Voluntary Framework for Enhancing Update Process Security

Technical Capabilities and Patching Expectations Working Group

This document was drafted by an open working group convened by the National Telecommunications and Information Administration in a multistakeholder process, including the following individuals and organizations: Arm; Mark Cather, UMBC; Chris Gates, Velentium; Tim Hahn, IBM; Shrinath Eswarahally, Infineon; Ethan Lucarelli, Inmarsat; Verizon. Others participated, but do not wish to be named.

Draft: December, 2017

Motivation

The proliferation of devices and growth in the Internet of Things (IoT) provides the opportunity for technical advances that could dramatically improve people's lives. As devices become increasingly integrated into society, security and safety risks to individuals, businesses, and society may increase without appropriate, risk-based security measures. One important tool to help mitigate risks is the ability to reliably update software within devices remotely, through a network via both wired and wireless connections (often referred to as "over-the-air" or OTA updates).

The level of update process security appropriate for a particular device will vary depending upon the manufacturer's unique business needs, resource availability, and risk tolerance. Executing a basic, over-the-air update can be done without including any enhanced update process security features; however, such an update may be vulnerable to blocking, spoofing, or other malicious attack. With no security mitigations, updates may inadvertently reduce the security of a device. Decisions concerning whether to employ additional features to enhance the security of the update process should be risk-based, achieving security goals in a cost-effective and prioritized manner.

The term "updatable" does not mean the same thing to everyone. Different people and different organizations may have their own ideas about what the term does (and should) mean. To better address update process security risks, it is important to have a common understanding of updatability. This can support manufacturers, purchasers, and other IoT stakeholders as they make risk-based decisions to enhance update process security.

Document Overview

This document is designed to support manufacturers in identifying and selecting appropriate, risk-based security features to mitigate vulnerabilities in the update process. Part I of this document provides an overview of basic steps in an illustrative update process. Part II provides

a menu of voluntary processes that manufacturers may choose to adopt to enhance the level of security in the update process depending upon individual business needs and risk tolerance. The appendix details specific security aspects and mitigations common to all steps.

Of note, update process security is only one aspect of many in enhancing the overall security of an IoT device. Devices may have vulnerabilities that cannot reasonably be addressed through updates. There are also physical and human security vulnerabilities that may not be addressable by software updates. This larger challenge of mitigating IoT device vulnerabilities can be addressed only through a set of coordinated actions, including industry best practices, device management and cyber-hygiene solutions, and greater awareness of context-specific risks. Adoption of the highest level of over-the-air update process security enhancements does not guarantee the security of the device itself. This voluntary guidance, therefore, is intended only for the limited purpose of developing a common lexicon to support manufacturers in enhancing security in effecting over-the-air updates.

Audience and How to Use This Document

This document is intended to provide IoT device and software manufacturers with a common lexicon or language to discuss risk mitigation in the over-the-air update process. A clear framework for updatability, with defined steps, will allow discerning risk-aware decision makers to understand the value of particular security features. Manufacturers[1] of devices and the components that go into devices have many reasons to be interested in their devices' update process. Further value results from a shared model of update processes across the diverse IoT product space. Even if the devices are quite different, the steps of the update and the potential security features are often similar.

Manufacturers can take steps to enhance the security of devices they design, produce, and sell. Voluntary hardware and software design guidance could aid manufacturers in achieving their unique security goals throughout their product development, production, sales, and support processes. Note that the information in this document is not explicitly targeted to end-consumers. Manufacturers may wish to consult "Communicating IoT Device Security Update Capability to Improve Transparency for Consumers" by the Working Group on Communicating Upgradability for guidance on how to communicate updatability to end-consumers. This document also does not address security risks from legacy devices or orphan devices that are no longer maintained.

This document references voluntary guidance from NIST, which is current as of November 2017, to provide additional background and information that may be helpful in enhancing security. This guidance is illustrative and subject to change. Readers are therefore encouraged to consult the most recent relevant industry guidance and best practices to inform decision making, particularly on encryption. For a more robust compilation of voluntary best practices

[1] The term "Manufacturer" used throughout this document is understood to also represent an "assigned agent," "service provider," or "vendor" as authorized and enabled by the Manufacturer.

and guidelines on IoT security, please refer to the draft Standards Catalog developed by NTIA stakeholders.

The information in this document may also be useful to enterprise-level procurement processes. Having knowledge of the importance of security updates and the risks associated with certain technologies can better guide decision-making.

Part I: Basic Steps in an Illustrative Over-the-Air Update Process

An update process can be broken up into a linear sequence of steps that broadly describes the sequence to be followed.[2] In practice, errors will occur in the update process requiring the linear process to be aborted or restarted from a previously known state. While the set of steps below is presented as a linear set of processes, note that at each point if an error or incorrect situation is detected the process may need to be reversed, cleaned up, and started again from a known good state. Refer to the Appendix "Rollback on Failed Update" section for additional information regarding fallback and recovery.

For each step, **security features** can vary greatly depending upon the intended final security posture desired by the manufacturer. While the sequence of steps is common to all risk models, the security features implemented in each step ultimately determines the level of security of the update process. The following normative sequence of update steps are the basic elements in the update process.

0. Create: Manufacturer creates image
    1. This step is assumed to be out of scope for this guideline, but is represented here as it is seminal to the initialization of this process.
    2. The update image or multiple update images are packaged into a deliverable structure.[3]
1. Sign: Ensure integrity and authenticity of update
    1. Manufacturer includes a signature or signatures in the update deliverable, to be used to vet the integrity and authenticity of the update deliverable contents.
2. Protect: Prevent exposure of update deliverable
    1. Manufacturer subjects the update deliverable to a translation (including encryption or obfuscation) to prevent exposure of software image.
3. Send: Data in motion

[2] This process allows for the presence of multiple upgradable targets in a given device, which may need to be upgraded at the same time.

[3] This step still has security concerns. Recent attacks on the update process have demonstrated the importance of the integrity and security of the update process itself. One stakeholder recommended the trust in the update process not rest on a single key or servers, and that at least one key required for an update to be trusted should be kept on a non-Internet-connected device.

    1. The update deliverable is communicated to the target system / device.[4]
4. Receive: Receive update deliverable
    1. The target system / device receives update deliverable.
5. Check: Process update deliverable
    1. Target system / device validates integrity of update deliverable.
    2. Target system / device decrypts / de-obfuscates update deliverable.
    3. Target system performs any special handling of update images as indicated.
6. Announce: User awareness of update on device [5]
    1. End user notification and/or approval of update installation.
7. Distribute: Distribution
    1. Update deliverable is parsed and distributed to intended target devices
    2. Distribution may be of a recursive nature.
8. Process: Process update image
    1. Each target (CPU, MCU, FPGA, etc.) receives its update image.
    2. Each target decrypts the update image (if encrypted).
    3. Each target validates the integrity of the plain text update image.
9. Stage: System pre-update state
    1. Any activities that need to be performed before the update occurs.
    2. Potentially initiate a backout state for recovery, in case the update fails.
10. Apply: Trigger update process
    1. Perform the actual update process of installing the update image.
11. Re-verify: Post-update verification
    1. Each target validates the integrity of the installed update.
    2. Communicate results of verification to relevant targets.
12. Activate: Activate / enable updated code
    1. New updated code actually begins to be executed on the target (assuming successful verification).
13. Clean-up: Post-update activities
    1. Verify that system is functioning appropriately.
    2. Post-processing messaging (internal[6] & external) and cleanup from update.
    3. This could include a negative outcome.

Part II: Security Features to Enhance Over-the-Air Update Process Security

[4] Steps should be taken to ensure that the appropriate code is sent to the appropriate device. This is outside the scope of this document.

[5] If the update is automatic, without requiring user involvement, then this step may not occur, or it may happen later as a notification to allow the user to understand that the current software and firmware has been updated. Alternatively, it may occur earlier to give the user notice and choice about downloading the update code.

[6] "Internal" refers to other targets inside of the device being updated; "external" refers to servers and other remote entities.

The steps listed above are necessary to ensure the integrity and the reliability of an update and the update process. However, without the addition of security specific features at each step, the steps themselves are vulnerable to attack by malicious actors and may in fact make the target device more vulnerable with an update process than without one.[7] The security needs of each step vary based on context, threat, etc.

Below, we present a framework to understand security features that could be implemented at each step to improve the security of an update process. The features themselves map to the steps of an update. Because the security decisions should be based on needs, context, technical capabilities, and risk evaluation, the security features are presented as a 'menu,' from basic security features, all the way to features designed to resist quantum computing-assisted attacks in line with current state-of-the-art encryption guidance.

1. Sign: Ensure authenticity and integrity of update

Risks: If there is no authenticity and integrity check, there is no way of verifying what you received is what you were supposed to receive, or that it originated from the expected source. Perhaps the largest risk of an update process is the potential for a third party to push malicious code onto the device. Changing the code on the device could lead to any number of deviations from the intended functionality, including preventing expected operations, adding new, undesired functions, changing how data flows from the machine, or weaponizing the device to attack other targets.

Mitigations: Signing the update payload cryptographically protects the integrity of the payload, including from undetected intentional modification by a bad actor. It also provides authenticity in the provenance of the payload. This is different from a more traditional approach of using non-cryptographic hash such as a cyclic redundancy check (CRC) or a checksum. These non-cryptographic hashes can validate the integrity against naturally occurring corruption of the payload, but can be easily subverted by bad actors. Similarly, failure to use a strong enough cryptographic signature or hash function also fails to completely mitigate these risks. For older, weaker hash functions, an attacker with sufficient motivation and resources could generate a malicious update that generated the same hash as the legitimate update.

Basic Implementation: Cryptographic signatures are used to detect modification and establish provenance of the update payload. Per NIST SP 800-131A, acceptable key lengths for signing are 2048-bit for RSA and 224 for ECDSA, and acceptable hash functions are the SHA-2 family.[8] Other, less computationally intensive, algorithms exist (e.g., hashing functions like MD5, SHA-1, etc.) but have been found to be susceptible to various forms of attack[9]. NIST SP 800-89 (Recommendation for Obtaining Assurances for Digital Signature Applications) provides recommendations on digital signature key management.

[7] Such attacks include (but are not limited to): man in the middle, spoofing, "bricking" the device, denial of service (blocking the update), version downgrade, and key theft.
[8] http://dx.doi.org/10.6028/NIST.SP.800-131Ar1
[9] http://nvlpubs.nist.gov/nistpubs/ir/2017/NIST.IR.8114.pdf

2. Protect: Prevent exposure of update deliverable

Risks: This step refers to improving the protection of an update's confidentiality or intellectual property by encrypting it, rather than sending it "in the clear." (Note that protecting confidentiality through encryption is not the same thing as protecting integrity through cryptographic hashes and signatures.) An update sent without encryption risks exposure of code contained in the device or update payload. This could allow an attacker to subvert other protections in the device through reverse engineering thus allowing all similar devices in the field to become exploited by being used in a botnet. In a business risk to the manufacturer, a competitor could extract valuable algorithms or techniques in the software.

Mitigation: Encryption of the update before transmission and decryption of the update on the device can reduce the risk of exposure during transmission regardless of the communications path(s) of the update deliverable.

Basic Implementation: Application layer encryption is used to protect the confidentiality of the update from time/place of creation to time/place of use. Best practices around application level encryption are available in the NIST Report on Lightweight Encryption.[10]

Further Security Considerations: Once the update deliverable has been received by the device, device design mitigations should be considered to avoid exposing the decrypted update deliverable in the event of a physical attack on the device during the update process.

Optionally, additional encryption provided by the communication path(s) can also be implemented as a secondary level of encryption to further mitigate the risk of exposure while the encrypted update deliverable is being distributed to the device. Note that communications path encryption only prevents exposure over the communications path. Any intermediate holding locations may result in exposure of the information (see data in motion section).

3. Send: Data in motion

While an update payload is being transmitted to a device, the path of communications may take several radically different types of communications (i.e., Ethernet, cellular baseband, satellite transmission, Wi-Fi, Bluetooth, etc.), many with no inherent security or integrity checking of the update payload. Identifying communicating parties is often part of establishing a secure communication channel so that senders and receivers know with whom they are communicating.

Risks: The risks at this step are captured in the two preceding steps: compromise of the integrity of the update and the confidentiality of the update. Transport layer authentication and encryption allow an extra layer of defense.

[10] http://nvlpubs.nist.gov/nistpubs/ir/2017/NIST.IR.8114.pdf

Basic Implementation: Transport-layer encryption, such as TLS or BLE 4.2+, can provide widely accepted levels of security between the endpoints. Using features such as pinning of certificates in TLS can authenticate the source, and user-pairing of devices in BLE can authenticate endpoints. VPNs also offer confidentiality and integrity of data in motion.

Further Security Considerations: “Defense in Depth,” where multiple security mitigations are overlaid in a redundant manner may be desirable. Endpoint validation can be improved by cryptographically confirming that the end system/device is the correct target before delivering the update, such as challenge/response mechanisms or pre-shared secrets.

4. Receive: Receive update deliverable

The device receives the update.

No design risks are specifically associated with the required step. However, normal good security hygiene practices should be followed, such as mitigations against buffer overflow.

5. Check: Process update deliverable

For this step, the device confirms that the update is valid. This step validates the security of the previous steps, which may include checking the identity of the device, and confirming that the update is appropriate for this device. The target system also performs any necessary special handling of update images. This is a critical step in the protection of the device itself, where ultimately each device needs to protect itself from a potentially malicious update payload, and all previous mitigations are enforced during this step.

Risks: In addition to the concerns above about authenticity and integrity (signature) and confidentiality (encryption), there are a number of security and performance concerns. In a “downgrade attack,” an older authentic update payload is substituted by a bad actor to re-introduce known vulnerabilities in the target to be exploited in a secondary attack. The update itself could be misconfigured to harm the device or its functionality.

Basic Implementation: In addition to the signature and encryption features above, a monotonic versioning system can prevent a downgrade attack.

Further Security Considerations: A system capable of disallowing previous versions requires an additional step for a manufacturer-driven rollback update, and can make user-driven rollbacks more complex. Alternatively, the device can securely validate the path and source of the update to ensure that the older version is not coming from an untrustworthy source.

6. Announce: User awareness of update on device

The manufacturer may wish user engagement in the update process. An update process may temporarily impede the functionality of a device, or the user may have some other interest in

approving an update beyond an automated process. In these instances, the user should be made aware of the presence of an update on the device to take further action.

Risks: Strictly speaking, this is not a security risk, but is a functional risk, as the process of updating the device may interrupt intended functionality or timing of functionality, resulting in a period of denial of service.

Risks of failing to properly address the concerns of this step include:

1. Non-performance of device intended functionality during an update process.
2. If the device is under the control or partial control of a bad actor (e.g., botnet), the bad actor may wish to remotely block a security update.
3. A legitimate user may wish to avoid an update that implements unwanted features.
4. The device may not be able to communicate to the appropriate user that an update is available, leading to out-of-date, vulnerable devices.

Basic Implementation: Manufacturer should consider the use and installation of a device to determine the optimal approach to automatic updates, user control, and uptime criticality.

Depending on the context and use case, there will likely be a need for a balance between giving a user a choice in the updating of devices and pushing an update after a period of time for the good of the user and everyone else on the internet. For more on the question of end user approval of updates, see "Communicating IoT Device Security Update Capability to Improve Transparency for Consumers" by the Working Group on Communicating Upgradability.

Further Security Considerations: If the user does not take action to update the device, the manufacturer or device administrator may wish to take further actions at a future date. How to address a non-updated device is outside the scope of this document.

7. Distribute: Distribution to devices

This guideline allows for multiple update targets in a given device or system, as such the potential to support a hierarchical relationship between update targets.

Risks: The update payload may be decrypted by the initial target and then distributed to sub-systems without further integrity or content protection (see section on "Protection"). This could expose intellectual property, more easily enabling reverse engineering, or potentially allow the code to be modified as it moves through the systems.

Basic Implementation: Update image remains encrypted and integrity protected while in motion if traveling across exposed transport media. Support is provided for multiple of layers of systems, devices, or CPUs to be targeted.

Further Security Considerations: The adversary might still be able to try to compromise non-exposed internal communication channels. To address this residual risk, the update image should remain encrypted while in motion.

8. Process: Process update image

Similar to Step #5, “Check,” this may be performed on a target in a lower “child” relationship, if a hierarchical relationship between update targets is implemented.

Basic Implementation: Each target validates the integrity of the plain text update image using a cryptographic hash signature. Each target decrypts its specific update image, if encrypted.

9. Stage: System pre-update state

During this step, any activities necessary to performing the update on the device can occur, including functions such as erasing flash memory, placing the device in a “safe mode” of operation, ensuring sufficient battery life to complete the operation, etc.

Risks: Decrypted update image is present in device’s memory for an extended period of time while waiting for this step to complete.

Basic Implementation: No security features are assumed; manufacturer may define them in specific contexts.

10. Apply: Trigger update process

The actual update process occurs. This includes writing to a file structure, updating the binary program space in flash memory, etc.

Risks: Decrypted update image may be communicated via internal communication channels or busses, and risks being intercepted and exposed.

Basic Implementation: No special processing is assumed.

Further Security Considerations: When updating multiple internal targets, coordination of timing should be considered. If needed, conversion of persistent data on each target should occur during this step. Update is placed into a separate flash region from existing image for reliability purposes (in case a failed update requires a rollback to the previous working version).

11. Re-verify: Post-update verification

As the importance of utilizing the intended, correct update image is paramount to the update process, one last test of the update integrity is performed before this new update software is executed.

Risks: Something went wrong in the update.

Mitigation: A redundant test of update integrity would confirm that the process of writing the update image to the intended target was performed correctly and that no malicious actor or device / memory failure altered the intended update image.

Basic Implementation: Each target vets the integrity of the installed update.

Further Security Considerations: Potentially use cryptographic hashing.

12. Activate: Activate/enable updated code

Once the code has been verified, it is actually enabled, and execution path switches to the new, updated code. No additional risks are incurred during this step.

Basic Implementation: No special processing is assumed.

Further Security Considerations: If multiple flash images are stored (redundant duplicate copies, or previous and current), then activation may entail pointing to the new image for subsequent boot cycles.

13. Clean-up: Post-update activities

Following an update operation, one final step is performed to clean up any loose ends, such as buffered data or encryption keys and intermediate values in memory. Post processing logging and messaging may be performed to inform other device targets of the status of the update on this target. The device may also communicate back to an external server the status of the update process. This status should identify the target in question; the particular device in question, and the outcome (which may not be a successful one).

Risks:

1. Some targets may not receive the status update and may not act in concert with all the other targets.
2. The external server may not receive the status update (either through a communication failure or malicious interception) and may believe the target or device has not been updated or be unaware that the target or device hasn't been updated, if the update operation did not succeed.

Basic Implementation: No special processing is assumed.

Further Security Considerations: If one is concerned about accidental or intentional communication failure, several mitigations exist. One approach is to allow remote querying by a central server. However, this can introduce further risks of attack by confusion or denial of

service. Discerning legitimate requests may require further validity checks. It is left up to the implementer to identify the appropriate solution.

For robustness of the system, one concern is coordination of versions between targets on a multi-target system. One solution is notification of successful update across the system by each target.

Appendix:

The following areas are common to multiple steps in the update system, and may address concerns beyond the security of the update process.

Key Management

Many of the voluntary, recommended steps in this document rely upon the presence of “shared secrets” such as cryptographic encryption keys. These keys should be ephemeral and unique to the targeted device to mitigate the risk that additional devices could be attacked if one device’s keys are exposed.

These keys are created/exchanged and stored in the device in “secured nonvolatile memory,” in a manner that should protect these keys from exposure. To mitigate the risk of key exposure, proper key management techniques should be followed (see NIST SP 800-57, 800-147, and NISTIR 8114).

The attacker may try to read the keys from the device’s memory during the decryption phase. Against this level of attacker, the device would be better protected through usage of a hardware component designed specifically to store, use, and process cryptographic keys.

Rollback on Failed Update

To reduce the risk that a failed update operation could result in a non-operational device, manufacturers may consider developing a nonvolatile program space with over twice the size of the update deliverable payload. For example, if the maximum intended executable image is 1 MB in size, then the available program space memory would be slightly greater than 2 MB. Besides the concurrent storage of two executable images, there is also the program storage space used for the update agent, and the boot handler code.

In a target device or system, targeted elements may wish to communicate this update failure between all of the other targeted elements of the update (such as multiple microcontrollers) in the device and potentially the system. In this manner, the entire device/system could be rolled back to its pre-updated operational state. The failure may also be communicated to the device owner/operator via on-device interface or the administrator or manufacturer over the network. Network communication should follow the security practices discussed above.

Power Budget

The update operation can consume significant amount of battery life on battery powered devices. Prior to initiating an update, checks should be performed that ascertain if the device’s current battery status can tolerate the power consumption required for the complete update process period of time. If insufficient battery life remains, the update operation should be delayed until such a time that the device can support the entire update process.

Hardware-Accelerated Cryptographic Functionality

The cryptographic functions discussed in this document (including decryption, hashes such as SHA1/SHA256/CMAC, random number generation, Physically Unique Function [“PUF”], and secure boot) are slow and consume significant amounts of power when implemented solely in software. Hardware acceleration offers orders of magnitude differences in performance, as well as very large savings in RAM and program space utilization.

Many modern chips offer these capabilities without significant cost increases. These hardware-accelerated cryptographic functions are compatible with the results of the software-based cryptographic functions, so they can be used interchangeably in a system being updated. Whenever possible these hardware-accelerated functions should be used in preference to the software-based operations.