TLP - CLEAR

EUROPEAN UNION AGENCY
FOR CYBERSECURITY

ENISA Technical Advisory on Secure Update Mechanisms

MAY 2026

About ENISA

The European Union Agency for Cybersecurity, ENISA, is the Union's agency dedicated to achieving a high common level of cybersecurity across Europe. Established in 2004 and strengthened by the EU Cybersecurity Act, the European Union Agency for Cybersecurity contributes to EU cyber policy, enhances the trustworthiness of ICT products, services and processes with cybersecurity certification schemes, cooperates with Member States and EU bodies, and helps Europe prepare for the cyber challenges of tomorrow. Through knowledge sharing, capacity building and awareness raising, the Agency works together with its key stakeholders to strengthen trust in the connected economy, to boost resilience of the Union's infrastructure, and, ultimately, to keep Europe's society and citizens digitally secure. More information about ENISA and its work can be found here: www.enisa.europa.eu.

CONTACT

To contact the authors, use product_security@enisa.europa.eu.
For media enquiries about this paper, use press@enisa.europa.eu.

AUTHORS

ENISA

LEGAL NOTICE

COPYRIGHT NOTICE

Table of Contents

About ENISA 1

1. Scope and context 4

1.1 Note on the current version of the document 5

2. Update architecture and data flows 6

2.1 The update lifecycle 6

2.2 Common update delivery architectures and models 7

2.2.1 In-band (built-in) updater 7

2.2.2 Platform-managed updates 8

2.2.3 Out-of-band manual updates 8

2.2.4 Enterprise-managed or staged updates 8

2.3 Design considerations for secure update mechanisms 9

3. Update lifecycle threats 10

3.1 Build/Package/Establish trust 11

3.2 Publish 11

3.3 Check for updates 11

3.4 Retrieve 12

3.5 Verify 12

3.6 Install 12

3.7 Record Status 13

4. Security controls 14

4.1 Preparation and Publication 14

4.2 Discovery and Retrieval 16

4.3 Verification and Installation 16

4.4 Observability and Recovery 17

5. Illustrative end-to-end example 19

5.1 Scenario 19

5.2 Setup 19

5.3 Security control implementation across stages 20
5.3.1 Preparation and publication 20
5.3.2 Discovery and retrieval 21
5.3.3 Verification and installation 21
5.3.4 Observability and Recovery 22
5.4 Security Claims and Evidence 22

A Annex 1: Stride mapping of update lifecycle threats 24

1. Scope and context

Recent incidents affecting software supply chains and update mechanisms have shown that build pipelines and update delivery systems are attractive targets. At the same time, recent advances in vulnerability research, including AI-assisted tools like Anthropic's Mythos, are likely to increase the number of vulnerabilities that manufacturers need to assess, prioritise, and ultimately patch through updates[1]. These trends reinforce the need for update mechanisms that can deliver security fixes securely, in a timely manner, and reliably.

In this document, software update mechanisms refer to services and components used to deliver updates to client systems. This includes publishing and discovering available updates, retrieving them, verifying their integrity and authenticity, and installing them. These update mechanisms typically support different types of updates, such as software versions, vulnerability patches, configuration changes, signatures, models, and firmware.

This advisory provides practical technical guidance for manufacturers of products with digital elements, specifically teams responsible for designing, implementing, or operating update mechanisms. It is particularly intended to help micro, small, and medium-sized manufacturers understand common update lifecycle threats and apply a set of controls that could mitigate such threats and support secure update delivery. It is not intended to prescribe a single best architecture, replace established secure update frameworks, or provide an exhaustive implementation model.

The document first introduces a typical update lifecycle, highlighting common update delivery models. Subsequently, it outlines threats across the lifecycle and moves on to provide security recommendations to mitigate these threats. Finally it concludes with an illustrative end-to-end example showing how recommended controls can be applied in practice.

The advisory considers regulatory requirements such as those set out in the Cyber Resilience Act (CRA)[2], in particular essential requirements relating to security updates. These include, among others, requirements related to:

- timely vulnerability remediation and security updates (Annex I, Part II, 2.2)
- integrity protection and corruption reporting (Annex I, Part I, 2(f))
- secure distribution of updates (Annex I, Part II, 2.7)
- prompt dissemination of security updates (Annex I, Part II, 2.8)
- separation of security and functional updates where feasible (Annex I, Part II, 2.2)
- automatic security updates with user controls where appropriate (Annex I, Part I, 2(c))
- user notification of available updates (Annex I, Part I, 2(c); Part II, 2.8)
- vulnerability information and remediation guidance (Annex I, Part II, 2.4)

This advisory may support manufacturers in addressing parts of these requirements in practice. It aims to provide technical context to such requirements by describing common threats across the update lifecycle and practical controls that can help mitigate them.

[1] https://therecord.media/british-cyber-ai-patch-wave, Accessed 20/05/26.
[2] https://eur-lex.europa.eu/eli/reg/2024/2847/oj/eng, Accessed 20/05/26.

However, this document does not constitute legal advice, a formal interpretation of the CRA, or a presumption of conformity. Compliance with applicable obligations remains the responsibility of the manufacturer and depends on the specific product, risks, and intended use.

Moreover, it does not cover how to develop or validate the technical correctness of the patch itself. For example, it does not explain how to correct a specific vulnerability, how to perform the root cause analysis, or how to determine whether a patch fully resolves a vulnerability.

The recommendations are intended to be technology-agnostic and are not intended to require any specific update framework. They are, however, consistent with recognised practices reflected in frameworks and guidance such as The Update Framework (TUF)[3], Uptane[4], and NIST SP 800-40[5].

1.1 Note on the current version of the document

<To-do> public consultation info

[3] https://theupdateframework.github.io/specification/latest/, Accessed 20/05/26.
[4] https://uptane.org/docs/latest/standard/uptane-standard, Accessed 20/05/26.
[5] https://csrc.nist.gov/pubs/sp/800/40/r4/final, Accessed 20/05/26.

2. Update architecture and data flows

2.1 The update lifecycle

Software update mechanisms are designed to deliver different types of artefacts, including binary updates, delta patches, plugin or module updates, content or signature updates, configuration or policy updates, and firmware updates.

Regardless of the type of artefact, the update process typically follows a lifecycle ranging from the preparation up until the installation and status recording.

The table below outlines the typical update lifecycle steps, actors involved, and their respective key actions. This lifecycle will be used as a reference throughout the document.

ID	Step	Actor(s)	Key action
1	**Build/package/Establish trust**	Manufacturer	Create update artefact and metadata and establish their authenticity and integrity (e.g. signing)
2	**Publish**	Publication service / Update repository	Make update artefact and metadata available
3	**Check for updates**	Client updater	Discover applicable update
4	**Retrieve**	Client updater (retrieves from update repository)	Download update artefact and metadata
5	**Verify**	Client updater	Validate authenticity, integrity, applicability
6	**Install**	Client updater (initiates) / Endpoint (executes)	Apply update, possibly with rollback point
7	**Record status**	Endpoint	Log and report outcome

The main actors involved in the typical update lifecycle include:

- **Producer (or manufacturer):** Develops, packages, and establishes authenticity and integrity for update artefacts and metadata, typically through cryptographic signing.
- **Publication service:** Publishes update artefacts and metadata for distribution.
- **Update repository:** Stores and serves published updates content to clients.
- **Client updater:** Discovers, retrieves, verifies, and installs updates.
- **Endpoint:** Executes installation and records the update status.

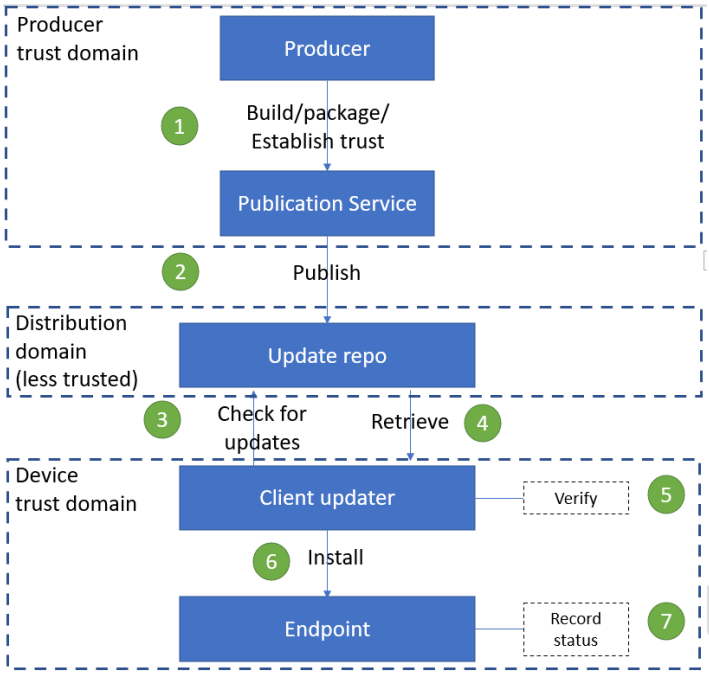

Figure 1: Typical update lifecycle

Across all stages, update security typically relies on a baseline trust model aligned with the trust domains (as shown in Figure 1):

- The device trust domain is provisioned with a root public key, which acts as a trust anchor
- The manufacturer uses the corresponding root authority to approve the keys, signing roles, or equivalent trust mechanisms that are allowed to authorise update metadata or artefacts.
- The distribution domain, including update repositories and intermediaries such as CDNs, is not trusted by default

This means that updates are trusted based on cryptographic verification (e.g. signatures and signed metadata), rather than because of where they are downloaded from. Even if the delivery mechanism is compromised, unauthorised or modified updates should be rejected. In higher-risk or more complex deployments, this baseline model can be extended with delegated signing roles, freshness metadata, threshold signatures, and additional repository roles[6].

2.2 Common update delivery architectures and models

While update mechanisms typically follow a lifecycle described above, the way those lifecycle steps are implemented can vary depending on the delivery architecture. This section describes common update delivery architectures and models, highlighting how responsibilities for discovery, retrieval, verification, and installation may differ across them.

2.2.1 In-band (built-in) updater

The product contains its own update client responsible for discovering and installing updates. In this setup, typically all of the update lifecycle steps are executed within the product and manufacturer-

[6] https://theupdateframework.github.io/specification/latest/, Accessed 20/05/26.

controlled services. The client updater performs discovery, retrieval, verification, and installation locally. In such cases, strong client-side verification and secure update design are essential, as the product is directly exposed to update sources.

2.2.2 Platform-managed updates

Updates are distributed and installed through an external platform that enforces validation and delivery policies. In this setup, parts of the update lifecycle, particularly discovery, verification, and distribution, are handled by an external platform. As a result, security depends on both the producer and the platform.

2.2.3 Out-of-band manual updates

Updates are obtained and installed manually by the user or administrator, without an automated update mechanism. In this setup, discovery and retrieval are typically performed manually by a user or administrator. This typically introduces a higher risk of user error, so strong authenticity verification and clear user guidance are important.

2.2.4 Enterprise-managed or staged updates

Updates are published by the producer but distributed to endpoints via customer-controlled infrastructure. In this setup, the update lifecycle is split between the producer and the customer organisation, so security controls are required both at the producer and enterprise levels.

The table below provides a few examples of common update delivery architectures and models.

Architecture/model	Examples
In-band (built-in) updater	Desktop application updater
	Application component updater (e.g. plugin updater within a product)
	Browser auto-update mechanisms
Platform-managed updates	Mobile application stores
	Linux package repositories
	Enterprise mobile device management (MDM) platforms
	Browser extension stores (with automatic update delivery)
Out-of-band manual updates	Installer downloaded from a vendor website
	Patch package delivered via secure portal
	Emailed update package
Enterprise-managed or staged updates	Internal update staging (e.g. WSUS-like systems)
	Enterprise mirrors
	Patch management servers
	Air-gapped update transfer

2.3 Design considerations for secure update mechanisms

Regardless of the delivery model, the security of the update mechanism ultimately depends on how trust is established and enforced. In simpler implementations, an approach could involve a device trusting a root public key and one operational signing key. The corresponding root authority authorises the operational signing key, and the operational signing key is used to sign update metadata. This approach is easier to implement and is common in embedded systems. More advanced update systems use multiple signing roles instead of a single signing key. For example, different roles may be used to:

- Authorise which signing keys or roles are trusted
- Define which updates are available
- Confirm the latest version of update metadata
- Limit how long metadata remains valid
- Ensure that related metadata remains consistent

This separation reduces risk. If one key or role is compromised, it does not necessarily provide the attacker full control over the update process. Higher-assurance deployments may also use threshold signatures where more than one authorised key holder must approve sensitive update metadata or key changes. Examples of frameworks that formalise these concepts include The Update Framework (TUF)[7] and Uptane[8].

[7] https://theupdateframework.io/

[8] https://uptane.org/

3. Update lifecycle threats

Software update mechanisms represent an interesting target for attackers, since a successful compromise can allow them to distribute malicious code (often executed with elevated privileges), prevent security patches, or disrupt products at scale.

The update lifecycle faces several threats across stages. This section lists threats faced at every step of a typical update lifecycle (as shown in the figure below). It also explains the impact they may have to a product or organisation, providing real world examples.

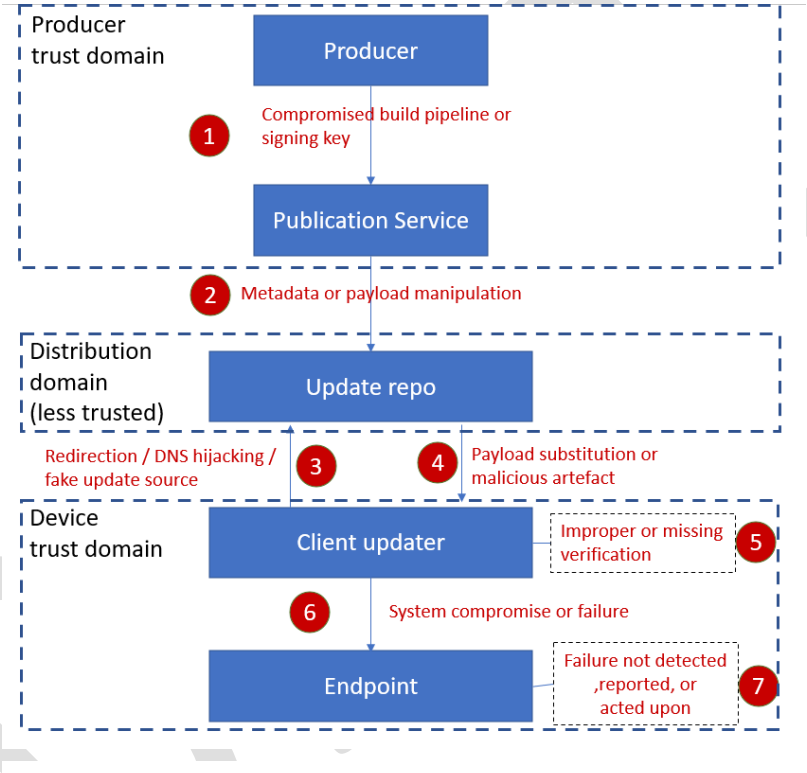

Figure 2: Threats across lifecycle

For the purposes of this advisory, the threat model assumes that attackers may attempt to compromise build pipelines, update artefacts, signing keys, publication services, repositories, CDNs, DNS or redirection paths, metadata replay or staleness, client-side update state, installation processes, or privileged operator workflows.

An indicative STRIDE[9] mapping of the threats described in this section is provided in Annex 1

[9] https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats, Accessed 20/05/26.

3.1 Build/Package/Establish trust

Threat: Key threats at this stage include the compromise of the build pipeline or signing keys. Attackers, or individuals misusing privileged internal access, may gain unauthorised access to CI/CD systems, build servers, scripts, dependency sources, or signing infrastructure in order to tamper with the update creation process or obtain signing credentials.

Impact: A compromised build pipeline can result in malicious code or configuration changes being embedded into update artefacts. If such malicious inclusions are not detected, and the rest of the update steps are performed as normal, then these changes will ultimately be published, accepted, and installed on the endpoints. If signing keys are compromised, attackers may also be able to create and sign malicious artefacts so that they appear legitimate, undermining the trust model of the update mechanism and making subsequent detection more difficult.

Example: During the SolarWinds supply chain compromise[10] attackers managed to compromised the software build environment and inserted malicious code into signed updates distributed to customers.

3.2 Publish

Threat: Threats at this stage occur after the update has been created, targeting the systems used to publish or distribute it. Significant threats at this stage include the manipulation of update metadata or payloads during publication, as well as the compromise of publication services. Attackers, or individuals misusing privileged internal access, can modify published files, replace legitimate artefacts or interfere with the release process.

Impact: Compromise at this stage can result in clients being directed to malicious or unauthorised update artefacts, or in legitimate updates being replaced, withheld, or corrupted. Manipulation of metadata such as hashes, version information, download locations, or release details may cause clients to retrieve malicious content or delay the deployment of necessary security updates. If such changes are not detected at later lifecycle stages, affected updates may be accepted and installed on endpoints, or necessary updates may not be installed at all.

Example: In the Trivy supply chain compromise[11], attackers targeted release and distribution processes to make compromised software artefacts available to users through trusted channels. Similar publication threats also arise in package manager ecosystems, where malicious or unauthorised packages may be made available through trusted repositories. Related considerations are addressed in ENISA's technical advisory on the secure use of package managers.[12].

3.3 Check for updates

Threat: At this stage, attackers could try to tamper with how the client determines whether updates are available. This could include redirection to unauthorised sources, DNS hijacking, interception of update requests, fake update services, replay of previously valid metadata, or freeze attacks that prevent the client from learning about newer updates. Individuals misusing privileged internal access may similarly manipulate update availability or version information.

[10] https://www.solarwinds.com/blog/an-investigative-update-of-the-cyberattack, Accessed 20/05/26.
[11] https://github.com/aquasecurity/trivy/security/advisories/GHSA-69fq-xp46-6x23, Accessed 20/05/26.
[12] https://www.enisa.europa.eu/sites/default/files/2026-03/ENISA%20Technical%20Advisory%20-%20Package_Managers_Final.pdf, Accessed 20/05/26.

Impact: A compromise at this stage can result in clients receiving false or misleading information about the availability, legitimacy, or urgency of specific updates. Products may be directed to malicious sources, prevented from obtaining valid updates, or caused to remain on older vulnerable versions.

Example: In the recent Notepad++ update hijacking supply chain incident[13] attackers compromised update discovery mechanisms by intercepting or redirecting update traffic, causing selected users to contact an attacker controlled sources instead of the legitimate update service.

3.4 Retrieve

Threat: During retrieval, attackers could attempt to interfere with the download of the update artefacts from repositories, mirrors, CDNs, or other distribution infrastructure. This can include the substitution of the payload, the delivery of malicious artefacts, or the corruption of downloads. Similarly, individuals misusing privileged internal access may alter or replace content made available to clients.

Impact: A compromise at this stage can result in devices downloading malicious, incomplete, or corrupted update artefacts. Even where transport security is in place, without proper validation of the retrieved artefact, unauthorised content may be allowed to proceed to later lifecycle stages. This can ultimately lead to failed updates, delayed remediation, or installation of malicious software.

Example: In the ASUS Live Update (ShadowHammer) incident[14] in 2019, shows an example of payload substitution through a trusted update mechanism, whereby attackers used the vendor's update retrieval channel to distribute malicious update artefacts through the normal download process to specific users.

3.5 Verify

Threat: During verification, improper or incomplete checks against the authenticity, integrity, or applicability of the update can result in security weaknesses. Examples include failures in, or weak, signature validation, trust chain verification, certificate checks, hash comparison, version control, rollback protection, or checks confirming that the update is intended for the specific device or configuration.

Impact: Weaknesses at this stage can allow threats that materialised in previous lifecycle steps, including malicious, corrupted, outdated, or unauthorised updates, to be treated as valid. Since verification is a primary trust decision point, a failure at this point can undermine the effectiveness of the entire update mechanism.

Example: After the Notepad++ update hijacking incident, the project strengthened its update mechanism by improving installer verification controls, highlighting the importance of proper update verification.

3.6 Install

Threat: Threats at the installation stage include the execution of malicious code with elevated privileges, the interruption of the installation process, and the partial application of updates.

[13] https://notepad-plus-plus.org/news/hijacked-incident-info-update/, Accessed 20/05/26.
[14] https://cert.europa.eu/publications/security-advisories/2019-007/, Accessed 20/05/26.

Impact: Failures or compromises at this stage can lead to a system takeover, the persistence of malicious code, inconsistent software states, loss of functionality, or even device failure and unavailability.

Examples: The CrowdStrike Falcon update outage in 2024[15], while not malicious, demonstrates how a faulty security update can cause widespread system crashes, loss of functionality, and operational disruption during the installation stage. While the SolarWinds incident demonstrated how malicious code delivered through a trusted update can be installed with elevated privileges on systems.

3.7 Record Status

Threat: Logging, monitoring, alerting, or reporting mechanisms may be absent, disabled, misconfigured, or deliberately suppressed by attackers or privileged insiders.

Impact: Failures in logging, monitoring, reporting, or alert handling could prevent update related problems from being detected and/or acted upon. This could lead to devices remaining unpatched, partially updated, rolled back, or compromised without user or admin awareness.

Example: The CrowdStrike Falcon outage clearly showed how timely visibility into update failures and affected endpoints is important for quick response and remediation, including at scale.

[15] https://www.crowdstrike.com/wp-content/uploads/2024/08/Channel-File-291-Incident-Root-Cause-Analysis-08.06.2024.pdf, Accessed 20/05/26.

4. Security controls

This section presents security controls aimed at addressing and reducing the likelihood or impact of the threats described in Section 3. The controls are grouped into four stages: preparation and publication, discovery and retrieval, verification and installation, and observability and recovery. The controls are aligned with security principles from the ENISA Secure by Design and Secure by Default playbook[16], translating the principles into engineering practices specific to update mechanisms.

Stage	Update Lifecycle stage	Actors	Security objectives	Relevant SBD principles
Preparation and Publication	1 2	Producer Publication service / Update repository	Ensure that only authorised, integrity-protected and correctly signed updates are produced and made available for distribution.	Supply chain controls, Lifecycle management, Least privilege, Defence in depth, Vulnerability and patch management Strong identity and auth architecture
Discovery and Retrieval	3 4	Client updater Update repository	Ensure that clients obtain update information and artefacts from authentic, trusted sources without exposure to redirection, replay or substitution.	Strong identity and authentication architecture, Secure communication by default, Attack surface minimization, Defence in depth, Automated maintenance and updates
Verification and Installation	5 6	Client updater Endpoint	Ensure that only valid, untampered and applicable updates are accepted and safely applied without compromising system integrity.	Defence in depth, Least privilege, Lifecycle management
Observability and Recovery	7 Cross cutting	Endpoint	Ensure that update outcomes are reliably recorded and that the system can detect, respond to and recover from update-related failures or compromise.	Logging, monitoring and alerting, Incident response and recovery, Transparent security posture Lifecycle management Secure Recovery and Ownership Lifecycle

4.1 Preparation and Publication

This stage covers the generation, authorisation, and publication of update artefacts and associated metadata. In the context of security updates, the preparation and publication processes should support timely release, safe deployment, and avoid unnecessary barriers that could delay deployment, allowing users to access and apply updates without unnecessary friction. The controls in this stage align with principles from the ENISA Secure by Design and Secure by Default playbook. In particular, supply chain controls and lifecycle management ensure that updates are built from trusted inputs, validated, and released through controlled processes, while least privilege supports key

[16] https://www.enisa.europa.eu/sites/default/files/2026-03/ENISA_Secure_By_Design_and_Default_Playbook_v0.4_draft_for_consultation.pdf, Accessed 20/05/26.

protection and access restrictions, and signing, validation, and metadata integrity checks reflect defence in depth. Secure publication mechanisms, dependency checks, and the timely and well-structured release of security updates further align with vulnerability and patch management.

The following table lists the recommendations for this stage:

Recommendation	Description
Secure signing practices	Apply strong cryptographic signing to update metadata and ensure update artefacts are cryptographically bound to that metadata. Where appropriate, apply direct signing to update artefacts as an additional control.
Key protection and management	Protect credentials (e.g. signing keys) used to create, approve, or publish updates against unauthorised use. This includes secure key storage (e.g. HSM), strong authentication, restricted access, separated key roles, monitoring, and timely rotation or revocation where compromise is suspected.
Metadata integrity and consistency	Generate and validate update metadata, Ensure that metadata fields are consistent and accurately describe the published update artefacts, including the version, applicability, hashes, and file sizes.
Pre-release validation	Validate the update artefacts and metadata before publication, including integrity checks, signature verification, and deployment readiness testing.
Trusted build environment	Perform build and packaging activities in controlled and isolated environments, ensuring that only authorised code, dependencies, and configurations are included in update artefacts.
Provenance and traceability	Maintain traceability between source code, build inputs, build tools, build environment, approvals, build outputs, and published artefacts, including provenance information where appropriate. Higher-assurance implementations should also use signed provenance or attestation mechanisms, such as SLSA[17] provenance or in-toto attestations[18], to verifiably link source code, build inputs, build processes, build environments, and release artefacts.
Dependency check	Check third-party components, dependencies, or externally sourced artefacts included in updates, to identify and address known vulnerabilities prior to release.
Controlled release workflow	Implement a formal release and publication processes with defined approval gates. Use separation of duties and approval controls for high-impact releases.
Secure publication interfaces	Protect the publication services, repositories, and release APIs with strong authentication, least privilege, access controls, audit logging, and monitoring.
Update structuring and release strategy	Design the update artefacts and the release processes to support the independent delivery of security updates from functional changes. This allows for timely releases of critical security updates while minimising disruption[19]. The update structure should also support automatic installation of security updates by default where possible.
Advisory linkage and user-facing information	Where updates address security vulnerabilities, ensure that updates are released with clear information describing the vulnerability, its impact, severity, and remediation, allowing users to understand the urgency and prioritise timely update deployment.
Safe update behaviour testing	Ensure that updates can be safely applied without causing system unavailability, instability or unintended side effects. This includes testing installation procedures, confirming compatibility with supported versions and configurations, and ensuring that rollback or recovery mechanisms work correctly in case of failure[20].

[17] https://slsa.dev/spec/v1.2/, Accessed 20/05/26.
[18] https://in-toto.github.io/, Accessed 20/05/26.
[19] https://discussions.apple.com/docs/DOC-250007290, Accessed 20/05/26
[20] https://www.crowdstrike.com/wp-content/uploads/2024/08/Channel-File-291-Incident-Root-Cause-Analysis-08.06.2024.pdf, Accessed 20/05/26.

4.2 Discovery and Retrieval

This stage determines how updates are discovered and retrieved in a manner that ensures that clients only interact with trusted sources and receive legitimate update information. Besides ensuring secure communication and source authenticity, the update mechanism should support timely discovery and retrieval of updates to allow for the prompt remediation of security related issues. The controls align with the Security by Design and Default principles of strong identity and authentication architecture and secure communication by default, ensuring that update services are verified while protecting update channels. Controls such as trusted sources, redirect validation, and separation of trust domains support the principle of attack surface minimisation, while the combination of metadata validation, integrity checks, and freshness validation reflects defence in depth. Support for automated update discovery and retrieval further aligns with secure-by-default practices for timely vulnerability remediation.

The following table lists the recommendations for this stage:

Recommendation	Description
Trusted update source	Configure the clients to obtain update information only from authorised and authenticated update endpoints.
Strong transport security	Enforce encrypted and authenticated communication channels (e.g. TLS[21]) for all update discovery and retrieval operations, without fallback to insecure protocols.
Controlled redirection handling	Redirects should not be followed unless explicitly validated. For example, clients can allow redirects only to approved update hosts or domains and reject redirects to unknown or unauthorised hosts.
Separation of trust domains	Treat CDNs, mirrors, and other distribution intermediaries as untrusted delivery mechanisms. Their compromise should not be sufficient to cause clients to accept unauthorised or modified update metadata or artefacts.
Integrity verification	Perform initial integrity check on the retrieved artefact (e.g. using hashes from metadata) to detect corruption or tampering during transmission. However, do not rely on retrieved hashes for trust until the metadata containing those hashes has been authenticated
Metadata freshness validation	Update metadata should include freshness information, such as an expiry time, timestamp, version number, or monotonic sequence number. Devices should reject stale, expired, replayed, or otherwise outdated metadata to avoid being kept on older vulnerable versions through replay or freeze attacks.
Controlled update discovery and notification	Configure clients to regularly check for updates and ensure they are able to notify users or systems of available updates in a timely and reliable manner.
Support for automated update retrieval	Products should support automatic discovery and secure retrieval of updates, with security updates enabled by default wherever feasible to ensure timely remediation of vulnerabilities. The users or administrators should be informed of available updates and, where appropriate, provided with clear controls to opt out or temporarily postpone updates, without preventing the timely application of critical security updates.

4.3 Verification and Installation

Discovery and retrieval controls ensure that updates are obtained from trusted sources, while verification and installation controls ensure that only valid and authorised updates are accepted and safely applied. The use of multiple validation steps, including signature verification, integrity checks, and version control, reflects the Secure by Design principle of defence in depth, and restricting which

[21] https://datatracker.ietf.org/doc/html/rfc8446, Accessed 20/05/26.

components can initiate and perform updates supports least privilege, while secure installation processes, atomic update behaviour, and rollback capabilities align with lifecycle management.

The following table lists the recommendations for this stage:

Recommendation	Description
Signature verification	Verify the authenticity of update metadata and ensure that update artefacts are cryptographically bound to that metadata (e.g. via signed hashes)
Applicability validation	Ensure that the downloaded updates are applicable to the specific product, version, and configuration before installation.
Integrity enforcement	Validate the integrity of update artefacts (e.g. using cryptographic hashes) prior to installation and reject any corrupted or modified content.
Version control and anti-rollback	Enforce version checks to prevent installation of outdated or unauthorised versions, unless explicitly permitted through a controlled process. Where applicable, use protected or tamper-evident anti-rollback state, e.g. rollback counters or monotonic sequence numbers, to prevent acceptance of older but validly signed updates.
Authorised installation context	Ensure that only trusted and authorised components can initiate and perform update installation, with appropriate privilege restrictions.
Secure installation process	Perform updates in a controlled manner that prevents partial or inconsistent system states, including protection against interruption during installation.
Atomic update behaviour	Ensure that updates are applied atomically, i.e. the system should transitions fully to the new version or safely remains on the previous version.
Recovery mechanisms	Provide the ability to recover from failed or incomplete updates, including retry, repair, re-download, fallback to a known-good version, or other controlled recovery actions where necessary.

4.4 Observability and Recovery

This stage covers how update outcomes are recorded, communicated, and acted upon after verification and installation. It ensures that failures or anomalies are visible to the user or administrator, and that the product can recover safely from unsuccessful or compromised updates. The controls in this stage align with Secure by Design and Default principles, particularly logging, monitoring, and alerting, incident response and recovery, transparent security posture, lifecycle management, and secure recovery and ownership lifecycle.

The following table lists the recommendations for this stage:

Recommendation	Description
Update status recording	Record the outcome of update operations (e.g. success, failure, rollback, partial installation states...).
Secure logging	Ensure that update related logs are protected against tampering or unauthorised access.
Integrity failure reporting	Detect and report failures in signature verification, integrity checks, or metadata validation.
Update status visibility	Provide visibility of update status to users (or administrators in managed environments).

Update failure communication	Ensure that failed or incomplete updates are clearly communicated to the user or administrator, including clear information to understand the issue and take appropriate action (e.g. retry or investigate).
Corruption detection and handling	Detect corrupted or incomplete update artefacts and ensure they are not installed. Trigger appropriate recovery or re-download mechanisms.
Recovery and rollback mechanisms	Provide the ability to recover from failed updates, including reverting to a known-good version where necessary.
Trust anchor and key recovery	Support secure update or replacement of trusted keys (e.g. for signing) in case of compromise or rotation.

5. Illustrative end-to-end example

5.1 Scenario

This section provides an illustrative example showing how the lifecycle and recommendations may be applied in practice. Taking inspiration from the recent notepad++ incident, specifically the root cause and the implemented countermeasure, we describe a setup where an embedded IoT device, a thermostat, needs to provide updates and ensure that the update is produced, delivered, and installed safely.

Embedded and IoT devices often use in-band update mechanisms, where a device-resident update client interacts directly with vendor-controlled update services, often via distribution intermediaries such as CDNs. This example reflects such an in-band (built-in) update mechanism, where the device contains its own update client that is responsible for discovering, retrieving, verifying, and installing updates.

In this example the firmware is a security update that patches EUVDB-202X-Y, an input validation flaw in the thermostat version 1.0.0.

Note: this example is intended to demonstrate how the recommendations can be implemented in a concrete scenario, and how different controls can, and should, be applied across the update lifecycle. It is not, however, intended to be prescriptive or exhaustive, and it does not represent a complete and universally applicable solution. Implementations will vary depending on system architecture, risk considerations, and operational constraints.

5.2 Setup

The manufacturer operates a root signing authority and a separate operational signing key used for the more routine update releases. The operational signing key is authorised by the root signing authority. This separation allows the root key to be kept offline and used only for authorising signing keys, while the vendor signing key is used operationally and can be rotated or replaced without changing the device's root trust anchor.

The thermostat device has:

- Root public key, provisioned during manufacturing, serving as the trust anchor used to verify authorised signing keys.
- Current vendor signing public key, used to verify signed update metadata. The signing key can be updated or replaced through the update process.
- 2 firmware slots: ***slot_a*** and ***slot_b*** to support A/B updates
- A ***staging*** area for unverified downloads
- A protected state mechanism for version, anti-rollback, and update status information
- A configured TLS trust store (e.g. ***ca.pem***) used to validate the update server's certificate during update retrieval

NOTE: In production environments, private keys are typically protected using hardware-backed mechanisms such as Hardware Security Modules (HSMs), ensuring that keys cannot be extracted and that signing operations are tightly controlled. In some deployment models, particularly in desktop or general-purpose systems, certificate authorities (CA) may be used to establish trust.

Also, this example uses a simplified two-level signing model based on a root authority and an operational signing key, for clarity. Higher-assurance deployments should consider more advanced role-separated designs, such as TUF or Uptane, which can provide additional signing roles, delegated metadata, freshness and consistency protections, and threshold signatures where appropriate.

5.3 Security control implementation across stages

5.3.1 Preparation and publication

The manufacturer creates the firmware source files and packages them into ***vendor/build/my-acme-thermostat-1.1.0.tar.gz***. The firmware is a security update that is built and released independently of functional changes to allow prompt deployment.

The build process is executed in a controlled environment with restricted access. Only authorised code and dependencies are included. Third-party components are verified before release through the generation of an SBOM on which vulnerability checks are run. Before release, the update is also tested to confirm that the security patch functions as intended and does not introduce reliability, security, or compatibility problems.

A manifest is generated, containing properties of the update including:

- **Artefact integrity information:** SHA-256 hash and file size of the update package. Any byte-level modification to the package, whether accidental or malicious, will produce a different hash or size mismatch, which will be detected during subsequent on-device verification.
- **Applicability and version information: product** model, target version, supported current versions, and update type (security update).
- **Freshness and anti-rollback information:** timestamp, expiry time, rollback counter.
- **Advisory information:** severity, advisory URL and summary.

The manufacturer also signs the manifest to produce a signature using the vendor signing private key, e.g.:

- ***openssl dgst -sha256 -sign keys/vendor_private.pem -out repo/manifest.json.sig repo/manifest.json***

Note: in production a Hardware Security Module (HSM) would provide high-assurance signing, ensuring cryptographic keys never leave the hardware.

The manufacturer then publishes:

- ***manifest.json***
- ***manifest.json.sig***

- ***my-acme-thermostat-1.1.0.tar.gz***

to the update server or CDN.

Where applicable, additional artefacts may be included in the release, including for example an updated vendor signing key and its authorisation signature to support key rotation.

5.3.2 Discovery and retrieval

The update client running on the thermostat checks for updates by retrieving the manifest and comparing the version on the update server against the currently installed version stored in '***device/state/installed_version.txt'***.

The thermostat validates that it is communicating with the intended update repository by establishing an HTTPS connection (using TLS). The server certificate is validated against a configured trust store, for example:

curl --tlsv1.2 --cacert /etc/ssl/certs/ca.pem https://updates.acme.com/device/manifest.json -o manifest.json

This step can be further strengthened by restricting which certificates are accepted. For example, by only trusting a vendor-controlled certificate authority, or to verify that the server presents a specific expected public key (certificate pinning).

The ***'update_type'*** and ***'severity'*** fields in the manifest allow the client to distinguish a security update from a routine functional release and to notify users or administrators of the urgency accordingly.

When the update client finds a relevant update, it downloads the update files and stores them in a '***staging***' directory. So far, nothing in the active firmware execution path is touched. If power is lost mid-download the device simply retries later and the running firmware is not affected.

The retrieved metadata is treated as untrusted until verification in the next step.

5.3.3 Verification and installation

Before installation, the device performs the following checks:

- Signing key trust check:
  - E.g.: ***openssl dgst -sha256 -verify root_public.pem -signature vendor_public.pem.sig vendor_public.pem***
- Signature check:
  - E.g.: ***openssl dgst -sha256 -verify vendor_public.pem***
- Hash check:
  - E.g.: ***sha256sum*** of the downloaded artifact vs '***sha256***' field in the manifest
- Anti-rollback check:
  - E.g.: manifest's ***rollback_counter*** >= the ***rollback_counter***
  - ***rollback_counter*** is maintained by the secure bootloader in protected non-volatile storage and is increased only after the new firmware has booted successfully and passed health checks.
- Applicability check:
  - E.g.: device verifies applicable model, version, timestamp, and configuration.

If any check fails, the installation is aborted.

If the checks are successful, the update is written to the inactive firmware slot. This A/B strategy ensures that a known good version always remains available. Activation is completed through an atomic slot switch, avoiding partial installation states and supporting recovery if the new firmware fails.

5.3.4 Observability and Recovery

Failures in signature verification, integrity checks, or metadata validation are logged and reported. Also, clear output messages are shown in the console and logs, including the reason for failure.

The installation script records the outcome of each update attempt in ***device/state/update.log***, including the timestamp and status (e.g. update started, verification failed, installation successful, or rollback triggered). This provides a traceable record of update operations.

Update logs are written and stored with restricted file permissions to prevent unauthorised access or modification.

In cases where a vendor signing key is compromised, the manufacturer can generate a new signing key and distribute it through a signed update. Devices will only accept new keys that are authorised by the trusted root key, preventing unauthorised keys from being introduced.

The root key itself is not updated through this mechanism. In the event of root key compromise, recovery would require a separate secure provisioning process (e.g. secure out-of-band update, firmware reinstallation, or factory reset).

5.4 Security Claims and Evidence

In this example, the manufacturer makes the following security claims regarding the update mechanism of its thermostat when placing the product on the market and throughout its supported lifecycle. These claims reflect the security properties achieved through the implementation of the controls defined in Section 4.

Example evidence is provided with each claim to show how the claim could be supported in practice, together with example artefacts that may demonstrate the relevant controls or outcomes. This approach is aligned with ENISA's Secure by Design and Secure by Default playbook[22], which promotes the use of structured, and where appropriate machine-readable, claims supported by evidence.

The example evidence and artefacts listed below are illustrative only and are derived from the implementation described above. They are not meant to be prescriptive or exhaustive. Actual evidence will vary depending on system architecture, operational processes, and deployment models.

Control Stage	Security claim	Example evidence	Example artefacts
Preparation and publication	The origin and composition of the update can be trusted	Controlled build; dependency checks; SBOM	Build logs; SBOM; SCA/SAST results; CI/CD records
	The update is protected from unauthorised release or impersonation	Signed manifest; root-authorised signing key; trust chain; controlled signing process	manifest.json.sig; vendor_public.pem.sig; root_public.pem, signing logs

[22] https://www.enisa.europa.eu/sites/default/files/2026-03/ENISA_Secure_By_Design_and_Default_Playbook_v0.4_draft_for_consultation.pdf

	Update metadata and artefacts are integrity protected and inconsistencies are detectable	SHA-256 hash in manifest; signed metadata	manifest.json; firmware package
	Security fixes can be deployed without unnecessary operational delay	Security-only release; advisory metadata	Release notes; manifest fields
Discovery and retrieval	The communication channel is authenticated and private	HTTPS/TLS; certificate validation; trust store	ca.pem; TLS settings
	The system can withstand a compromise of the distribution network	signed metadata verified; modified artefacts rejected before activation	staging/; downloaded files
	The system is protected against being frozen on a vulnerable version	Version comparison; freshness checks; expiry, timestamp, or sequence validation	manifest version/timestamp/expiry; protected version state; freshness validation logs
Verification and installation	The device verifies authenticity before installation or activation	Root verifies vendor key; vendor key verifies manifest	Public keys; signature files; verification logs
	The device is protected against downgrade attacks	Version and anti-rollback check	anti-rollback state; manifest version, counter, or sequence number
	Update integrity is verified prior to installation	SHA-256 comparison; install blocked on failure	Manifest hash; checksum output
	The transition to the new version is safe and stable	A/B slots; atomic slot switch; fallback retained; known-good state set after health check	slot_a; slot_b; boot status; health-check logs; state records
Observability and Recovery	Update failures or tampering are detected and reported	Failure logging; console alerts; status records	update.log; console output
	The system can recover from a failed update state	Health-check; fallback or controlled rollback; recovery event	Health-check logs; recovery or rollback record
	Trusted signing keys can be rotated, and root trust-anchor recovery is handled through a separate secure recovery process	Vendor signing-key rotation; root-authorised key replacement; secure reprovisioning for root compromise	New signing key package; key authorisation signature; reprovisioning records

A Annex 1: Stride mapping of update lifecycle threats

This annex provides a mapping of the update lifecycle threats described in Section 3 to the STRIDE threat modelling categories. The mapping is not meant to be exhaustive but is intended to support a structured review . Several threats may span more than one STRIDE category depending on the specific implementation.

Lifecycle stage	Threats	STRIDE categories
Build/Package/Establish trust	Build pipeline or signing key compromise	Tampering, Spoofing, Elevation of privilege
Publish	Metadata or payload manipulation	Tampering, Spoofing, Denial of service
Check for updates	Redirection, fake update source, replay or freeze attack	Spoofing, Tampering, Denial of service
Retrieve	Payload substitution or malicious artefact retrieval	Tampering, Spoofing, Denial of service
Verify	Improper or missing verification	Spoofing, Tampering
Install	System compromise or installation failure	Elevation of privilege, Tampering, Denial of service
Record status	Failure not detected, reported, or acted upon	Repudiation, Tampering, Denial of service

ABOUT ENISA

The European Union Agency for Cybersecurity, ENISA, is the Union's agency dedicated to achieving a high common level of cybersecurity across Europe. Established in 2004 and strengthened by the EU Cybersecurity Act, the European Union Agency for Cybersecurity contributes to EU cyber policy, enhances the trustworthiness of ICT products, services and processes with cybersecurity certification schemes, cooperates with Member States and EU bodies, and helps Europe prepare for the cyber challenges of tomorrow. Through knowledge sharing, capacity building and awareness raising, the Agency works together with its key stakeholders to strengthen trust in the connected economy, to boost resilience of the Union's infrastructure, and, ultimately, to keep Europe's society and citizens digitally secure. More information about ENISA and its work can be found here: www.enisa.europa.eu.

ENISA
European Union Agency for Cybersecurity

Athens Office
Agamemnonos 14
Chalandri 15231, Attiki, Greece

Brussels Office
Rue de la Loi 107
1049 Brussels, Belgium

enisa.europa.eu

Publications Office
of the European Union