Our motion sensor might be adjusted to limit the chance that a family pet will trigger the lights, to take one example, accidentally reducing the chance of detection for an intruder who stays close to the ground. While that particular example may belong in a heist movie, we can imagine others. A temperature sensor visible to more than one system, for example, may be re-configured from Celsius to Fahrenheit in one system, causing other systems to get false readings for any data not accompanied by the scale.

The reality is that any system deployed as an overlay of sensors and actuators will have some level of uncertainty about whether other systems are using the same data in a different overlay. The sensors and actuators may not themselves know which systems are listening, especially in wireless deployments, or they may lack the ability to report it. As a result, it may be prudent to presume that any system outside a vertically integrated stack shares some of its devices with other systems.

At first glance, that might seem to preclude software update without risk. There are, however, some common-sense approaches to limiting the impact of these issues. The first of these is probably the most important: avoid changing the output parameters of a declarative device if you can change the interpretation instead. Instead of reconfiguring a temperature sensor to

report in a new scale, change the thermostat reading it so that the temperature is displayed after a conversion. Similarly, do not change a motion sensor to limit its reporting of motion below a set height, but instead mask that data when it arrives at the lighting console. Given that the thermostat and lighting console are likely more capable systems in any case, adjustments to them should also face fewer constraints for software update.

The second common principle is to avoid inferences based on incomplete versioning. The author believes that it is good practice to send data using the most primitive schema and element which yields a correct interpretation (for privacy reasons outlined elsewhere[1]). A corollary to that is that no system interpreting data from a device or actuating a device should assume that the schema or version ascribed to one data element is the same for all data elements or activities. An update to the capabilities of a binary light to change its color temperature, for example, might not change the data element for switch power.

A third principle is that changing the rate of reporting may have unintended consequences and should be avoided when possible. While some systems are composed of periodic reporters and always on receivers, others rely on a mesh with synchronized or staggered-trigger sleep-wake cycles. Changing the rate at which a device reports can caused periodic listeners to fail to get data or, in some cases, to misinterpret that data.

The fourth principle is that a software update that is applied without knowledge of all of the systems using an element cannot take the full security into account. If it does not know what threat models those systems protect against, it cannot take into account the impact of the changes on their security. The best it can do is to take into account common threats, such as blocking commands to actuators or falsifying data and attempt to do no harm to the mitigations for those which may be in place. The implication will often be that software updates should be as limited as possible to achieve their desired effects.

A multi-system deployment of the Internet of Things presents challenges besides the need to mitigate the risks above. Each component will likely have different update mechanisms, different trust anchors, and different methods for audit and discovery of suitable versions; managing those presents a considerable burden. Despite that, support for a multi-system Internet of Things may be critical to ensure that the field does not succumb to walled gardens and vendor lock-in. In other words, if IoT is to truly be part of the Internet landscape these are prices we should be willing to pay.

[1] https://www.iab.org/wp-content/IAB-uploads/2016/03/LociofInteroperabilityforIoT.pdf