SANDIA REPORT
SAND2021-11150
Printed September 2021

Recommendations for Distributed Energy Resource Patching

Jay Johnson, Ingo Hanke

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico
87185 and Livermore,
California 94550

ABSTRACT

While computer systems, software applications, and operational technology (OT)/Industrial Control System (ICS) devices are regularly updated through automated and manual processes, there are several unique challenges associated with distributed energy resource (DER) patching. Millions of DER devices from dozens of vendors have been deployed in home, corporate, and utility network environments that may or may not be internet-connected. These devices make up a growing portion of the electric power critical infrastructure system and are expected to operate for decades. During that operational period, it is anticipated that critical and noncritical firmware patches will be regularly created to improve DER functional capabilities or repair security deficiencies in the equipment. The SunSpec/Sandia DER Cybersecurity Workgroup created a Patching Subgroup to investigate appropriate recommendations for the DER patching, holding fortnightly meetings for more than nine months. The group focused on DER equipment, but the observations and recommendations contained in this report also apply to DERMS tools and other OT equipment used in the end-to-end DER communication environment. The group found there were many standards and guides that discuss firmware lifecycles, patch and asset management, and code-signing implementations, but did not singularly cover the needs of the DER industry. This report collates best practices from these standards organizations and establishes a set of best practices that may be used as a basis for future national or international patching guides or standards.

ACKNOWLEDGEMENTS

The authors would like to thank the SunSpec/Sandia DER Cybersecurity Workgroup subgroup on patching. Contributors included many industry experts: Ifeoma Onunkwo, Aegir Jonsson, John Drummond, David Cohen, Allyson Ugarte, Michael Cowan, AJ Rossman, Alfred Tom, Brian Noel, Daniel Sylawa, Emily Hwang, Dzung Nguyen, Gordon Lum, Ron Ih, Hassan Mohammed, Ioannis Zografopoulos, Keith Brown, Keith Rose, Randy King, Andrew Levy, Matthew Hyde, Mehmet Cintuglu, Nate Diamond, Prasanth Gopalakrishnan, Roger Alexander, Shuo Chen, Todd Davis, Assaf Ben Or, Nicholas Manka, Candace Suh-Lee, Alexander Miranda, Kyle Song, Kirk Rosener, Tanner Kural, Bret Young, Donald Coffin, Leon-Gerard Vandenberg, Kevin Harnett, Graham Watson, Jeremiah Miller, Kemal Celik, Steve Peters, Robert White, Chris Connell, Susanna Huang, Pietro Calo, Cuong Nguyen, Steve Wurmlinger, Limin Huang, Jon Bichel, Jules LaPrairie, Nir Hasson, Assaf Harel, Michael Allen, Larry Truong, Yong Du, Gaurav Shah, James Chase, Terence Parker, Glenn Skutt, Kip Gering, Nirmal Thaliyil, William Rubin, Dachuan Yu, David Lawrence, Edward S, Ray Fleming, Gerald Robinson, Linkesh Diwan, Michael Flood, Min-Ju Hsieh, Paydon Wilson, Paul Duncan, Robert Flottemesch, Sulaiman Speed, Virgil Titeu, Zubair Ahmad, Mike Savoie, Albert Moussa, Eric Kashiwamura, Charalambos Konstantinou, AJ Rossman, Danish Saleem, Dave Chisholm, Dave Bradshaw. We also thank Timothy Zgonena and Douglas Biggs from UL for their review of the recertification section of the paper.

Additional comments or questions may be sent to Jay Johnson (jjohns2@sandia.gov) or Ingo Hanke (ingo.hanke@sma.de).

This material is partially based upon work supported by the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy (EERE) under the Solar Energy Technologies Office Award Number 34216.

CONTENTS

LIST OF FIGURES

ACRONYMS AND DEFINITIONS

Acronym	Definition
ACL	Access Control List
ANSI	American National Standards Institute
API	Application Programming Interface
CIP	Critical Infrastructure Protection
DER	Distributed Energy Resource
DERMS	Distributed Energy Resource Management System
HTTP	Hypertext Transfer Protocol
IEC	International Electrotechnical Committee
IEEE	Institute of Electrical and Electronics Engineers
IT	Information Technology
NERC	North American Electric Reliability Corporation
NIST	National Institute of Standards and Technology
OEM	Original Equipment Manufacturer
OT	Operational Technology
PKI	Public Key Cryptography
PV	Photovoltaic
RTO	Regional Transmission Organization
SDO	Standards Development Organization
SP	Special Publication (from NIST)
TLS	Transport Layer Security
TSO	Transmission System Operator

TERMS AND DEFINITIONS

Term	Definition
Acess Control	The process of limiting access to the resources of a system only to authorized programs, processes or other systems
Asset	Anything that has value to an organization, including, but not limited to, another organization, person, computing device, information technology (IT) system, IT network, IT circuit, software (both an installed instance and a physical instance), virtual computing platform (common in cloud and virtualized computing), and related hardware (e.g., locks, cabinets, keyboards). (From NISTIR 7693)
Cybersecurity patch	A patch which addresses one or more cybersecurity vulnerabilities
DER Aggregator	Entity which communicates with a collection of DER devices, typically for the purpose of providing grid services (See: *DER Service Provider*), customer services, or other services.
DER Integrator	Entity which interconnects DER equipment to the power system and performs the commissioning process.
DER Owner	Person or entity that owns the DER device or system.
DER Product Supplier	DER vendor, Original Equipment Manufacturer (OEM), or the entity accountable for supplying patches, per IEC 62443-2-3.
DER Service Provider	Entity which provides grid services to another party such as a distribution or transmission system operator.
Device	Electronic equipment made for a particular purpose
Firmware	Permanent software programmed into a read-only memory
Non-repudiation	Assurance that the sender of information is provided with proof of delivery and the recipient is provided with proof of the sender’s identity, so neither can later deny having processed the information. (From NIST SP 800-18 Rev. 1)
Patch	A software component that, when installed, directly modifies files or device settings related to a different software component without changing the version number or release details for the related software component. (From ISO/IEC 19770-2). A patch is an immediate solution to an identified problem that is provided to users. A patch is usually developed and distributed as a replacement for or an insertion in compiled code, e.g., a binary file or object module. (From NIST SP 800-44 Version 2)
Patch management	The systematic notification, identification, deployment, installation, and verification of operating system and application software code revisions. These revisions are known as patches, hot fixes, and service packs. (From NIST SP 800-137)
Public Vulnerability	A vulnerability that is accessable from the public domain, i.e., a publicly-disclosed vulnerability.
Update	New, improved, and/or fixed software, which replaces older versions of the same software. For example, updating an OS brings it up-to-date with the latest drivers, system utilities, and security software. Updates are often provided by the software publisher free of charge. (based on NIST SP 1800-15C) See: *Patch*.

Term	Definition
Upgrade	A patch or other modification to code that corrects security problems and/or expands functionality of software. (Adapted from NIST SP 800-40 Rev. 3) See: *Patch*.
Vulnerability	Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source. (From NIST SP 800-53 Rev. 5)

1. INTRODUCTION

Patch management is part of a comprehensive cyber security strategy that resolves bugs, operability, reliability and cyber security vulnerabilities. A patch, as defined in NIST SP 800-44 Version 2[1] and ISO/IEC 19770-2[2], is a software component that, when installed, directly modifies files or device settings. A patch is an immediate solution to an identified problem that is provided to users. A patch is usually developed and distributed as a replacement for or an insertion in compiled code (e.g., a binary file or object module). In many operating systems, a special program is provided to manage and track the installation of patches. Patches are also called "repair jobs" for a piece of programming, fixes, software updates, software upgrades, firmware upgrades, service packs, hotfixes, etc.

In the case of the DER industry, equipment is often expected to operate in the field for 25 or more years. During this period, there will undoubtably be newly discovered vulnerabilities in software packages or custom code that is running on the equipment. In those situations, it may be necessary to patch that software to prevent adverse control of the DER equipment.

However, poorly implemented or uncoordinated patch management may lead to unforeseen challenges. To secure the software supply chain, code signing or equivalent mechanisms which identify the source of the patch and confirm the integrity of the data are critical. Additionally, if the patch changes the operations of the equipment, a well-intended update may lead to incompatibility between DER and control system software, false positives with antivirus or anti-malware, or degradation of DER or power system performance, reliability, or operability. There are also many technical and policy challenges associated with patching DER devices. Issues include ensuring privacy, poor incentives to distribute software updates, questions about who is authorized to update devices, and contracting language or interconnection standard requirements for opt-in/opt-out automated updates.

Fortunately, there are many standards and guides for patch management in the literature and an active community researching solutions for Industrial Control Systems (ICS) and Internet of Things (IoT) firmware updates. In this report, we investigate the application of these requirements and recommendations within DER environments, elaborate on the technologies underpinning the requirements, and provide a suggested guidance for practitioners. This report concludes with a collection of suggested requirements for DER Product Suppliers, Integrators, Aggregators, and Owners.

[1] NIST SP 800-44 Version 2, Guidelines on Securing Public Web Servers, Sept 2007.
[2] ISO/IEC 19770-2, Information technology — IT asset management — Part 2: Software identification tag, 2015.

2. PRIOR WORK

There is a vibrant community of researchers working on discovering firmware update issues and proposing improvements to software update processes. There also are many national and international patching standards and guides that have been developed in the last 10 years for power system and internet of things devices. The research and standardization literature includes the vulnerability life cycle model, code signing approaches, device fingerprinting, distributed technologies, equipment requirements, hot patching, rollback mechanisms, certificate authority requirements, and patching metadata formatting requirements. In this section, patching research, standards, and guidance are investigated to better describe the state-of-the-art and identify best practices that may be applied to the DER environment.

2.1. Patching R&D

Historically, there have been many issues with patching equipment, especially with constrained devices with limited computing power, embedded operating systems (OSs), or no OSs. Some critical security considerations fall within the following categories:[3]

- **Unauthorized device**: an illicit device retrieves an authentic copy of the firmware, allowing reverse engineering or modification.
- **Reverse engineering**: attacker analyzes the update to get sensitive information.
  - Researchers demonstrated the ability to reverse engineer unencrypted firmware from a Withings Activité fitness tracker, create a new image, update the checksum, and inject code into an Apple iOS application that pushed the code to the product via Bluetooth.[4] There were no authenticity checks in the firmware update process.
- **Firmware modification**: attacker injects extra code in the update.
  - In a well-publicized attack, the build system at SolarWinds was compromised and malicious remote access trojan code was signed before being pushed out to thousands of organizations.[5]
  - Kaseya VSA, a cloud-based IT management and remote monitoring tool, had their server platform exploited which allowed attackers to deliver ransomware via the supply-chain to their customers.[6]
  - A hacker found a means to update MacBook battery microcontroller firmware settings to change the charging behaviors.[7] By adjusting the internal voltage, current, or temperature parameters of the microcontroller, the attacker could cause overcharge of the Li-Ion cells.[8]
- **Obtaining authorization**: attacker gains access to fielded equipment through external authorization mechanisms.

[3] H. Mansor, K. Markantonakis, R. N. Akram, and K. Mayes, "Don't Brick Your Car: Firmware Confidentiality and Rollback for Vehicles," in 10th IEEE International Conference on Availability, Reliability and Security (ARES). pp. 139–148, 2015.

[4] J. Rieck, "Attacks on Fitness Trackers Revisited: A Case-Study of Unfit Firmware Security," Sicherheit, in Lecture Notes in Informatics (LNI), Gesellschaft fur Informatik, Bonn, 2016.

[5] J. Williams, What You Need to Know About the SolarWinds Supply-Chain Attack, 15 Dec 2020.

[6] M. Loman, S. Gallagher, A. Ajjan, "Independence Day: REvil uses supply chain exploit to attack hundreds of businesses," Sophos News, July 4, 2021, URL: https://news.sophos.com/en-us/2021/07/04/independence-day-revil-uses-supply-chain-exploit-to-attack-hundreds-of-businesses

[7] C. Miller, "Battery Firmware Hacking," DEF CON 19, 2011.

[8] C. Foresman, "How a security researcher discovered the Apple battery 'hack'," ARS Technica, 25 July 2011.

    - At least one malicious Android app has masqueraded as legitimate system update to deploy remote access trojans.[9]
- **Installing unauthorized firmware**: a malicious firmware payload is installed giving attacker full access to the equipment.
    - Malicious firmware updates were used to subvert a DHCP service at startup to create a netcat reverse shell and gain root access on a smart plug.[10]
    - Cui *et al*. demonstrated a remote firmware flaw in MIPS- and ARM-based printers, including HP LaserJet devices.[11] With their malicious, wormable firmware delivered to the products, the researchers could perform network reconnaissance and data exfiltration.
- **Vulnerable firmware update software**: attacker can gain access to systems by exploiting firmware update code.
    - The Broadband Forum Technical Report 069 (TR-069) defines CPE WAN Management Protocol (CWMP) which provides support functions for software or firmware image management via Auto Configuration Servers. These services exist on millions of internet-facing routers. Researchers discovered buffer overflow vulnerabilities and server-side attacks which gave them remote code execution and complete control of entire fleets of devices on consumer premises.[12,13,14,15]

Additionally, there have been multiple examples of code signing failures. A smart home hub company, Wink Inc., designed a product to control home appliances (lights, thermostat, garage door, etc.). At one point, the company had to recall products because of an expired certificate.[16] Wink also reportedly pushed a software update that included an expired certificate, which disconnected all Wink hubs from their servers, rendering the products unusable.[17] Expiration of certificates is common: Logitech allowed Harmony Link certification to expire as a means of ending the product's life[18] and Microsoft Azure's SSL certificate expired in 2013.[19]) Another Wink Inc. release caused new issues with previously functioning ZigBee devices, groups, and routes.[20]

Corrupted firmware or drive images have also been distributed, which highlights the need to authenticate software prior to field deployment. Notably, some products have been shipped with malware. Chen provides a number of examples,[21] including: Apple iPod music/video players pre-installed with a Windows virus called RavMonE.exe; TomTom GPSs shipped with Windows malware; password stealing trojans distributed on McDonald's-distributed MP3 players; Seagate shipped hard

[9] L. Ropek, "Dangerous Android App Pretends to Be a System Update to Steal Your Data," Gizmodo, 29 Mar 2021.
[10] Z. Ling, J. Luo, Y. Xu, C. Gao, K. Wu, and X. Fu, "Security Vulnerabilities of Internet of Things: A Case Study of the Smart Plug System," IEEE Internet Things J., vol. 4, no. 6, pp. 1899–1909, 2017.
[11] A. Cui, M. Costello, and S. J. Stolfo, "When Firmware Modifications Attack: A Case Study of Embedded Exploitation," NDSS Symposium, 2013.
[12] S. Tal, "I Hunt TR-069 Admins: Pwning ISPs Like a Boss," DEF CON 22, 2014.
[13] L. Oppenheim and S. Tal, " Too Many Cooks - Exploiting the Internet-of-TR-069-Things," 31C3, December 2014.
[14] D. Martyn, "A look at TR-06FAIL and other CPE Configuration Management Disasters," Still Hacking Anyway (SHA2017), Aug. 6, 2017.
[15] S. Tal and L. Oppenheim, "Come to the Dark Side - We Have (Misfortune) Cookies," ShmooCon XI, 2015.
[16] M. Korolov, "Expired certificates cost businesses $15 million per outage," CSO, 30 Sept. 2015.
[17] R. Lawler, "Wink smart home hubs knocked out by security certificate (update)," Engadget, 19 Apr. 2015.
[18] C. Welch, "Logitech will brick its Harmony Link hub for all owners in March," The Verge, 8 Nov. 2017.
[19] S. Martin, "Windows Azure Service Disruption from Expired Certificate," Microsoft Azure, 24 Feb. 2013.
[20] "Update from Wink engineering team," Reddit, Accessed 6/3/21, URL: https://www.reddit.com/r/winkhub/comments/5fxtzr/update_from_wink_engineering_team/
[21] K. Chen, Reversing and exploiting an Apple firmware update, Black Hat, 2009.

drives preinstalled with Windows trojans that disabled anti-virus and stole user credentials; Insignia digital picture frames sold by Best Buy with a Windows virus; and Hewlett-Packard sold infected USB keys with its ProLiant servers.

Researchers are seeking solutions to these challenges with novel software and hardware technologies. Patching research and technologies are described in the following sections which may be helpful in advising the DER industry.

2.1.1. *Software Vulnerability Disclosure Policies*

Exposing vulnerabilities accelerates development of solutions but also exposes systems to more attackers. In ethical hacking environments, vulnerabilities are publicly disclosed after providing the responsible organization a grace period (e.g., 30-120 days) to develop a patch for the bug.[22] This is called "responsible disclosure" or "coordinated vulnerability disclosure". Some research has been conducted to determine the optimal time to disclose software vulnerabilities based on the social benefit (minimized when the patch release coincides with the disclosure) and the number of users who employ the patch (based on the quality/protection period of the patch).[23] Based on their analytical model, the researchers found vendors will release a patch later than the social optimum unless threatened with disclosure, so public disclosures are a viable approach to reducing risk.

Other researchers from Idaho National Laboratories (INL) have been quantifying the risk presented from vulnerabilities. They defined an average active vulnerabilities (AAV) and vulnerability free days (VFD) to capture the periods of time in Figure 1 between points (a)-(c) outside of (a)-(g), respectively.[24]

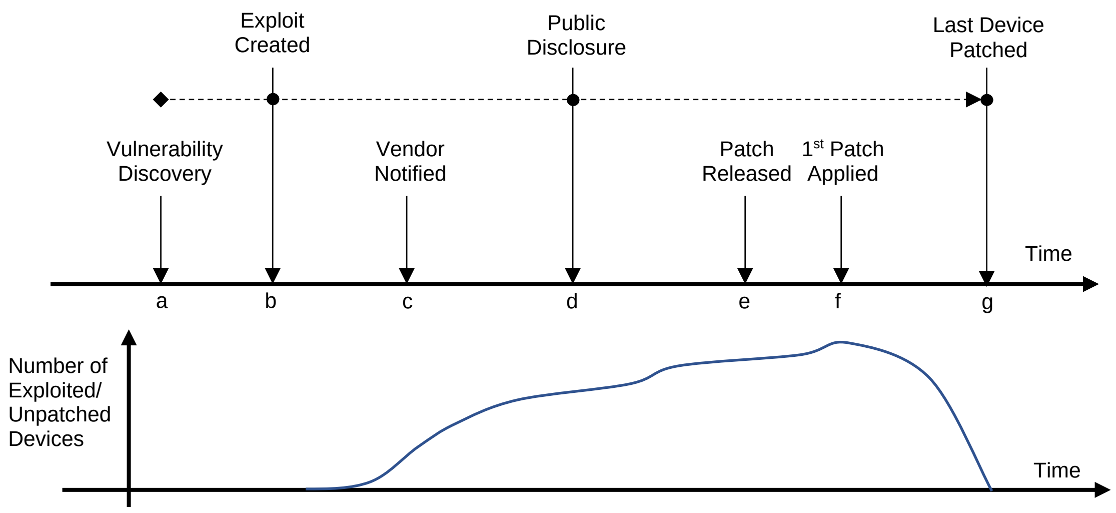


Figure 1: Vulnerability lifetime model. Adapted from [23].

Qualys, Inc. investigated the data from 104 million vulnerability scans in 2008 to develop their laws of vulnerabilities[25] that included the following findings:

[22] Zero Day Initiative, "Disclosure Policy," URL: https://www.zerodayinitiative.com/advisories/disclosure_policy/
[23] A. Arora, R. Telang, and H. Xu, "Optimal Policy for Software Vulnerability Disclosure," 2005. http://dx.doi.org/10.2139/ssrn.669023
[24] J.L. Wright, M. McQueen, L. Wellman, "Analyses of Two End-User Software Vulnerability Exposure Metrics, 7th International Conference on Availability, Reliability, and Security, August 2012.
[25] W. Kandek, "The laws of vulnerabilities 2.0," in BlackHat, Las Vegas, NV, USA, July 2009.

- Half-life: time required to reduce the vulnerability occurrence by 50%. On average this was 30 days.
- Persistence: the total lifespan of vulnerability remains virtually unlimited and not all devices will be patched. They found that 5-10% of devices never receive the update.
- Exploitation: time between exploit announcement and first attack was 10 days (1/6th the time compared to 2004).

These approaches to determining optimal disclosure times and quantifying their impact on risk metrics could potentially be used in the future to set DER vulnerability disclosure policies. Future research for the DER community in these areas is recommended.

2.1.2. Code Signing

Code signing is a process for ensuring authenticity and integrity of data. Code signing uses a digital signature mechanism to verify the identity of the data source and a checksum/hash to verify the data has not been altered in transit. As described in a NIST white paper,[26] there are three primary actors in a code signing architecture:

- The *developer* of the code or data who submits the code to the signer.
- The *signer* entity that is responsible for managing the signing keys. The signer securely generates the private/public key pair and then, either:
  - provides the public key to a certification authority (CA) through a certificate signing request (CSR) to tie their identity to the public key, or
  - shares the public key using a trusted, out-of-band mechanism.

  The signer has access control mechanisms or other tools to confirm the identity and authorization of the code/data developer.
- The *verifier* that is responsible for validating the signed code signature.

An example code signing architecture is shown in Figure 2. The developer creates the code and confirms it is free of errors with an auditing mechanism. This is then sent to the signing tool which requires the developer authenticate themselves to the signer. Once authenticated, the signing framework confirms the developer has permissions to sign this type of project and then signs the data using the private key of the signer (the software vendor in the figure). This key is securely stored using a secure, tamper-proof, cryptographic hardware device, like a hardware security module (HSM). Note there will likely be many private keys held in the DER HSM used for development, production, and individual keys for each software component/project.

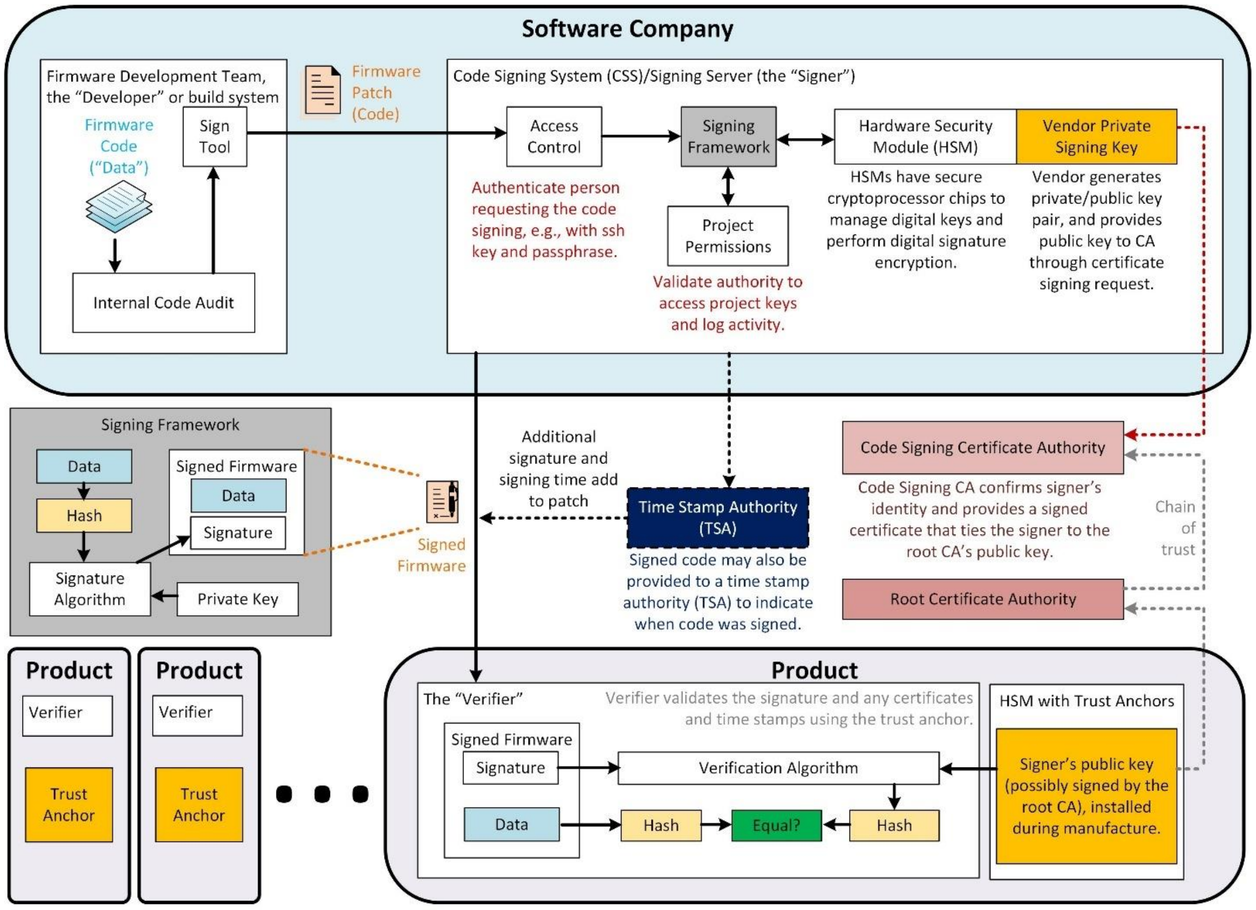
Optionally, if it is desired to be able to confirm the data source (identify who controlled the code at the time it was signed), the signer's public key may be tied back to a trusted root CA. This is done with a code signing request (CSR), wherein the code signing CA ensures the public key is associated with the signer. The signer's public key (which has been signed by the root CA private key or one of its intermediate CA proxies) is then installed in each of the products to verify the authenticity of the signed firmware data. The root CA is typically not used as the certificate signing CA. That way if there is a compromise of the private key of either the signer or the code signing CA, the system can recover by revoking the certificates associated with the lost private key.

The signing process first hashes the data and then runs this hash through a signature algorithm to create the signature that is packaged with the data to create the signed data/firmware. It is also possible to use a Time Stamp Authority (TSA) to add an additional digital signature with the signing

[26] D. Cooper, et al., "Security Considerations for Code Signing," NIST Cybersecurity White Paper, Jan. 26, 2018.

time to the package. The advantage of using a TSA is that the verifier can determine the validity of code based on the timestamp if the signing key has expired or been compromised/revoked.

Once the data reaches the product for installation, the signature algorithm uses the public key of the manufacturer, pre-installed on the device, to verify the validity of the data. To do this, it generates the hash of the data and compares that to the decrypted signature. If they match, the authenticity of the data is confirmed. The digital signature ensures the public key in your name is indeed your key, thereby providing authenticity and non-repudiation.



Software Company
Firmware Development Team, the "Developer" or build system
Firmware Code ("Data")
Sign Tool
Internal Code Audit
Firmware Patch (Code)
Code Signing System (CSS)/Signing Server (the "Signer")
Access Control
Authenticate person requesting the code signing, e.g., with ssh key and passphrase.
Signing Framework
Project Permissions
Validate authority to access project keys and log activity.
Hardware Security Module (HSM)
HSMs have secure cryptoprocessor chips to manage digital keys and perform digital signature encryption.
Vendor Private Signing Key
Vendor generates private/public key pair, and provides public key to CA through certificate signing request.
Signing Framework
Data
Hash
Signature Algorithm
Private Key
Signed Firmware
Data
Signature
Signed Firmware
Additional signature and signing time add to patch
Time Stamp Authority (TSA)
Signed code may also be provided to a time stamp authority (TSA) to indicate when code was signed.
Code Signing Certificate Authority
Code Signing CA confirms signer's identity and provides a signed certificate that ties the signer to the root CA's public key.
Chain of trust
Root Certificate Authority
Product
Verifier
Trust Anchor
Product
Verifier
Trust Anchor
Product
The "Verifier"
Verifier validates the signature and any certificates and time stamps using the trust anchor.
Signed Firmware
Signature
Data
Verification Algorithm
Hash
Equal?
Hash
HSM with Trust Anchors
Signer's public key (possibly signed by the root CA), installed during manufacture.

Figure 2: Example Code Signing Architecture.

2.1.2.1. Public Key Infrastructure (PKI) Code Signing

In 1978, MIT researchers Rivest, Shamir, and Adleman, published their approach to using a private key to sign a message which could be verified using the corresponding public key.[27] PKI code signing with digital signatures differ from other cryptographic primitives like hashes, message authentication code (MAC), and hash-based message authentication code (HMAC) in both the underlying technologies and the security features they provide. A comparison of multiple security approaches to security principles is provided in Table 1. The ability to provide non-repudiation is one reason why

[27] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems." Commun. ACM 21, pp. 120-126, 1978.

there has been substantial research on code signing using PKI and why it is recommended for DER firmware updates.

Table 1: Comparison of different cryptographic primitives.

	Hash (no key)	**MAC/HMAC** (symmetric keys)	**Digital Signature** (asymmetric keys)
Integrity	Yes	Yes	Yes
Authentication	No	Yes	Yes
Non-repudiation	No	No	Yes

2.1.2.2. Hash-Based Signatures for Data Integrity

It is important to have a secure hashing algorithm for code signing because the hash determines the legitimacy of the code—even in the case of using PKI digital signatures. Code could be maliciously changed if the hash can be falsified. For example, hash collisions were used in Stuxnet and Flame malware.[28] There are many hashing functions including cyclic redundancy checks (CRCs), checksums, MD5, SHA-1, SHA-256, NTLM, etc. These take an arbitrary length input and create a fixed length output which can be computed quickly and efficiently. A good cryptographic hash has several key security features that prevent forgeability:

- Preimage resistance – The algorithm is a one-way function, H, such that given a hash, h, of data, d, it is difficult to find a d that satisfies, $H(d) = h$.
- Second preimage resistance – Given some d, it is difficult to find different data which produces the same hash, $H(d) = H(d')$.
- Collision resistance – It is difficult to find any d, d', such that $H(d) = H(d')$. This is like second preimage resistance, except the attacker may select any d and d'.

The history of hashing for the purpose of verifying the integrity of data began in 1979 when Leslie Lamport developed the Lamport one-time signature algorithm, which used a one-way function to verify the authenticity of a message.[29] The major limitation of this approach was the private keys could only be used a single time. Fortunately, Ralph Merkle developed and later patented a "tree signature" which efficiently verified the contents of N messages by creating a hash tree where non-leaf nodes are labeled with the hash of its child nodes.[30,31] The tree head hash acted as the public master key and each of the leaves were Lamport public keys. To verify a message, the signature was verified to be one of the leaf Lambert public keys using the Merkle tree. So long as the root hash was provided by a trusted source, the Merkle tree allowed all subsequent internal and leaf nodes to be verified to be free of damage or falsification.

Merkle trees are widely used today for many security applications like blockchains. The Merkle Tree Signature (MTS) algorithm is one such application seeing renewed interest. It uses a secure one-way hash function to verify digital signatures,[32] requires low computational power, and is thought to be

[28] Roel Schouwenberg, "Stuxnet and Flame – burning ring of fire" Kaspersky Lab Presentation.
[29] Constructing Digital Signature from a One Way Function, 18 Oct, 1979.
[30] R. Merkle, "Method of providing digital signatures," US Patent 4309569, published Jan 5, 1982.
[31] R. Merkle, "A Certified Digital Signature." CRYPTO (1989).
[32] R. Housley, "Use of the Hash-based Merkle Tree Signature (MTS) Algorithm in the Cryptographic Message Syntax (CMS)," 21 September 2016.

secure to quantum computing threats. The latter feature makes it a good choice for the DER industry as equipment will be in the field in the post-quantum world. One weakness with Merkle trees is that the depth of the tree is not indicated by the hashes, so there is the possibility of a hash collision in the chain by removing intermediate nodes. To prevent second preimage attacks on Merkle trees, Certificate Transparency from IETF RFC 6962 can be used, whereby specific bytes are prepended to leaf and internal nodes to indicate the depth of the chain.[33]

2.1.3. Alternative Firmware Update Options

2.1.3.1. Fingerprinting with Physical Unclonable Functions

In traditional code signing, the identity of the data signer is tied to a PKI ecosystem. However, there are proposed alternatives. Physical Unclonable Functions (PUFs) coined by Gassend et al. in 2001,[34] refer to one-way functions that generate cryptographic keys or provide key storage using features in silicon circuitry or other hardware elements that have random physical attributes generated during manufacturing—the machine equivalent of using biometric data to authenticate humans.[35] Many physical characteristics of PUF devices are included in a mathematical mapping between challenge and response to ensure it is unclonable, such as random delays in silicon gates for ring oscillator PUFs[36] or setting up race conditions in an arbiter PUF.[37] A well-designed PUF provides uniqueness between manufactured devices, unpredictability in the response to a challenge, and reliability in the result. The latter is difficult with current PUF technology because the designer must guarantee a reliable result over the lifetime of the product and with different conditions (temperature, input voltage, etc.)—something that has led to error-correcting, post-processing technologies.[38] There are two primary uses for PUFs:

- A "strong" PUF which supports many (ideally exponential) challenge-response pairs (CRPs) and can provide authentication without auxiliary cryptographic hardware. By using a challenge-response authentication mechanism, the PUF uniquely identifies itself based on the physical attributes of the local hardware.[39,40,41] Since the response to a given challenge is represented in the hardware structure/equipment directly, the "keys" cannot be stolen and resistant to spoofing attacks.

- A "weak" PUF, which has a limited number of CRPs, but can be used as a key generator so long as the key is kept secret (and, typically, in volatile memory). For example, a weak Static

[33] Laurie, B.; Langley, A.; Kasper, E. (June 2013). "Certificate Transparency". IETF: RFC6962. doi:10.17487/rfc6962
[34] Blaise Gassend, Dwaine Clarke, Marten van Dijk and Srinivas Devadas. Silicon Physical Random Functions. Proceedings of the Computer and Communications Security Conference, November 2002.
[35] C. Herder, M. Yu, F. Koushanfar and S. Devadas, "Physical Unclonable Functions and Applications: A Tutorial," in Proceedings of the IEEE, vol. 102, no. 8, pp. 1126-1141, Aug. 2014, doi: 10.1109/JPROC.2014.2320516.
[36] G. E. Suh and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," 2007 44th ACM/IEEE Design Automation Conference, 2007, pp. 9-14.
[37] S. Devadas, E. Suh, S. Paral, R. Sowell, T. Ziola, V. Khandelwal, "Design and Implementation of PUF-Based "Unclonable" RFID ICs for Anti-Counterfeiting and Security Applications, 2008 IEEE International Conference on RFID, Las Vegas, NV, April 16-17, 2008.
[38] C. Boehm, Physical Unclonable Functions in Theory and Practice. Springer, 2012.
[39] McGrath, Thomas; Bagci, Ibrahim E.; Wang, Zhiming M.; Roedig, Utz; Young, Robert J. (2019). "A PUF taxonomy". Applied Physics Reviews. 6 (11303): 011303.
[40] J. Guajardo, S. Kumar, G.-J. Schrijen, and P. Tuyls. FPGA intrinsic PUFs and their use for IP protection. In Cryptographic Hardware and Embedded Systems - CHES 2007, volume 4727 of Lecture Notes in Computer Science, pages 63–80. Springer Berlin / Heidelberg, 2007.
[41] R. S. Pappu. Physical One Way Functions. PhD thesis, Massachusetts Institute of Technology, 2001.

Random-Access Memory (SRAM) PUF can physically obfuscate a private key; on boot-up the SRAM has a random but repeatable pattern of 0 or 1 values in the transistors on the chip based on voltage mismatch during manufacturing. This technique, for example, could be used to better secure the private key of the code signer.

Typically, PUFs are used to uniquely identify edge-devices like DER (client authentication) by generating a list of challenge-response pairs (CRPs) during manufacture that are saved on a server. However, in the case of DER patching, it is the identity of the signing server that must be authenticated; and embedding a full list of CRPs on each fielded device is an intractable problem because the list of CRPs must be stored and secured on all the constrained DER devices and CRPs cannot be reused or a man-in-the-middle attack could inject malicious firmware. Some researchers have proposed online approaches around this issue by using strong PUF CRPs and employing fuzzy and reverse fuzzy extractors, random number generators, hash functions, and message authentication codes in the fielded device.[42] However, other theoretical work, has found indications that PUFs are insufficient for offline software protection due to virtualization attacks.[43] So, if taking the PUF approach, practitioners must be careful to select the right commercial PUF implementations from those on the market and appropriately use PUFs to authenticate DER firmware updates.

2.1.3.2. Blockchain-Based Secure Firmware Updates

Blockchain-based solutions for securing firmware updates have been proposed in the literature. In general, these blockchain approaches are designed to replace the PKI code-signing ecosystem as the authentication mechanism. Lee and Lee proposed a peer-to-peer blockchain network that would validate the firmware version and provide data integrity using verification nodes.[44] In another approach, the blockchain records the OEM and firmware data, and the smart contract points to Message Queuing Telemetry Transport (MQTT) servers on blockchain nodes to distribute firmware updates.[45] Yohan and Lo devised a secure peer-to-peer blockchain verification and distribution mechanism that used mutual authentication.[46] Their Firmware-Over-the-Blockchain (FOTB) framework provided integrity and authenticity by only initiating a firmware update once a patch smart contract has successfully passed the peer-to-peer verification process blockchain consensus mechanism.

2.1.4. Bootloader Rollback Mechanisms

Firmware update security must be provided at the device level as well. Firmware is stored in flash or Electrically Erasable Programmable Read-Only Memory (EEPROM). In the event of a failure (like power disruption or incompatible hardware) in the update process, there must be a rollback mechanism to return the device to an operational state. Using a representative example of automotive

[42] W. Feng, Y. Qin, S. Zhao, Z. Liu, X. Chu, D. Feng, "Secure Code Updates for Smart Embedded Devices Based on PUFs." In: Capkun S., Chow S. (eds) Cryptology and Network Security. CANS 2017. Lecture Notes in Computer Science, vol 11261. Springer, Cham., 2018.

[43] R. Nithyanand, J. Solis, "A Theoretical Analysis: Physical Unclonable Functions and The Software Protection Problem," SAND2011-6673, 2011.

[44] B. Lee, J.H. Lee, "Blockchain-based secure firmware update for embedded devices in an Internet of Things environment," J Supercomput 73, 1152–1167 (2017).

[45] M. -H. Tsai, Y. -C. Hsu and N. -W. Lo, "An Efficient Blockchain-based Firmware Update Framework for IoT Environment," 2020 15th Asia Joint Conference on Information Security (AsiaJCIS), pp. 121-127, 2020.

[46] A. Yohan, N.W. Lo, "FOTB: a secure blockchain-based firmware update framework for IoT environment." Int. J. Inf. Secur. 19, pp. 257–278, 2020.

Electronic Control Units (ECUs),[47] a bootmanager/bootloader on the device selects to load into flashloader or application mode. Application mode is used for normal operations of the device. If flashloader mode is selected and there is a request for reprogramming, the flashloader loads a flashdriver from additional flash memory or Random Access Memory (RAM), which has instructions for erasing and writing data/program flash memory. If the process fails, a firmware recovery procedure must exist. This requires a copy of the old firmware that has been previously saved or a method to extract and save the current firmware in memory before the update process. Upon failure, the bootmanager/bootloader reverts to the previous working version of the firmware stored in flash.

Additionally, it is good practice to use a secure bootloader that includes a firmware verifier. Upon any reboot/restart, the verifier will inspect the firmware metadata/manifest to ensure the firmware image is cryptographically legitimate. For more information on device-level security guidance, refer to the NIST guidelines for protecting system firmware from local attacks.[48]

2.1.5. Hot Patching

Hot patching or dynamic software updating (DSU) is the process of updating code without stopping operations or restarting the operating system. This capability exists in iOS, Linux, and Windows systems and would allow the DER equipment to remain operational during the update process. In addition to reducing downtime, hot patching reduces the impact on the power system, reduces the chances of hardware reboot failure, and provides near-continuous visibility into the operations of equipment. There are many examples of hot patching tools on the market after its theoretical demonstration of Dynamic Modification System (DYMOS) in 1983 by Insup Lee.[49] For instance, released in 2014-2015, kpatch and kGraph were added to the Linux kernel source code. They swap out functions wholesale for their patched versions without stopping running processes. In a recent DOE project, researchers migrated, updated, and restored ICS software using containerized applications, like Docker.[50]

2.1.6. Software Bill of Materials

Modern DER devices include many open-source and/or proprietary packages, libraries, and binaries. These software elements provide a range of functionality; however, they also provide new attack vectors if they are not effectively patched. For example, there have been multiple discoveries of low-level vulnerabilities that have affected millions of devices:

- Urgent/11 was a collection of 11 TCP/IP stack zero-day vulnerabilities in VxWorks—affecting many Real-Time Operating Systems (RTOS) used in over 2 billion industrial, medical and enterprise devices.[51]
- Ripple20 was a collection of low-level TCP/IP vulnerabilities that allowed Remote Code Execution.[52,53]

[47] H. Mansor, K. Markantonakis, R. N. Akram and K. Mayes, "Don't Brick Your Car: Firmware Confidentiality and Rollback for Vehicles," 2015 10th International Conference on Availability, Reliability and Security, pp. 139-148, 2015.
[48] A. Regenscheid, "NIST Special Publication 800-193, Platform Firmware Resiliency Guidelines," 2018.
[49] I. Lee, "DYMOS: A Dynamic Modification System," Doctor of Philosophy in Computer Science thesis, University of Wisconsin – Madison, 1983.
[50] A. Chavez, "Containerized Application Security for Industrial Control Systems," DOE CEDS Peer Review, 6-8 Nov. 2018.
[51] B. Seri, G. Vishnepolshy, D. Zusman, "Urgent/11: Critical vulnerabilities to remotely compromise VxWorks, the most popular RTOS", BlackHat USA, Las Vegas, 3-8 Aug 2019.
[52] M. Kol, S. Oberman, "Ripple20: CVE-2020-11896 RCE, CVE-2020-11898 Info Leak" JSOF Whitepaper, June 2020.
[53] M. Kol, A. Schon, S. Oberman, "Ripple20: CVE-2020-11901" JSOF Whitepaper, August 2020.

- CDPwn affected the Data Link Layer of the Cisco Discovery Protocol (CDP) that enabled Remote Code Execution and Denial of Service.[54,55]
- QNX Real Time Operating System (RTOS) was affected by a BadAlloc vulnerability which allows arbitrary code execution. This affected many products including BlackBerry phones.[56]

To help address this concern, there has been a recent push to provide greater transparency in the supply chain using Software Bills of Materials (SBOMs) for vendor and asset owner vulnerability tracking.

Following Executive Order 14028 on Improving the Nation's Cybersecurity, the Department of Commerce, National Telecommunications and Information Administration (NTIA) published *The Minimum Elements for a Software Bill of Materials (SBOM)*[57]. In this document, they defined the broad, inter-related elements as:

- **Data fields**, e.g.,
    - Supplier name
    - Component name
    - Version of the component
    - Cryptograph hash of the component
    - Any other unique identifier
    - Dependency relationship
    - Author of the SBOM data
- **Operational considerations** that represent operational/business decisions and actions that define the practice of requesting, generating, sharing, and consuming SBOMs, e.g.,
    - **Frequency** of when and where SBOM data is generated and tracked.
    - **Depth** of dependencies, dependencies of dependencies, etc.
    - **Delivery** – (a) the existence and timely availability of the SBOM and (b) mechanisms to retrieve or transmit the SBOM with appropriate access control.
- **Automation support** to enable scalability across organizational boundaries with automated generation and machine-readable formats, such as:
    - Software Package Data eXchange (SPDX)[58]
    - CycloneDX[59]
    - Software Identification (SWID) Tags[60][61] format

Adoption of this best practice would be helpful for the DER community to track software supply chain vulnerabilities and support patching processes.

[54] B. Hadad, B. Seri, Y. Sarel, "CDPwn: Breaking the Discovery Protocols of the Enterprise of Things," Armis, Inc. Whitepaper, 2020.

[55] CMU Software Engineering Institute CERT Coordination Center, Cisco Discovery Protocol (CDP) enabled devices are vulnerable to denial-of-service and remote code execution, Vulnerability Note VU#261385. 2020-02-05, URL: https://kb.cert.org/vuls/id/261385/

[56] US-CERT, "Alert (AA21-229A): BadAlloc Vulnerability Affecting BlackBerry QNX RTOS," Aug 17, 2021, URL: https://us-cert.cisa.gov/ncas/alerts/aa21-229a, visited Sept 9, 2021

[57] US Department of Commerce, The Minimum Elements for a Software Bill of Materials (SBOM), July 12, 2021.

[58] SPDX, URL: https://spdx.dev/, visited August 13, 2021.

[59] CycloneDX, https://cyclonedx.org/, visited August 13, 2021.

[60] ISO/IEC 19770–2:2015, "Information technology - Software asset management - Part 2: Software identification tag," 2015.

[61] D. Waltermire, B.A. Cheikes, L. Feldman, G. Witte, Guidelines for the Creation of Interoperable Software Identification (SWID) Tags," NISTIR 8060, 2016, URL: http://dx.doi.org/10.6028/NIST.IR.8060

2.2. Patching Standards and Guides

There are multiple national and international guides and standards that cover patching for power systems, industrial control systems, or internet of things devices at some level. Many indicate the necessity of patch management,[62,63,64,65,66,67] but fewer go into detailed requirements. A summary of the more comprehensive standards that the subgroup reviewed to generate the DER recommendations are provided in Table 2.

Table 2: Guides and standards reviewed by the DER Cybersecurity Subgroup.

Title	**Type**	Contents
Patch Lifecycle and Patch Management Program Requirements		
IEC TR 62443-2-3: Security for industrial automation and control systems – Patch management in the IACS environment	Standard	Provides patch management program requirements for asset owners and industrial automation and control system (IACS) product suppliers.
IEC 62443-4-1: Security for industrial automation and control systems – Secure product development lifecycle requirements	Standard	Product development life-cycle requirements related to cybersecurity for IACS products.
Power System or ICS Patching Requirements		
NERC CIP 007-006: Systems Security Management Requirements	Guideline	Requires patch management process for bulk power equipment with a 35-day evaluation schedule.
IEEE C37.231-2006: IEEE Recommended Practice for Microprocessor-Based Protection Equipment Firmware Control	Recommended Practice	Detailed recommendations for administration of firmware revisions for protection equipment.
DHS Recommended Practice for Patch Management of Control Systems	Recommended Practice	DHS ICS patch management program elements and patching analysis guidance.
Internet-of-Things Patching Requirements		
ETSI EN 303 645: Cyber Security for Consumer Internet of Things: Baseline Requirements	Standard	Provides multiple requirements for IoT security updates in Section 5.3.
Communication and Firmware-over-the-Air Requirements		
IEEE 2030.5-2018: IEEE Standard for Smart Energy Profile Application Protocol	Standard	This communication protocol includes a mechanism for downloading patch data.
IETF RFC 4108 Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages	Standard	Highly technical details covering hardware requirements, firmware package protections, and signatures elements for firmware updates.
IETF RFC 9019: Firmware Update Architecture for Internet of Things	Informational	IoT firmware update background and a standardized manifest format for describing and protecting patches

[62] NIST SP 800-53 Revision 5, "Security and Privacy Controls for Information Systems and Organizations," 2020.
[63] NIST SP 800-82 Revision 2, "Guide to Industrial Control Systems (ICS) Security," 2015.
[64] UL 2900-1, "Software Cybersecurity for Network-Connectable Products, Part 1: General Requirements," 2017.
[65] Federal Office for Information Security, BSI-CS 106 Version 1.00, "Security Requirements for mechanical engineers and integrators" 2014.
[66] IEEE 1686, "Standard for Intelligent Electronic Devices Cyber Security Capabilities," 2013.
[67] ISO/IEC 27019, "Information security controls for the energy utility industry," 2017.

Additionally, the Cloud Security Industry Summit (CSIS) Supply Chain Workgroup wrote a Secure Firmware Development Best Practices[68] document which focused on firmware used in large scale data centers but also provides excellent recommendations for firmware development, testing, debugging, and source control in general.

The major requirement/recommendation themes are detailed in the following subsections.

2.2.1. Patching Lifecycle

IEC TR 62443-2-3:2015 provides a detailed patch state model that indicates the stages of a patch and the stakeholder who manages these states. The flow between these states is shown in Figure 3, where the states are defined as follows:

- *Available*: The patch has been provided by a supplier or third party but has not been tested.
- *In Test*: The patch is being tested by the product supplier.
- *Not Approved*: The patch has failed the testing and should not be used.
- *Not Applicable*: The patch has been tested and is not relevant.
- *Approved*: The patch has passed testing by the product supplier.
- *Released*: The patch is released for use or is directly applicable by the asset owner for their internal development/test systems.
- *In Internal Test*: The patch is being tested by the asset owner testing team.
- *Not Authorized*: The patch has failed internal testing or is not applicable.
- *Authorized*: The patch is released by the asset owner and meets company standards for updateable devices or did not need testing.
- *Effective*: The patch is posted by the asset owner for use.
- *Installed*: The patch is installed on the system.

Defining patching states is helpful for the DER ecosystem so there is no confusion within the industry about where in the process a patch is currently located. IEC 62443-4-1 provides additional product lifecycle requirements for industrial systems that fall within the *In Internal Test* state. In the Security Update Management (SUM) section, they include a range of requirements for creating processes for:

- Verifying security updates to address the intended security vulnerabilities
- Confirming the update is not contradicting other operational, safety, or legal constraints
- Ensuring the patch does not adversely affect operation of the product

As a result, there is a significant responsibility placed on the asset owner to confirm the patch is effective prior to installation. This is a reasonable expectation for grid operators and aggregators, but in the case of residential systems, it is anticipated that the asset owner states would be bypassed.

[68] Supply Chain Technical Working Group, "Secure Firmware Development Best Practices," Cloud Security Industry Summit, July 2019.

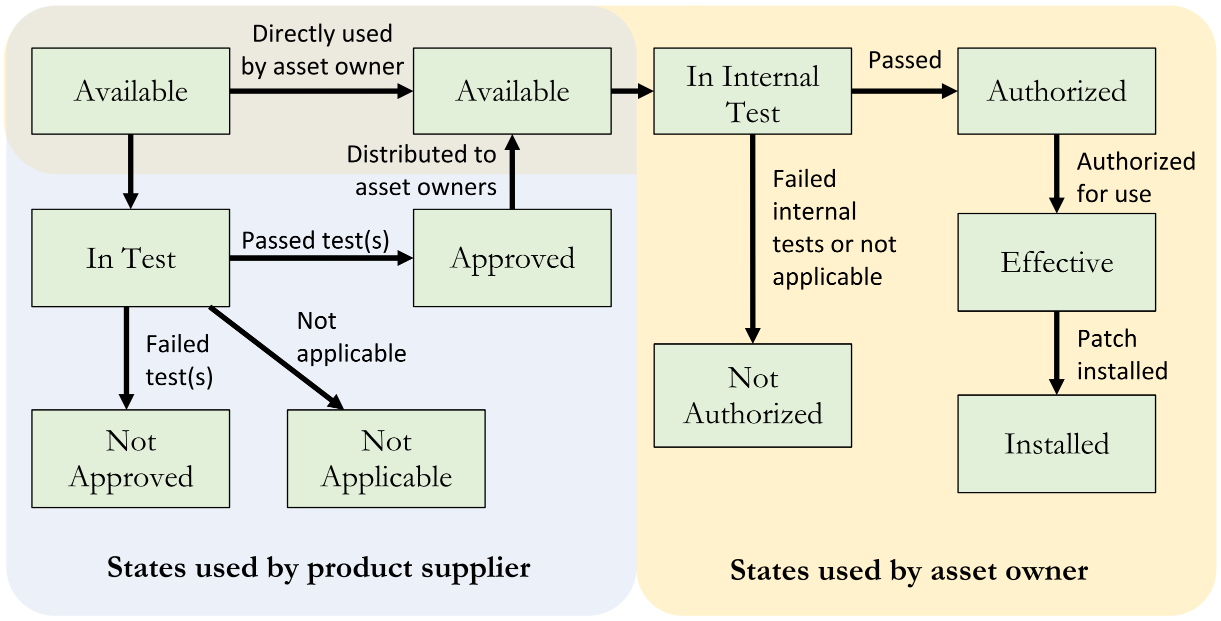


Figure 3: IEC 62443-2-3 patching states and flow for product supplier and asset owner.

2.2.2. Over-the-Air Patching Process

The mechanics for installing the patch—i.e., moving from the IEC 62443-2-3 *Effective* state to the *Installed* state—are nontrivial, as described above in the Patching R&D section. IETF RFC 9019 provides two examples for transmitting over-the-air firmware to the fielded equipment. One implementation is shown in Figure 4, wherein the firmware and manifest (described in the following section) are uploaded to the manufacturer's (or asset owner's) status tracker and firmware servers. The DER is notified of the new firmware version, pulls down the firmware and manifest, and verifies the package and stores it. Then the DER Status Tracker Client initiates a bootloader reboot and the secure boot process confirms the validity of the firmware and boots into the new firmware. Alternative approaches are certainly possible. For instance, IEEE 2030.5 includes a firmware update mechanism that could be employed for DER gateways or DER devices running that protocol.

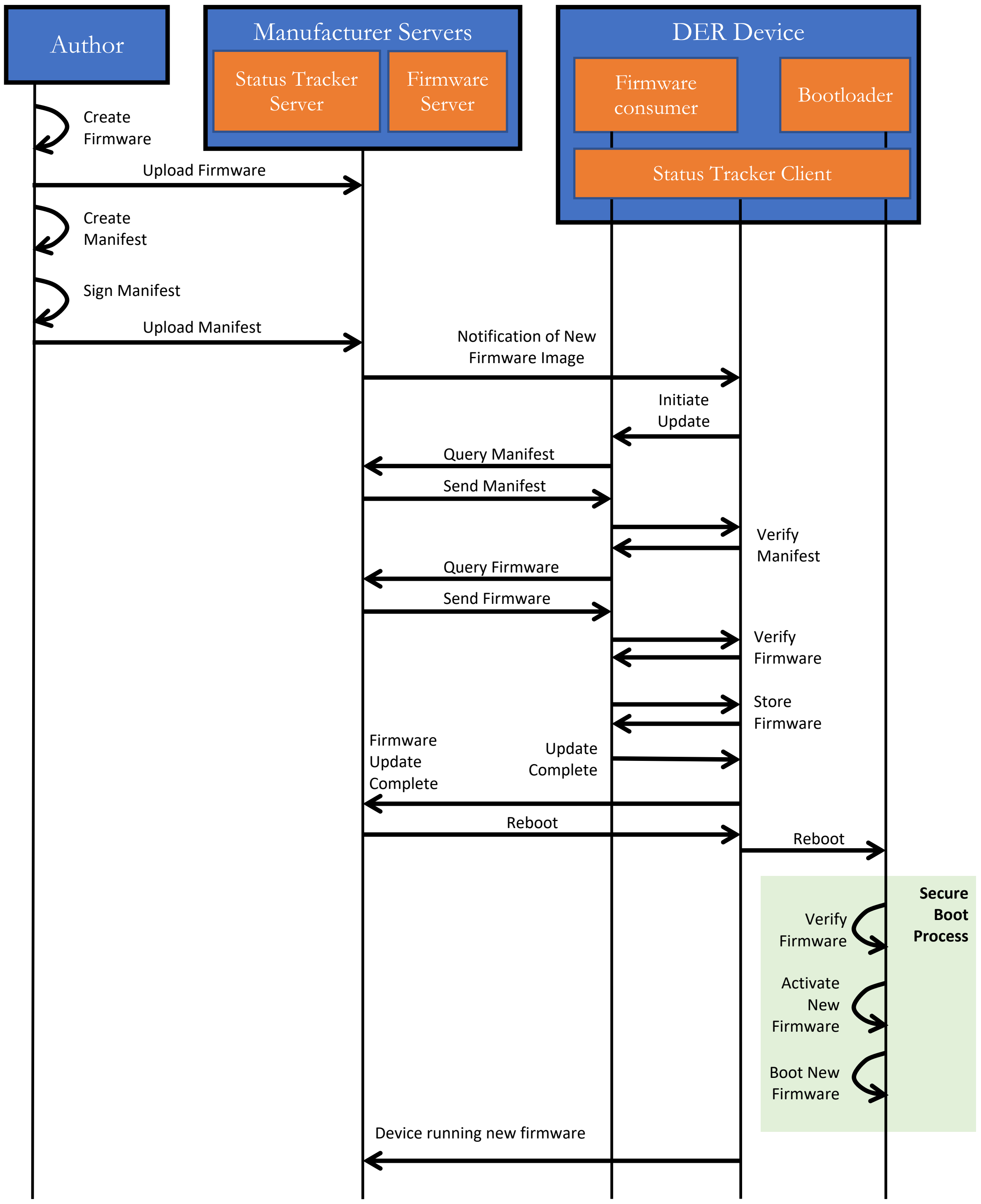


Figure 4: Firmware update process example from RCF 9019.

2.2.3. *Firmware Package Format and Manifest*

In IEC 62443-4-1, the SUM-2 "Security Update Documentation" requirement states, a process shall be employed to ensure that documentation about product security updates is made available to product users that includes but is not limited to:

a) the product version number(s) to which the security patch applies
b) instructions on how to apply approved patches manually and via an automated process
c) description of any impacts that applying the patch to the product can have, including reboot
d) instructions on how to verify that an approved patch has been applied
e) risks of not applying the patch and mediations that can be used for patches that are not approved or deployed by the asset owner.

To facilitate this information exchange from the product vendor/supplier to the asset owners, common firmware package formats/manifests have been defined. This simplifies configuration management of large collection of devices. IETF RFC 4108[69] offers one choice for a digitally signed firmware package format using Cryptographic Message Syntax (CMS). The format includes fields to determine:

- The hardware module type and revision needed to run the firmware package
- A firmware package identifier and version number
- A human-readable description of the firmware package
- Dependencies on other firmware packages, if any (Also see IEC 62443-4-1 SUM-3 "Dependent component or operating system security update documentation")
- Firmware-decryption key identifier when the firmware package is encrypted
- Cryptographic and compression algorithms implemented by the firmware package
- The identity and certificate of signer of the firmware package
- The date and time that the firmware package was signed

RFC 4108 also offers a digital signature to protect firmware packages from undetected modification and ensure authentication of the source, optional encryption to protect the firmware from disclosure; and an optional "firmware package loading receipt" to acknowledge loading the firmware package on a hardware module.

The IETF Software Updates for Internet of Things (SUIT) team has been creating a manifest information model for IoT devices.[70] It extends the RFC 4108 list to include:

- Payload format, processing steps, and storage location
- *Manifest Envelope Elements*: Delegation Chain and Signature
- Security considerations including threat descriptions derived from the STRIDE threat model[71]
- *Security Requirements* like authenticated payload type, authenticated precursor images, payload encryption, access control, etc.
- *User Stories* that provide expected use cases for device operators, developers, etc. This may include items like secure execution using manifests, decompress on load, payload in manifest, etc.
- *Usability Requirements* which satisfy the *User Stories* above. They include pre-installation checks, override remote resource location, target version matching, etc.

[69] R. Housley, "Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages." RFC 4108, 2005.
[70] B. Moran, H. Tschofenig, H. Birkholz, "A Manifest Information Model for Firmware Updates in IoT Devices," IETF Draft, 2021.
[71] Microsoft, "The STRIDE Threat Model", May 2018.

In addition to a manifest information model, the SUIT team is also actively working on drafts for a serialization format for the updates,[72] network access requirements (e.g., DHCP, IEEE 802.1X) in a Manufacturer Usage Description (MUD) specification,[73] and a firmware encryption mechanism.[74]

IEC 62443-2-3 defines a format for the patch compatibility information based on eXtensible Markup Language (XML) using an XML schema definition (XSD) file. The patch information file is identified as vendor patch compatibility (VPC). The VPC XSD file format is described in detail in the document and Appendix A. This *PatchData* object includes the vendor name, patch vendor name, product, version, release date, severity, what patches it replaces, etc.

The DER industry should investigate each of the elements in these metadata manifests to select an appropriate implementation for the industry. In the interest of harmonization, it would be preferred to use one of the standardized manifest formats that have already been created by IETF or IEC.

2.2.4. Code Signing Standards

Where possible, the DER industry should apply best practices and employ tested, standardized approaches for code signing. Two areas where the DER industry should adopt industry standards are for certificate authorities and trust anchors.

2.2.4.1. Code Signing Certificate Authority Requirements

NISTIR 7924[75], CA/Browser Forum[76], and the CA Security Council[77] provide reference certificate policies or requirements for CAs that issue code signing certificates. For example, CA/B Forum requires private keys are generated, stored, and used in a cryptographic module (i.e., HSMs) that meet or exceed the requirements of FIPS 140-2 Security Level 2.[78] This means the cryptographic module must include tamper-evidence technologies, role based authentication, and an approved algorithm or security function. Adoption of these best practices for code signing CAs for the DER industry is advised.

2.2.4.2. Code Signing Trust Anchors

IETF RFC 4108 requires the trust anchor consist of a public key signing algorithm, associated public key, and a public key identifier. The public key is used with the signature validation algorithm to validate the firmware signature by comparing the data hash with the decrypted hash. The subject public key identifier in the X.509 certificate validates the origination of the firmware package signature by confirming the private key of the organization that signed the data.

---

[72] B. Moran, et al., "A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest," IETF Draft, 2021.
[73] B. Moran, H. Tschofenig, "Strong Assertions of IoT Network Access Requirements," IETF Draft, 2021.
[74] H. Tschofenig, "Firmware Encryption with SUIT Manifests," IETF Draft, 2021.
[75] H. Booth and A. Regenscheid, NIST Internal Report (IR) 7924, Reference Certificate Policy, Second Draft, National Institute of Standards and Technology, Gaithersburg, Maryland, May 2014. https://csrc.nist.gov/publications/detail/nistir/7924/draft
[76] Guidelines for the Issuance and Management of Extended Validation Code Signing Certificates, Version 1.4, CA/Browser Forum, July 2016. https://cabforum.org/wp-content/uploads/EV-Code-Signing-v.1.4.pdf
[77] Code Signing Whitepaper, CA Security Council, December 2016. https://casecurity.org/wp-content/uploads/2016/12/CASC-Code-Signing.pdf
[78] FIPS 140-2, Security Requirements for Cryptographic Modules, 2002.

NIST FIPS 140-2 defines four levels of security for cryptographic hardware modules. FIPS 140-2 Level 3 HSMs are highly resistant to tampering and relatively inexpensive,[79] so compliance with FIPS 140-2 Level 3 could be recommended for local storage of software keys at the Signer and on the fielded products depending on the risk tolerance of the associated organization or corporation.

[79] L. Seltzer, "Securing Your Private Keys as Best Practice for Code Signing Certificates," URL: https://www.thawte.com/code-signing/whitepaper/best-practices-for-code-signing-certificates.pdf, accessed Aug 13, 2021.

3. CHALLENGES FOR DER PATCHING

There are unique challenges for DER patching because the equipment supports critical infrastructure; DER sites are not under common management and locations cannot be readily reached by support teams; and there are multiple pathways and mechanisms to get patches to DER devices (e.g., differences in residential vs utility-scale installations). In the case of larger systems, contractual agreements between the utility and commercial system owners may further complicate roles and responsibilities of the parties—which are untrusting of each other. In this section, we dive into some of the thorny issues of DER patching and the implications for the industry.

3.1. Safety and Interconnection Certification Voidance

At the distribution level and occasionally at the transmission level, DER equipment is listed to UL 1741. UL 1741 includes safety requirements as well as compliance with the US interconnection standard, IEEE 1547, and the associated IEEE 1547.1 test procedure. The listing process for DER equipment takes months, wherein the electrical performance and safety of the equipment is closely quantified through construction evaluation and testing to US consensus requirements. As part of this process, the critical inverter firmware or software controlling monitoring and protection of limits for temperature, current, voltage, arc faults, ground faults, and PV rapid shutdown functionality is evaluated for reliability in accordance with UL 1998, *Standard for Software in Programmable Components*, or UL 60730-1, *Standard for Safety Automatic Electrical Controls – Part 1: General Requirements*, which tests software safety.

In support of the IEEE 1547 requirement to test each DER product version including changes in critical hardware or software, UL 1741 Section 40 states that each combination of microprocessor model, manufacturer, and firmware/software version used in the production of a utility-interactive inverter or interconnection system equipment shall be evaluated. Given the normal certification process is so lengthy, what options do DER vendors have for pushing critical security patches to equipment? An exception: for units with firmware/software in compliance with UL 1998 or UL 60730, subsequent firmware/software revisions may be entitled to a limited revaluation as determined by the subsequent UL 1998 or UL 60730 evaluation of the revised firmware or software. Stated another way, any IEEE 1547 and UL 1741 firmware version change requires the utility-interactive inverter to undergo a full recertification unless it complies with UL 1998 or UL 60730 and is determined to require a limited re-evaluation. The scope of the re-evaluation shall be defined by the potential impact of the firmware or software revisions. For example, patches that apply to the communication module—not the power electronics controller—would not invalidate the original power system testing certification. However, new firmware updates to the communication module may void the interoperability certification requirements in IEEE 1547.1-2020. For this reason, most modern inverter evaluations include full software evaluations to UL 1998 or UL 60730 as part of their UL 1741 certifications. The ability to reduce or eliminate full IEEE 1547.1 retesting is a significant benefit that likely outweighs the comparatively minor cost of the software/firmware evaluation when compared to the time and expense of months or IEEE 1547.1 retesting. In some cases, NRTLs may make formal allowances for any revisions to non-safety critical code, so there are significant advantages to complete UL 1998 or UL 60730 certification so cybersecurity updates can be pushed quickly to products, assuming the code is written in a segmented/compartmentalized manner.

Additionally, UL 5500 *Standard for Safety for Remote Software Updates* is designed to be used with the end product safety standard to maintain certifications through remote firmware update processes. UL 5500 is applicable for remote software updates to safety critical software but does not directly address certification maintenance. The scope states, “this standard covers REMOTE software updates taking into account the manufacturer's recommended process. It is limited to software elements having an influence on safety and on compliance with the particular end product safety standard." Therefore, if the firmware update is for the communication elements of the DER device, leaving all the protection code untouched, it is not applicable. In most cases, critical cybersecurity patches are going to update or change web interfaces, communication protocol implementations, or software related to the exposed interfaces of the DER. As a result, UL 5500 is not applicable to those updates as long as the change does not impact safety critical software or compliance to UL 5500 itself.

3.2. Patch Timeline Requirements

In many of the patching standards, vulnerability discovery, disclosure, and patch development are out of scope. In IEC 62443-2-3, the patch lifecycle states start at *available* and end at *installed*. IEC 62443-4-1 *SUM-5 Timely delivery of security patches* does lightly weigh in on this issue, stating: "a process shall be employed to define a policy that specifies the timeframes for delivering and qualifying security updates to product users and to ensure that this policy is followed. At a minimum, this policy shall consider the following factors:

a) the potential impact of the vulnerability
b) public knowledge of the vulnerability
c) whether published exploits exist for the vulnerability
d) the volume of deployed products that are affected
e) the availability of an effective mitigation in lieu of the patch"

IEC 62443-2-3 continues, "Security updates typically have target release timing which is based on the factors listed in this requirement. For example, some companies classify patches as required to be addressed within 30 days, 60 days or 90 days or longer of being found." In the future, a standardized scoring mechanism like the Common Vulnerability Scoring System (CVSS) could be used to help determine the appropriate timeline for fixing the vulnerability. This could be done when creating Common Vulnerabilities and Exposures (CVEs) or internally by the DER vendor to determine the severity and timeline for patching their products.

DER vendors should participate in bug bounty programs and responsible disclosure/Coordinated Vulnerability Disclosure (CVD) practices should be used within the DER ecosystem. But, establishing a vulnerability disclosure model and policy in which a vulnerability or an issue is disclosed within a fixed amount of time is much more difficult. When must a vulnerability and the associated patch be announced by a DER vendor? We suggest the requirement in DER-PS.12 in Section 4, Guidance for Practitioners, to address this issue.

3.3. Public Notification Process

Once a patch is available for DER equipment, what are the appropriate notification mechanisms? Is it enough to post updates to a company website? Or should the vendor send owners an email or provide a notification on the DER equipment itself? Mandating a website with all patches for DER products with the reason (associated security vulnerability being addressed and mitigation), seems like a minimum practice. Additional notification mechanisms are not currently defined.

3.4. Incentivizing or Mandating Asset Owners to Patch Equipment

It is not clear what an appropriate asset owner concurrence model should be for DER patches. Should there be an opt-in or opt-out policy? Do the owners need to be notified and agree to each patch? Should interconnection agreements include language around automated patch updates? Are there any warranty issues around patch updates? What are the enforcement mechanisms, i.e., can the interconnection agreement be revoked if a critical patch was not applied? Can a patching incentive program be created to further encourage adoption? Can a DER owner reject a critical security patch? What should happen if a device is not applying critical updates?

It is assumed that interoperable, residential-scale DER equipment will have direct or indirect communication connections to DER vendors to update firmware. In the case of residential solar and rooftop commercial installations, it is likely the facility owner will have the patch digitally delivered directly from the DER product supplier. In those cases, the service provider or aggregator should not interfere in that update process, unless these updates could affect their business operations. In that case, (a) the service providers and aggregators may closely track and manage the status of patches within their fleet of DER devices, or (b) the service providers may be responsible for distributing patches—though these chains of command have not been defined. In the first case, when a new patch is available, DER devices would automatically download and install the patch based on pre-defined requirements set forth in the interconnection, vendor, or aggregator participation agreements, especially if the patch is identified as a critical security patch. Perhaps in other cases, patches are classified as optional and it is up to the owner to choose to apply the update or not. Addressing these considerations are necessary for robust residential patch management.

3.5. Challenges in the Utility Environment

In the case of utility-owned or larger commercial-scale DER equipment, it is anticipated that the utility or site owner/operator will manage the patching process, like that defined in IEC 62443-2-3. It is likely that not all patches will be applied for utility applications based on their patch management internal review processes.

Few utility-scale sites are air-gapped and at least have a connection to the utilities OT/SCADA systems. However, if the DER equipment does not have the ability to reach out to the vendor to request firmware updates, alternative (likely manual) methods may be required to update the firmware on this equipment. This process in its most simple form could be a technician downloading the firmware on a USB stick and going to the site to install the update (i.e., the sneakernet approach). More likely, the utility or commercial owner/operator will have the ability to log into the DER devices and provide them with the update firmware package. IEC 62443-2-3 Appendix B discusses asset owner guidance on patching:

- **Information gathering activities** – This includes creating the inventory of updateable devices, building product supplier relationships, and evaluating/assessing the existing environment and its supportability requirements.
- **Project planning and implementation activities** – This includes developing the business case, defining the roles and responsibilities, establishing a patch deployment and installation infrastructure, and establishing backup and restoration infrastructure.
- **Procedures and policies for patch management** – This includes monitoring for patches, evaluating patches, testing patches, installing patches and change management.
- **Operating a patch management system** – This includes executing the patch management procedures and policies, vulnerability awareness, outage scheduling, inventory maintenance, new device additions, reporting, key performance indicators, auditing and verification.

Additionally, it would be valuable for utilities to establish sample procurement/contracting language for DER vendors that include patching requirements. For instance, these requirements could include the following, "The vendor must conduct patch testing to verify functionality with the products," "The vendor must provide in writing their approach for publicly releasing patches and their notification process," etc.

3.6. DER Code Signing

As discussed earlier, the primary technology used for secure patching is PKI-based code signing. In the DER environment, a code-signing environment may look like that in Figure 5. Starting from the upper left, a firmware development team creates new *data* that likely represents a firmware update in the *build system*. The software team performs whatever internal reviews of the code are necessary and then they send this to a *sign tool* to sign the software. The *signer* then takes the software data and authenticates the person requesting the code be signed with an access control system. Once passing this authentication step, the user can request the data be signed by the signing framework. Only those persons/users with the correct permissions/authority for a given project/submodule can sign this code. This action is logged for later auditing. Once the firmware has been signed with the private key of the vendor, the software is made available for the DER owners, operators, aggregators, and service providers.

In the case of utility-owned DER, the patch may be downloaded and either (a) added to a DERMS or patch management system that can update the utility device firmware, (b) the new firmware could be transported to the site by a technician for local update, or (c) appropriate OT boundary protection rules could be created such that the DER can reach out to the DER vendor directly to update its firmware. In the case of smaller residential or commercial installations, these will use the public internet to download patch updates and, after concurrence from the owner, be installed on the DER device.

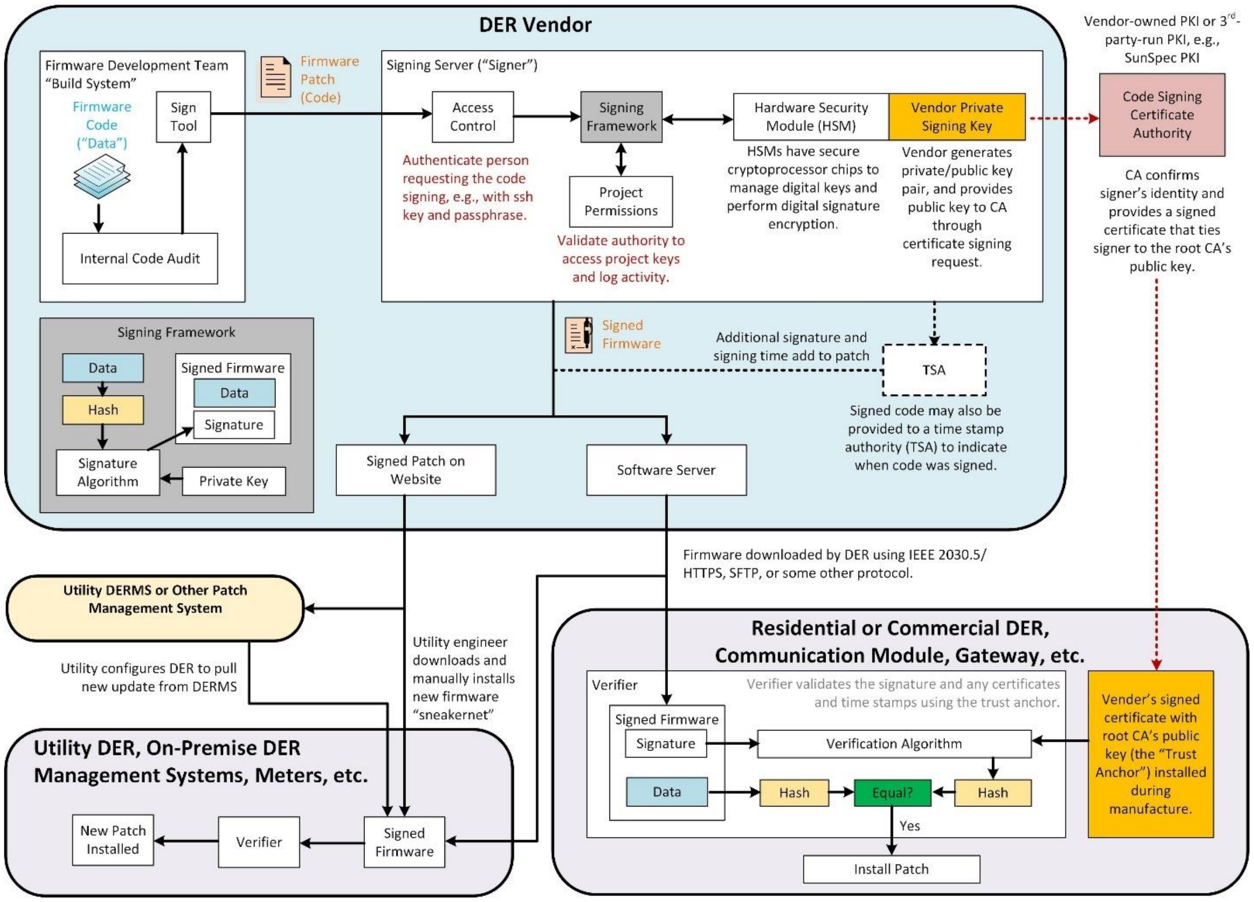

DER Vendor
Firmware Development Team "Build System"
Firmware Code ("Data")
Sign Tool
Internal Code Audit
Firmware Patch (Code)
Signing Server ("Signer")
Access Control
Authenticate person requesting the code signing, e.g., with ssh key and passphrase.
Signing Framework
Project Permissions
Validate authority to access project keys and log activity.
Hardware Security Module (HSM)
HSMs have secure cryptoprocessor chips to manage digital keys and perform digital signature encryption.
Vendor Private Signing Key
Vendor generates private/public key pair, and provides public key to CA through certificate signing request.
Vendor-owned PKI or 3rd-party-run PKI, e.g., SunSpec PKI
Code Signing Certificate Authority
CA confirms signer's identity and provides a signed certificate that ties signer to the root CA's public key.
Signing Framework
Data
Hash
Signature Algorithm
Private Key
Signed Firmware
Data
Signature
Signed Firmware
Additional signature and signing time add to patch
TSA
Signed code may also be provided to a time stamp authority (TSA) to indicate when code was signed.
Signed Patch on Website
Software Server
Firmware downloaded by DER using IEEE 2030.5/ HTTPS, SFTP, or some other protocol.
Utility DERMS or Other Patch Management System
Utility configures DER to pull new update from DERMS
Utility engineer downloads and manually installs new firmware "sneakernet"
Utility DER, On-Premise DER Management Systems, Meters, etc.
New Patch Installed
Verifier
Signed Firmware
Residential or Commercial DER, Communication Module, Gateway, etc.
Verifier
Verifier validates the signature and any certificates and time stamps using the trust anchor.
Signed Firmware
Signature
Data
Verification Algorithm
Hash
Equal?
Hash
Yes
Install Patch
Vender's signed certificate with root CA's public key (the "Trust Anchor") installed during manufacture.

Figure 5: Code-Signing for DER Devices.

In this process, there are multiple threats to code-signed firmware.[80] For example, it is possible that software developed by an organization has malicious firmware embedded in the signed version. This could be perpetrated by an insider or through compromise of the firmware development environment, as was the case in the well-known SolarWinds attack. NIST's "Security Considerations for Code Signing" White Paper[81] covers the following security considerations with code-signing firmware updates:

- **Theft of private signing key**: Private signing keys that are not properly protected are at risk of theft, allowing a successful attacker to sign arbitrary code.
- **Issuance of unauthorized code signing certificates**: Weak protections on CA private keys used to issue code signing certificates, or weak vetting procedures used to issue those certificates, could allow an attacker to obtain one or more unauthorized code signing certificates.
- **Misplaced trust in certificates or keys**: Verifiers could trust certificates or keys for code signing that were never intended to be used for code signing.
- **Signing of unauthorized or malicious code**: Code signing procedures could allow malicious or unauthorized code to be signed.

[80] K. Goldman, et al. "Best Practices for Firmware Code Signing", IBM White Paper.
[81] D. Cooper, et al., "Security Consideration for Code Signing," NIST Cybersecurity White Paper, URL: https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.01262018.pdf

- **Use of insecure cryptography**: Use of weak or insecure cryptographic algorithms or key generation methods could allow cryptanalytic or brute-force attacks.

Each of these threats could be realized with inside actors so appropriate access control mechanisms and protections must be in place in each stage of the signing process. Supply chain threats are especially pertinent for DER device code signing processes too, because portions of the code may be generated and signed internationally. Awareness of this risks and application of appropriate mitigations are critical for all DER vendors. The NIST whitepaper provides the following recommendations for vendors:

- Identify and authenticate trusted users
- Authorize trusted users and maintain and regularly review/audit a list that specifies who can submit code.
- Separate roles within the Code Signing System (CSS), especially the CSS administration role and authorized code submitter.
- Establish policies and procedures for reviewing, vetting and approving code.
- Use strong cryptography.
- Protect the signing keys, preferably in an HSM with minimal functionality (e.g., signing).
- Use a separate CSS for development
- Isolate and protect the CSS as a critical asset.
- Utilize auditing and periodically review logs.

3.7. End-of-Service

Generally, it is a best practice for product suppliers to notify users of end-of-service, end-of-feature, and end-of-device-support events with enough lead time to allow owners time to make appropriate adjustments to their systems. Since DER equipment may operate for 25 or more years in the field, this becomes a difficult prospect, especially as companies are regularly going out of business or being bought out. This raises several questions about who becomes responsible for updating vulnerable code when a vendor is no longer in existence.

4. GUIDANCE FOR PRACTITIONERS

The DER Cybersecurity Workgroup systematically worked through patching best practices and standards to determine applicability to the DER ecosystem. The following sections provide a list of suggested firmware update requirements for DER Equipment, Product Suppliers, Integrators, Aggregators, and Owners. These requirements were carefully down-selected based on their application to the DER ecosystem and their ability to be objectively verified through an certification or auditing process. For the requirements below, MUST indicates an obligatory requirement, SHOULD indicates a recommended practice, and MAY is an optional security action.[82]

4.1. Requirements for DER Equipment

DER-Eq.1. DER devices MUST support the ability to be updated.

DER-Eq.2. DER devices MAY support local updates.

DER-Eq.3. DER devices SHOULD support the ability for remote updates.

DER-Eq.4. DER devices SHOULD support the ability for automated updates.

DER-Eq.5. DER devices SHOULD support the ability to be configured for and disable automated updates.

DER-Eq.6. DER devices SHOULD check periodically whether there is an available update.

DER-Eq.7. DER devices MUST verify the authenticity and integrity of software updates.

4.2. Requirements for DER Product Suppliers[83]

DER-PS.1. DER product supplier MUST document the firmware patching policy.[84]

DER-PS.2. DER product supplier MUST provide a policy on product support including firmware updates.

DER-PS.3. DER product supplier MAY publish the firmware patching policy publicly.

DER-PS.4. DER product supplier MUST provide software updates that are non-repudiable. Software update non-repudiation provides the product supplier with proof of patch delivery and the recipient is provided with proof of the product supplier's identity.

DER-PS.5. DER product supplier MUST use an appropriate standard to approve the selection of cryptography modules. DER product supplier MAY use a FIPS 140-2 Level 2 or greater cryptography modules for all keys used in code signing processes, with equivalent certifications recognized as acceptable.

[82] NOTE: definitions are included in the *Acronyms and Definitions* and *Terms and Definitions* sections.

[83] NOTE: Product supplier refers to the company or organization that is accountable for supplying patches. The term "product supplier" is used to align with the IEC 62443-2-3 nomenclature, as described in Section **Error! Reference source not found.**.

[84] Patching policy, or patch management policy, may include an overview of the corporate objectives, purpose, scope of the policy, what equipment is subject to the policy, roles and responsibilities, procedures, monitoring and reporting, auditing, enforcement strategy, and any exceptions.

DER-PS.6. DER product supplier MUST inform integrators, aggregators, and owners in a recognizable and apparent manner (e.g., on a website) that a security update is available.

DER-PS.7. DER product supplier MUST provide information on applicability, compatibility, and risks mitigated by the patch.

DER-PS.8. DER product supplier MUST notify the aggregator, owner, or operator when the application of a software update will disrupt the basic functioning of the DER.

DER-PS.9. DER patches MAY include the requirements in DER-PS.6, DER-PS.7, and/or DER-PS.8 in a IETF RFC 4108-compliant patch manifest.

DER-PS.10. DER product supplier MUST provide, in an accessible way that is clear and transparent to the aggregator, owner, or operator, the defined DER device support period.

DER-PS.11. DER product supplier MUST publish a list of all patches and their approval status.

DER-PS.12. When technically feasible, DER product supplier MUST provide a patch or alternative risk mitigation for security vulnerabilities within 60 days of notification or internal discovery by the DER product supplier.

DER-PS.13. DER product suppliers SHOULD use mechanisms that prevent firmware downgrade attacks, i.e., prevent updates to previous software versions.

DER-PS.14. DER product supplier SHOULD write segmented/modular software and complete UL 1998 or UL 60730 certification of all DER firmware to minimize product re-evaluations.

DER-PS.15. DER product supplier SHOULD provide adequate warning (at least two years in advance) when components are reaching 'end of life' or if cyber security patches will no longer be made available.

DER-PS.16. DER product supplier SHOULD include a Software Bill of Materials (SBOM) NTIA Minimum Elements in a machine-readable format. It is recommended to use SPDX, CycloneDX, SWID Tags or other industry compliant formats.

DER-PS.17. DER product supplier MAY inform asset owners within 30 days after a patch is released for third-party software operating within the DER.

4.3. Requirements for DER Integrators

DER-I.1. DER devices SHOULD be commissioned with the most recent firmware version.

4.4. Requirements for DER Aggregators[85]

DER-A.1. DER aggregator MUST document the patching policy.

DER-A.2. DER aggregator MUST maintain a patching schedule which includes a planned interval for evaluation of new patches.

DER-A.3. DER aggregator SHOULD provide the ability to opt-out of automated updating.

[85] In cases where the DER aggregator is acting as an asset owner/operator or manager, the DER-O requirements also apply.

DER-A.4. DER aggregator SHOULD maintain an up-to-date inventory of all electronic devices in the DER system with the associated patch version.

DER-A.5. DER aggregator SHOULD maintain a record of all previously installed firmware versions for each device.

4.5. Requirements for Large Commercial or Utility DER Owners[86]

DER-O.1. DER owner MUST document the firmware update policy.

DER-O.2. DER owner MUST document patch management process for tracking, evaluating, and installing DER security patches. The tracking portion shall include the identification of source(s) of DER security patches.

DER-O.3. For applicable DER patches, within the schedule established in the firmware update policy, the DER owner MUST take one of the following actions: (a) apply the applicable patch(es), (b) create a dated mitigation plan, or (c) revise an existing mitigation plan. Mitigation plans shall include planned actions to mitigate the vulnerabilities addressed by each security patch within 35 days or a timeframe established by the update policy.

DER-O.4. DER owner MUST implement the mitigation plan within 35 days or a timeframe specified in the plan, unless a revision to the plan or an extension to the timeframe is approved.

DER-O.5. DER owner SHOULD check every 35 days or other fixed periodicity documented in the firmware update policy whether there is an available update.

DER-O.6. DER owner SHOULD maintain an up-to-date inventory of all electronic devices in the DER system with the associated patch version.

DER-O.7. DER owner SHOULD maintain a record of all previously installed firmware versions for each device.

DER-O.8. DER owner SHOULD test the installation of patches in a way that accurately reflects the production environment.

DER-O.9. DER owner SHOULD update records as indicated in the firmware patch policy to include installed, authorized, effective, and released firmware versions.

[86] Owners with special agreement(s) with an Aggregator or Area EPS may be exempt. In many cases, DER-O requirements will apply to equipment operators.

5. CONCLUSION

The SunSpec/Sandia DER Cybersecurity Workgroup met for more than nine months to discuss issues associated with patching DER equipment. This report provides a summary of those discussions, including pertinent R&D literature and standards related to firmware updates, challenges associated with DER patching, and guidance for practitioners completed in this process. Several complex issues were identified which must be addressed by the DER industry to create a harmonized approach to secure firmware updates. Recommendations provided in this report are designed to be ported over to and refined within IEEE 1547.3, utility handbooks, and other national and local DER interconnection standards in the future.

DISTRIBUTION

Email—Internal

Name	Org.	Sandia Email Address
Charles Hanley	08810	cjhanle@sandia.gov
Summer Ferreira	08812	srferre@sandia.gov
Brian Gaines	09366	bgaines@sandia.gov
Jay Johnson	08812	jjohns2@sandia.gov
Ifeoma Onunkwo	09366	ionunkw@sandia.gov
Technical Library	01977	sanddocs@sandia.gov

Email—External

Name	Company Email Address	Company Name
Jeremiah Miller	jeremiah.miller@ee.doe.gov	U.S. Department of Energy
Guohui Yuan	guohui.yuan@ee.doe.gov	U.S. Department of Energy

This page left blank