TLP - CLEAR

EUROPEAN UNION AGENCY
FOR CYBERSECURITY

ENISA Security by Design and Default Playbook

A Practical Guide to Security by Design and Default for SMEs

MARCH 2026

Document History

//DRAFT ONLY - DELETE THIS SECTION AND PAGE UPON FINAL PUBLICATION

Date	Version	Modification	Author
19/03/2026	0.4	Version for public consultation	ENISA

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

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

LEGAL NOTICE

COPYRIGHT NOTICE

Generative AI was used in a limited capacity to support language refinement and preliminary document screening. All outputs were reviewed and validated by subject-matter experts. No AI-generated content was used without substantive human oversight.

Table of Contents

Executive Summary

Modern products with digital elements are increasingly being expected to be *Secure by Design* and *Secure by Default*. However, many organisations, in particular SMEs where development teams are small and security expertise can be limited, may face distinct challenges in applying these concepts consistently, requiring targeted solutions.

This document puts forward a set of principles and tangible guidance on the application of *Security by Design* and *Default* across the lifecycle of a product. In particular, the report focuses on explaining these principles in clear, repeatable actions that can be applied to existing engineering, product, and release processes.

The *Security by Design* principles are organised into two groups, namely **Architectural Foundations** and **Operational Integrity.** The former address how the system is designed and built, while the latter focuses on how the system is managed and maintained. Similarly, the *Security by Default* principles are grouped into **Default Hardening** and **Guided Protection**. The former ensures that products start in a secure and restrictive state, while the latter aims at supporting users in maintaining the secure baseline through clear defaults, warnings, and recovery mechanisms.

A core part of this document is a set of practical *Secure by Design* and *Default* playbooks. Each playbook presents the principle's objective, practical actions, and a set of evidence that could support the demonstration of its implementation. Annex C provides an indicative mapping between the principles presented in this report and the Essential Requirements set out in Annex I of the EU Cyber Resilience Act (CRA).

The report also presents an illustrative example of a Machine-Readable Security Manifest (MRSM), a voluntary, manufacturer-issued artifact designed to provide a high-fidelity record of a product's security posture. Rather than proposing a new schema or prescribing a specific format, the example illustrates how machine-readable security attestations could express security claims, supporting evidence, and verification results in a structured format, enabling automated processing and validation.

This approach also allows for the automation of security checks and release decisions, which helps to ensure that the Security by Design and Default properties are maintained over time. This capability is particularly valuable for SMEs with limited security resources because it reduces the need for manual audits while increasing confidence in the product's security posture.

SECTION 1

Introduction

1. Introduction

1.1 Objectives

Security by Design represents a fundamental shift in product security, embedding protective measures from conception rather than retrofitting them post-development. For micro, small, and medium-sized enterprises (SMEs) manufacturing products, this transition presents distinct challenges requiring tailored solutions[1] [2].

Security by Design is increasingly expected, including under the CRA[3]. This report does not provide legal guidance, but instead offers practical, technically grounded approaches that can support manufacturers in applying these principles in practice. For reference, Annex C provides an indicative mapping between the Secure by Design and Default principles described in this report and the Essential Requirements set out in Annex I of the CRA.

The report aims to bridge the gap between aspirational security principles and practical implementation within SME constraints, providing guidance that development teams can immediately apply during design, build, and deployment phases.

The report also addresses the persistent challenge SMEs face in translating security principles into engineering practice, acknowledging limitations in time, budget, expertise, and resources. It provides structured, easy-to-follow checklists enabling organisations to:

- **Identify relevant security controls (quick wins)** - determine a priority set of protective measures that align with specific product requirements;
- **Implement controls systematically (approachability)** - apply security measures through technically feasible approaches;
- **Enable continuous improvement** - establish measurable baselines for security enhancement.

1.2 Scope and methodology

This guidance was developed with SMEs, including micro-enterprises, in mind, particularly SME manufacturers developing products with digital elements, encompassing embedded software, IoT devices, connected systems, standalone software, and any hardware incorporating programmable components. It addresses security requirements throughout the product lifecycle from product design through decommissioning.

[1] For a more general overview of cybersecurity challenges faced by SMEs, see ENISA Cybersecurity for SMEs – Challenges and Recommendations, 2021: https://www.enisa.europa.eu/publications/enisa-report-cybersecurity-for-smes
[2] Alladean Chidukwani, Sebastian Zander, Polychronis Koutsakis, *Cybersecurity preparedness of small-to-medium businesses: A Western Australia study with broader implications*, Computers & Security, Volume 145, 2024, https://doi.org/10.1016/j.cose.2024.104026.
[3] https://eur-lex.europa.eu/eli/reg/2024/2847/oj

SMEs are defined as organisations with fewer than 250 employees and annual turnover below €50 million[4][5]. These manufacturers typically face[6]:

- Budget constraints - limited financial resources for security investments[7];
- Limited expertise - no dedicated security personnel; reliance on general IT staff;
- Skills gaps - minimal specialist security knowledge within the organisation;
- Time pressures - security competing with core business priorities.

For the purpose of this report, ENISA performed an analysis of existing security frameworks, such as those previously published by ENISA and other EU-based cybersecurity agencies, guidance from NIST and OWASP[8]. We've identified common requirements and implementation patterns which were evaluated against SME capabilities to determine feasibility and adaptation requirements.

While this document can support a solid technical foundation, it serves as introductory guidance rather than a comprehensive compliance manual. Adherence to the principles outlined in this document does not inherently ensure certification against specific international standards or regulatory mandates. Instead, it acts as a foundational bridge to help teams navigate the complexities of secure product development.

1.3 Target audience

This guidance is structured as a technical companion for software and product developers, systems engineers, and technical leads. It is intended for those tasked with the practical implementation of Security by Design and Default principles within their product development lifecycles.

The primary groups that will benefit from this guidance include:

- **Software Developers & Engineers**: Professionals seeking tactical methods to embed security into the codebase while operating within rapid delivery cycles.
- **Technical Product Managers**: Individuals responsible for balancing functional requirements with the necessity of fundamental security resilience.
- **SME Security Leads**: Personnel tasked with interpreting enterprise-grade frameworks for organisations with limited budgets, specialised niches, or leaner teams.
- **System Architects**: Designers aiming to build robust infrastructures that prioritise security from the initial conceptual phase.

1.4 Report structure

The report has the following structure:

- **Section 2** covers the product security lifecycle, and practical ways in which risk management activities and threat modelling can be applied in the context of an SME, the section includes a series of simple to follow steps on how each of the activities can led to tangible deliverables.
- **Section 3** covers fundamental *security by design* and *security by default* principles. They are explained in an easy-to-understand manner. This section lays the foundation for the next chapters. The report also includes a mapping on how these principles relate to other ENISA

[4] https://eur-lex.europa.eu/EN/legal-content/glossary/small-and-medium-sized-enterprises.html
[5] https://single-market-economy.ec.europa.eu/smes/sme-fundamentals/sme-definition_en
[6] https://www.enisa.europa.eu/sites/default/files/publications/ENISA%20Report%20-%20Cybersecurity%20for%20SMES%20Challenges%20and%20Recommendations.pdf
[7] https://www.enisa.europa.eu/sites/default/files/publications/CSPA%20-%20NIS%20Investments%20-%202023.pdf
[8] For a complete list, of references used for preparing this document check the appendix (Bibliography).

security best practices and well-established resources, such as *MITRE CWE* and *OWASP Top 10*.

- **Section 4**, the most extensive part of the report, details each of the **22** principles by covering their individual objective, key elements to consider when applying it in an engineering context, means to collect evidence that the principle has been followed, and a set of criteria that can be used in a release review.
- **Section 5** covers the concept of a Machine-Readable Attestation, using an example Machine-readable Security Manifest (MRSM) approach, where a hierarchical data model that ensures every high-level security claim is backed by granular technical evidence. To support the concept, we provide a short example/scenario on how MRSM could be implemented in practice.
- **Annexes** summarise the CRA essential cybersecurity requirements, providing a mapping showing how Secure by Design and Default principles described in this document can support their implementation

SECTION 2

Secure by Design and Default across a Product's Lifecycle

2. Secure by Design and Default across a Product's Lifecycle

Security by Design and Security by Default require more than applying principles during development. They must be operationalised end-to-end across the full product lifecycle, from the initial concept up until the product's eventual decommissioning. This lifecycle view is especially important for connected products, where evolving threats, supply-chain dependencies, and long-lived deployments can erode otherwise sound designs if governance and assurance do not persist over time.

ENISA's Report on Guidelines for Securing IoT highlight a common failure node "*security goals can often fail, even in the presence of good design, if there is a lack of tools that enable stakeholders to understand and assess security issues*.[9]" In practice, Secure by Design and Default depends on both good engineering decisions and the organisational mechanisms (methods, artefacts, metrics, and review gates) that make risk visible, decisions repeatable, and trade-offs explicit.

2.1 Product Lifecycle

As highlighted in the ENISA report on basic security recommendations for IoT[10] [11], security must be considered throughout the entire product lifecycle. Regardless of the production model used (V-model or Agile), the following lifecycle phases require explicit security consideration to ensure product security:

- **Requirements**: the foundation for the development cycle. During this phase user, business and functional requirements are defined. These requirements reflect the intended use of the software and will be translated to specifications that will guide design, development and maintenance/deployment decisions at a later stage.
- **Design**: During the design phase the architecture and the design of the solution are created. This phase involves the creation of a set of documents that describe how the user/business and functional requirements will be translated to system specifications and essentially how the product will work.
- **Development/Implementation:** during the development/Implementation phase, specifications and software design diagrams written in an appropriate notation are transposed into code.
- **Testing and acceptance**: The testing and acceptance phase involves all necessary steps to verify that the developed software actually meets the identified requirements and design principles of the previous phases.
- **Deployment and integration**: The deployment and integration phase follow the acceptance of the software subject to successful testing in the previous phase, i.e., after it has been approved for release. It involves integrating all necessary elements of the solution in the production environment and its deployment.
- **Maintenance and disposal**: solutions deployed in production needs to be constantly maintained to ensure availability and integrity of the provided functionality. When the solution

[9] https://www.enisa.europa.eu/sites/default/files/publications/ENISA%20Report%20-%20Guidelines%20for%20Securing%20the%20Internet%20of%20Things.pdf
[10] https://www.enisa.europa.eu/publications/baseline-security-recommendations-for-iot
[11] https://www.enisa.europa.eu/sites/default/files/publications/WP2019%20-%20O.1.1.1%20Good%20practices%20for%20security%20of%20IoT.pdf

becomes obsolete, it is important to ensure a secure disposal to preserve privacy management, providing data erasure mechanisms.

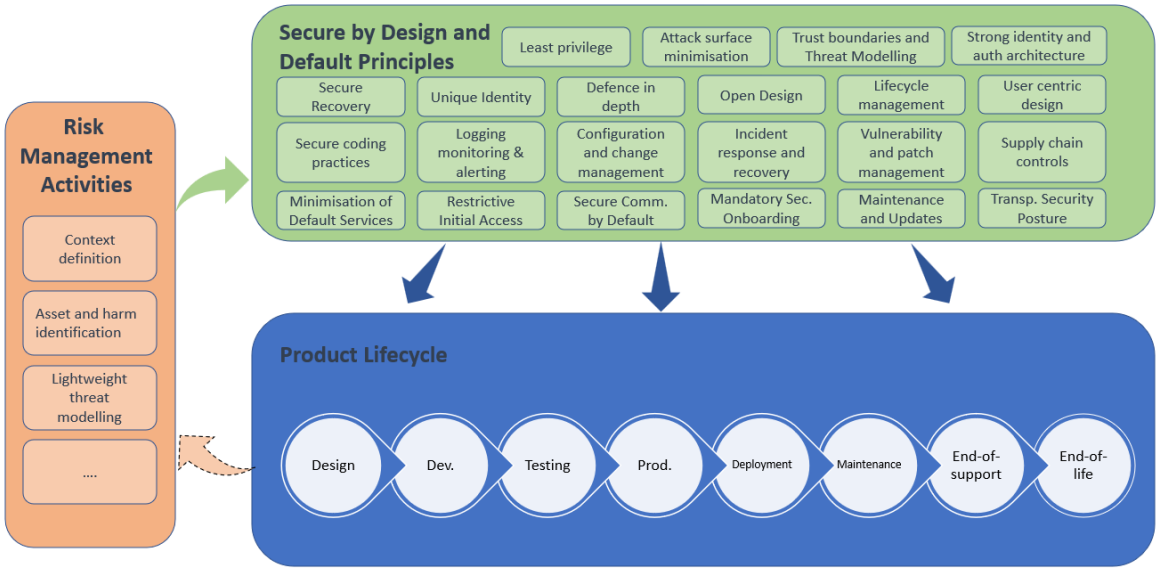


Figure 1: Secure by design and default across a product's lifecycle

Figure 1 shows the relationships between the product lifecycle, Secure by Design and Default principles, and risk management activities.

Risk management activities establish the security context by clarifying important considerations, including what needs to be protected, the threats of concern, and the acceptable level of residual risk given the product's intended purpose and security impact. Secure by Design and Default principles translate this context into practical security decisions that are applied throughout the phases of the product lifecycle, from initial design through to end-of-life.

Risk management activities should be revisited in response to changes or events arising during phases of the product lifecycle. Similarly, while the product lifecycle phases are presented sequentially, both the lifecycle and the application of Secure by Design and Default principles across the phases are inherently iterative. Events or findings in later phases, such as testing results, deployment in new environments,

incidents, discovered vulnerabilities, or the implementation of new features, often require re-entry into earlier lifecycle phases.

For SMEs, especially those operating with rapid iterations, the key is to keep the lifecycle lightweight, risk-driven, and automation-first:

- Use small, reusable artefacts (one-page context, simple diagrams, checklists);
- Prefer automated controls in CI/CD over manual reviews, reserving deep review for high-risk changes.
- Introduce fast security gates aligned to existing agile ceremonies (Definition of Ready/Done, PR checks, release checklist).

Table 1 - Cybersecurity activities product lifecycle (SMEs)

Phase	Actions	Deliverable
Requirements	Define product context (users, environments, data), "non-negotiable" security defaults, and top risks/abuse cases; establish clear criteria for addressing risks based on the product's intended purpose, expected use, and security impact.	1-page **Security Context & Assumptions** + short **Security Requirements Checklist** (including secure defaults).
Design	Maintain one architecture diagram with **trust boundaries**; do a lightweight threat model on the top 5–10 abuse cases; decide the critical design controls (authn/authz[12], update mechanism, secrets, logging).	**Architecture + trust-boundary diagram** + **Top threats & mitigations** (bulleted list).
Development / Implementation	Build secure defaults into code/config; enforce dependency hygiene; protect secrets; require PR review for security-sensitive changes; automate SAST/dependency scanning as part of CI. environments (Agile/DevOps/DevSecOps).	CI evidence (pipeline logs) + lightweight **Secure coding / PR checklist** (often a repo file).
Testing & acceptance	Run automated security checks (SAST/dependency, basic DAST where relevant); ensure default configuration is validated; run targeted pen test when potential risk triggers hit (e.g. in the case of a substantial modification). Note: Security tests should be integrated into developer workflows and CI pipelines as early as possible (shift-left). Testing/Acceptance serves as a validation checkpoint, not the primary execution point for these controls.	**Release security checklist** (pass/fail + exceptions) + documented **known issues/residual risk**.
Deployment & integration	Ensure secure provisioning/enrolment, least-privilege runtime config, and monitoring of key "health/security" indicators; treat updates as controlled change management.	**Deployment hardening checklist** + **Rollback plan** + minimal **monitoring/alert list**.
Maintenance & disposal	Define patch intake + SLAs, vulnerability monitoring, incident handling, and an end-of-support/EOL plan; ensure secure disposal (data erasure, credential revocation).	**Vuln & patch process** (1 page) + **EOL/disposal note** + maintained **risk register** updates.

[12] https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/05-Authorization_Testing/README

2.2 Risk Management Activities

Risk management activities provide the foundation for implementing Secure by Design and Default across the product lifecycle. They ensure that security decisions are based on an informed and shared understanding of what needs to be protected, from whom, and under what constraints.

These activities establish the context within which Secure by Design and Default principles are applied. Their output directly informs:

- security requirements and architecture decisions;
- default configurations and “out-of-the-box” hardening;
- prioritisation of controls and assurance activities (e.g., testing depth, review gates);
- acceptable trade-offs inherent to the product’s intended use and deployment context; and
- ongoing operational posture, including patching and end-of-life handling.

Risk management is typically initiated early, but it must be iterative: revisited at defined lifecycle gates (e.g., major release, supplier change, new deployment context) and triggered by significant events (e.g., newly disclosed vulnerabilities, threat shifts, incident learnings). Examples of risk management activities directly supporting Secure by Design and Default decisions include:

- Product context definition, including intended purpose, operating environment, users, data processed, and constraints
- Establishing risk acceptance criteria, expressed as simple and practical guidelines to support consistent decision making
- High-level risk assessment focused on threat modelling, key assets, trust boundaries, and risk treatment decisions

This report does not aim to define or replace a formal risk management framework. Rather it aims to highlight common risk management activities and outputs that feed security decision making. To support

SMEs, the following list is a minimum set of activities that can drive Secure by Design and Default decisions without creating heavy process:

Table 2 - Risk management activities for SMEs

Activity	*Actions*	*Deliverable*
Product context & scope	Define intended use, deployment environments, user/admin roles, data types/sensitivity, and key external dependencies (cloud, mobile app, third parties).	1–2 page "Product Security Context" note (scope, assumptions, dependencies).
Asset & harm identification	List top data, hardware, or function assets (e.g., credentials, customer data, essential functions) and the key harm outcomes (privacy breach, takeover, outage, fraud, safety impact).	Asset list + "Top harms" list (one page).
Lightweight threat modelling	See section 2.3	See section 2.3
Risk register	Record 10–30 risks with likelihood/impact (simple scale), owner, treatment, status; link high risks to backlog items/controls.	Living risk register (spreadsheet or ticket board).
Risk acceptance criteria	Define a set of non-negotiable risk conditions (e.g. misuse of software updates, unauthorised administrative access, or exploitation of default credentials is not acceptable), and establish criteria for accepting residual risks. For example, the accepted residual risks should not undermine the essential cybersecurity requirements, given the product's intended purpose and reasonably foreseeable use.	1-page Risk Acceptance & Exceptions policy.
Security requirements baseline	Translate top risks into testable "must" requirements (authn/authz, secure defaults, secrets, encryption, logging, updates).	SME security requirements checklist (testable controls).
Release risk review gate	As part of the secure development lifecycle, include a formal pre-release gate to verify compliance with the requirements defined above. This review should confirm checklist met, defaults verified, known vulns triaged, high risks treated/accepted with rationale; decide go/no-go.	Release security review record (ticket/comment) + documented exceptions.
Change-triggered reassessment	Re-run context/threat/risk steps when major changes occur (architecture, auth model, data, critical dependency/supplier, deployment environment), including changes that could be considered substantial modifications, or after incidents.	Updated context note, threat shortlist, and risk register entries (with date).

2.3 Threat modelling

Threat modelling is a structured, repeatable form of risk assessment that helps teams identify credible attack paths, prioritise risks, and decide which controls (including secure-by-default settings) are necessary: It assumes that potential threats, such as structural vulnerabilities, can be identified, enumerated, and prioritised. Most commonly, the STRIDE (Spoofing, Tampering, Information

Disclosure, Repudiation, Denial of Service and Elevation of Privilege) methodology is used to identify and classify threats.[13].

Threat modelling should align with widely accepted threat modelling principles[14], recognising that it is a collaborative and iterative activity focused on understanding and reducing real security risks, rather than simply producing documentation artefacts. Common anti-patterns to avoid include treating threat modelling as a one-off compliance exercise, over-engineering models that do not influence design or secure-by-default decisions, or failing to review the model following substantial product modifications or changes in the threat landscape.

For SMEs, particularly those developing products intended for non-critical or lower-risk environments, the objective is not exhaustive analysis; it is a "minimum viable" model that is fast to produce, easy to refresh, and tightly coupled to delivery (architecture decisions, default configuration, and release gates). Where products are intended for higher-risk or critical use cases, depending on the product's intended use and associated risks, a more comprehensive analysis will be required.

Table 3 - Threat modelling for SMEs

ACTIVITY	ACTIONS	DELIVERABLE
Define scope, assumptions, and security objectives	Time-box a short scoping step to make the exercise decision-focused. Capture what is in/out of scope (device/app/cloud/admin tools), the deployment context(s), and the assumptions/constraints you are relying on (e.g., "device may be physically accessible," "customer network is untrusted," "cloud APIs are internet-exposed", "advanced user capabilities"). Then state the security objectives that matter for this product (confidentiality/integrity/availability, plus privacy/safety if applicable).	1 page (or equivalent ticket/wiki) "Threat Model Scope & Objectives" covering: ▪ Purpose (what decisions this will drive) ▪ Scope boundaries (components + environments) ▪ Assumptions/constraints ▪ Security objectives and "crown jewels" (what must not fail)
Model the system at a useful level of abstraction	Produce a single, simple architecture / data-flow view that answers OWASP's core question "what are we building?[15]" In SMEs, the fastest high-value approach is a DFD-style diagram that shows main components, data stores, external entities, and the key data flows/entry points, enough to reason about exposure without	Diagram covering: ▪ Main components (device, app, cloud services, APIs) ▪ External entities (users, admins, third-party services) ▪ Data stores (device storage, cloud DB, logs) ▪ Entry points (APIs, admin UI, update channel, local ports)

[13] https://www.enisa.europa.eu/sites/default/files/publications/WP2019%20-%20O.1.1.1%20Good%20practices%20for%20security%20of%20IoT.pdf
[14] https://www.threatmodelingmanifesto.org/
[15] https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html

	overthinking it. Consider using the diagram from Table 2.	
Mark trust boundaries and privilege paths; identify key assets	Annotate the diagram with (a) trust boundaries[16] [17] (Boundaries between environments with different security properties) and (b) the highest-privilege operations (e.g., firmware/OTA update, remote admin, key provisioning, identity issuance). This is the step that turns "architecture" into "security-relevant architecture."	Diagram: ▪ Trust boundaries (internet - backend, device - cloud, user - admin, tenant - tenant) ▪ Privileged paths (updates, auth, key management, admin actions) ▪ Top assets (credentials/keys, sensitive data, device control functions, availability)
Identify and prioritise the top threats (abuse cases)	Generate a short list of realistic abuse cases mapped to entry points and boundaries (e.g., "credential stuffing → account takeover → remote control," "malicious update," "API authorization bypass," "MITM on onboarding"). Then rank them with a lightweight scheme (High/Med/Low) based on impact and plausibility. OWASP[18] describes this as threat identification and ranking as a core step in most threat modelling approaches.	Top threats table: ▪ 5–10 abuse cases tied to specific entry points/boundaries ▪ Simple likelihood + impact assessment (H/M/L is sufficient) ▪ "Top risks" list that drives design/default choices
Define mitigations, secure defaults, and verification; set refresh triggers	For each top threat, specify the mitigation strategy, the required control(s), and the secure-by-default setting that should ship (e.g., "admin interface disabled by default," "no default passwords," "signed updates enforced," "least privilege roles," "authentication attempts rate-limited") [14]. Then map each control to how it will be verified (CI checks, tests, configuration validation, release gate). Finally, define the triggers that require re-running the model (new internet-exposed interface, new auth model, new sensitive	Controls, Defaults and Verification checklist: ▪ Control/default decision per top threat ▪ Test/verification mapping (automation-first) ▪ Explicit refresh triggers (so the model stays current)

[16] https://owasp.org/www-community/Threat_Modeling_Process
[17] https://www.enisa.europa.eu/sites/default/files/publications/WP2019%20-%20O.1.1.1%20Good%20practices%20for%20security%20of%20IoT.pdf
[18] https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html

	data, new critical dependency, major architecture change).

SECTION 3

Principles

3. Secure by design and default principles

Secure by Design and *Secure by Default* provide a framework for building systems that are resilient and security harden from the start and that remain so once released into real-world environments. ENISA's *Good Practices for Security of IoT (SDLC)*[19] report describes Security by Design as a "holistic approach" that must be applied throughout the entire lifecycle of a product or service.

Security is not a static state, but a continuous process involving specific development guidelines, threat modelling, and the integration of security controls at the earliest stages of design to minimise the impact of cyberattacks.

- **Security by Design** embeds protective measures into products during development rather than adding them retrospectively. This includes threat modelling, secure architecture patterns, validated cryptography, and systematic vulnerability management integrated into development processes.
- **Security by Default** ensures products ship with the most secure configuration reasonably possible. Users should not need technical expertise to achieve baseline security - protective measures must be active upon installation, with any reduction requiring deliberate user action.

3.1 Security by Design Principles

Security by design focuses on incorporating security principles from the earliest stages of development. It requires organisations to embed security into the structure, logic and behaviour of a system rather than treating it as an afterthought. We distinguish two categories distinguish relevant for how 1) a system is built (*Architectural Foundations*) and 2) how it is managed and maintained (*Operational Integrity*)[20].

[19] ENISA, Good Practices of Security of IoT (SDLC), Nov 2019.
[20] for other frameworks and models see - NIST SP 800-160 Vol. 1 Rev. 1 https://csrc.nist.gov/pubs/sp/800/160/v1/r1/final and OWASP SAMM model https://owaspsamm.org/model/

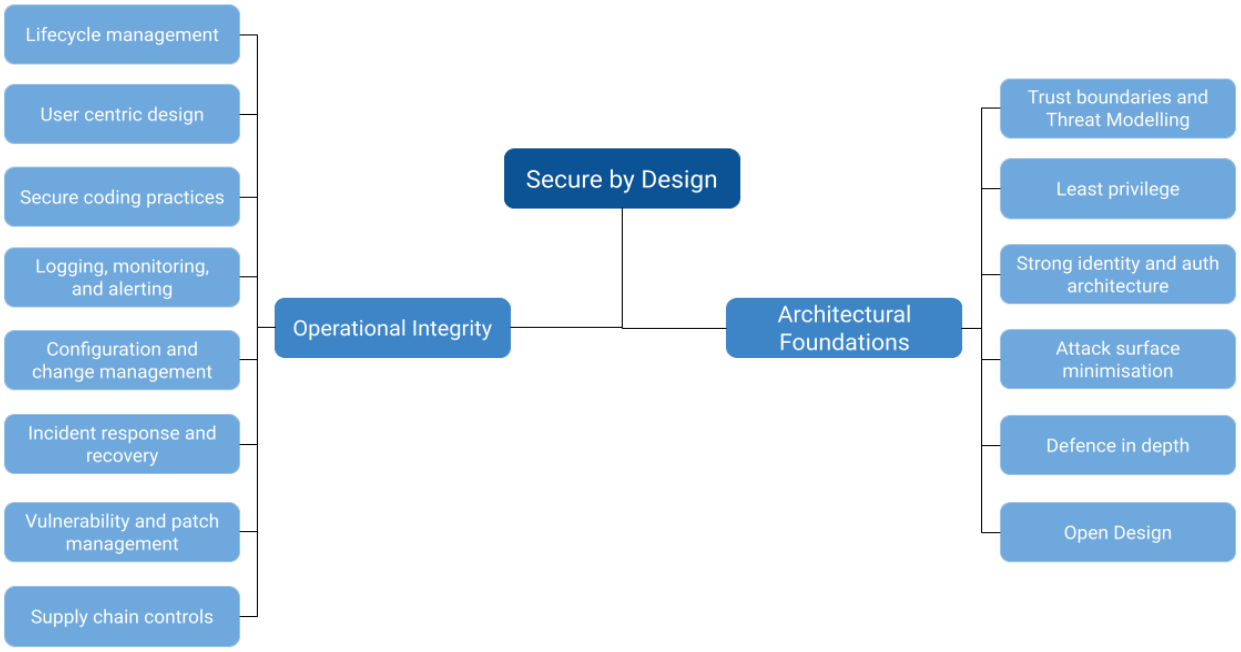


Figure 2: Secure by Design Principles

3.1.1 Architectural Foundations

Architectural Foundations are the "blueprints" of a system's security, that focus on the structural design choices that make a product inherently difficult to compromise or exploit. This category establishes the fundamental rules for how data flows, how components interact, and how the system contains a potential breach before it can spread.

The following principles fall under this category:

- **Trust boundaries and threat modelling**: Secure architectures make trust explicit rather than assumed. Trust boundaries define where data, identities, and execution contexts cross from a more trusted domain to a less trusted one (or vice versa), such as between a device and a cloud service, a user and an admin interface, or one microservice and another. Boundaries clearly identify which components must authenticate each other, where input must be treated as untrusted, and where additional controls (for example, validation, rate limiting, encryption, or isolation) are required. Threat modelling also provides a structured way to identify what could go wrong at those boundaries by mapping key assets (e.g., credentials, firmware images, customer data), likely threat actors, and realistic attacker capabilities and assumptions.
- **Least privilege:** Systems and users should be granted only the feasible minimum level of access required to perform their functions. Restricting permissions reduces the impact and limits the scope of compromise. Least privilege should be applied consistently across user accounts, service accounts, APIs, and administrative roles, and privileges should only be elevated when needed and for the shortest feasible duration.
- **Strong identity and authentication architecture**: A secure product architecture requires a clear and consistent approach for how identities are created, verified, and managed for users, devices, services, and administrators. This includes defining authoritative identity sources, establishing how authentication occurs across interfaces (for example, web portals, APIs, local

management consoles, and device-to-cloud communication), and ensuring that authentication is resistant to common attacks such as credential stuffing, replay, and session hijacking.

- **Attack surface minimisation:** Unnecessary features, services and interfaces increase the number of potential attack vectors. Therefore, reducing system complexity and disabling unused or unnecessary components reduces the likelihood of vulnerabilities being introduced or exploited. This includes removing default accounts, uninstalling unused packages, closing nonessential ports, removing unused code, and limiting exposed management interfaces to trusted networks only. Standardised secure baselines and hardened configuration templates make it easier to keep systems consistently minimal as they scale. Ongoing vulnerability scanning and asset inventory are also important, because you cannot minimise what you do not know exists.
- **Defence in depth:** Applying layered security controls ensures that the failure of a single mechanism does not result in complete compromise. Layers can include preventive controls (e.g., MFA, network segmentation), detective controls (e.g., monitoring, logging, anomaly detection), and corrective controls (e.g., automated isolation, backup/restore). Effective defence in depth also assumes that some controls will be bypassed, so systems should be designed to degrade gracefully and still protect critical assets. This approach reduces reliance on any single "silver bullet" and increases the attacker's cost and time to achieve their objectives. It is strongest when controls are diverse (not all dependent on the same technology or trust assumption).
- **Open Design (Avoiding Obscurity):** Systems should not depend on secrecy of design or hidden behaviour for protection. Security controls should remain effective even if an adversary understands how the system operates. This principle encourages the use of well-studied algorithms and protocols, clear documentation, and designs that can withstand scrutiny through review and testing. "Open design" does not mean making secrets public, but rather that the security should rest on protected keys, strong authentication, and robust implementation, not on keeping the mechanism itself hidden. In practice, it also supports maintainability, because transparent designs are easier to audit, validate, and improve over time.

3.1.2 Operational Integrity

Operational Integrity addresses the human and procedural elements that keep a system secure from the first line of code to its eventual retirement. It ensures that security is treated as a continuous, lived experience for developers and users alike, rather than a one-time checklist completed during the design phase.

This category includes the following principles:

- **Lifecycle management**: Security responsibilities extend beyond initial development. Components must be maintained, updated and eventually retired in a controlled manner, taking the expected duration of use into account. The practical application of lifecycle management, including how secure by design and secure by default principles are applied from design and development through to decommissioning, is explored in Section 2.
- **User centric design:** Security mechanisms must be usable and understandable even by everyday users. Poor usability often leads to insecure workarounds or misconfigurations. For example, if a system requires the user to perform complex manual configuration to enable encryption, they may skip the step or apply it incorrectly. On the other hand, providing a simple, guided setup that enables encryption automatically reduces the chance of misconfiguration and encourages the use of the security feature.

- **Secure coding practices:** Developers should follow established secure coding standards to prevent common vulnerabilities. Early identification and mitigation of insecure code ensure that weaknesses are not built into the system. For example, developers can use static application security testing (SAST) tools[21] while coding to identify vulnerabilities, and software composition analysis (SCA) to detect vulnerable third-party libraries. They can then apply dynamic testing tools[22] (DAST) before deployment to uncover runtime issues. These practices ensure code related vulnerabilities are identified early and are not after the product is released.
- **Logging, monitoring, and alerting:** Security depends on visibility. Systems should generate appropriate security-relevant logs, retain them for a defined period, and protect them from tampering so they can support investigation and compliance needs. Rather than simply collecting data, monitoring should be designed to detect suspicious behaviours such as repeated failed authentication attempts, privilege escalation, unexpected configuration changes, or unusual outbound connections.
- **Configuration and change management:** Secure operation requires configurations to be controlled, consistent, and auditable. Baseline hardening standards should be defined (for example, secure defaults, disabled unused services, and enforced encryption settings) and applied through repeatable mechanisms such as templates and infrastructure-as-code to reduce drift. Changes to systems should follow a governed process that includes review, testing, approval, and rollback plans, particularly for security-sensitive components.
- **Incident response and recovery:** Developers must be prepared to respond quickly and effectively to security incidents that affect their products in the field, including vulnerabilities, compromised code, malicious updates, and misuse of product functionality. This requires defined internal roles, escalation paths, and decision-making authority for security events, as well as documented playbooks for containment and customer communication.
- **Vulnerability and patch management:** Vulnerability and patch management should be practical, repeatable, and prioritised by risk. Manufacturers need a simple way for customers and researchers to report issues (for example, a dedicated security email address and a basic disclosure process), and an internal process to triage findings quickly and decide what needs urgent action.
- **Supply chain controls:** Developers and manufactures should protect product integrity without excessive process overhead, focusing on the points where a compromise would have the largest impact: code repositories, build systems, signing keys, and the channels used to distribute updates. At a minimum, source code and CI/CD access should be limited to named individuals, protected with multi-factor authentication, and reviewed through lightweight peer approval for changes to security-critical areas. SBOMs should be generated and maintained to support dependency transparency, vulnerability management, and product lifecycle security.

3.2 Secure by Default Principles

Security by default complements security by design by focusing on the product's security configuration presented to the user. Even well-designed systems can become vulnerable if released with insecure or overly permissive defaults settings.

The goal of security by default is to minimise the attack surface after deployment by ensuring that the most secure configuration is applied automatically[23]. This includes disabling unnecessary services,

[21] https://owasp.org/www-community/Source_Code_Analysis_Tools
[22] https://owasp.org/www-community/Vulnerability_Scanning_Tools
[23] Other useful references that cover the topic include ETSI EN 303 645 https://www.etsi.org/deliver/etsi_en/303600_303699/303645/03.01.03_60/en_303645v030103p.pdf, and NIST SP 800-213, https://csrc.nist.gov/pubs/sp/800/213/final

applying restrictive access controls, and providing clear information to users about the default security posture. Secure defaults reduce reliance on user expertise and limit the potential for misconfigurations, which are a common source of security incidents.

To summarise, Security by Design focuses on how the system is engineered, while Security by Default focuses on how the system arrives and behaves when the user first turns it on. We can organise Security by Default into two distinct categories: *Default Hardening* and *Guided Protection*.

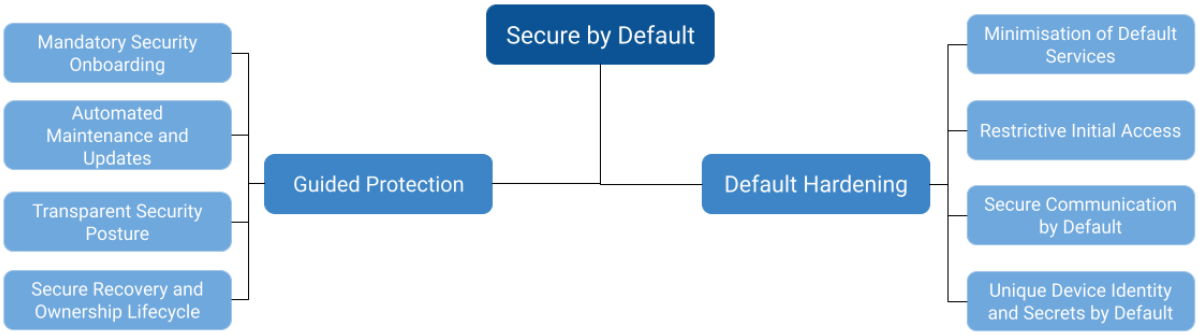


Figure 3: Secure by Default Principles

3.2.1 Default Hardening

The *default hardening* category focuses on the factory-shipped state of the software, ensuring that the initial configuration is as restrictive as possible. It aims to eliminate "low-hanging fruit" for attackers by removing unnecessary features and ensuring that all active components are operating at their highest security level without requiring any user input.

- **Minimisation of Default Services**: Any feature or service that is not essential for the core functionality of the product should be disabled by default. If a web server includes an optional file-sharing module that most users won't need, that module should be "off" until the user explicitly opts in, thereby reducing the immediate attack surface.
- **Restrictive Initial Access**: Systems should ship with the most restrictive permissions possible. This includes the elimination of universal "admin/admin" credentials and the enforcement of unique passwords and mandatory password changes upon first boot.
- **Secure Communication by Default**: All external communications should be encrypted and authenticated from the first connection. Rather than allowing an unencrypted HTTP or Telnet connection "for convenience," the system should strictly enforce protocols like TLS 1.3 or SSH, ensuring that data is protected the moment it leaves the device.
- **Unique device identity and secrets by default:** The product should ship with unique, per-device credentials and cryptographic identity (keys/certificates) rather than shared defaults. Any secrets used for authentication, update verification, or encrypted communications must be generated uniquely and protected against extraction. This reduces the risk that compromise of a single device or a leaked credential can be used to attack other customers or the wider installed base.

3.2.2 Guided Protection

Guided Protection addresses the interaction between the user and the system's security features. It acknowledges that human error is a primary cause of breaches and uses automated prompts,

mandatory setup steps, and clear feedback to ensure the user cannot easily or accidentally leave the system in an insecure state.

- **Mandatory Security Onboarding**: Critical security features should not be "hidden" in a settings menu; they should be part of the initial setup wizard. For instance, requiring a user to configure Multi-Factor Authentication (MFA) or an encryption key during the first-run experience ensures that the system enters a secure state before it is exposed to the internet.
- **Automated Maintenance and Updates**: A secure default posture must be sustainable. By enabling automatic security updates by default, the manufacturer ensures that the product remains protected against newly discovered vulnerabilities without requiring the SME to have a dedicated IT team to manually manage patching cycles.
- **Transparent Security Posture**: The system should clearly communicate its security status to the user. If a user chooses to disable a security feature or if a specific configuration increases risk, the system must provide a clear, understandable warning and offer a "one-click" path to return to the secure baseline.
- **Secure Recovery and Ownership Lifecycle:** The product should provide guided, low-friction recovery and transfer processes (credential reset, account recovery, secure factory reset, and ownership transfer) that are simple for users to follow but resistant to account takeover and social engineering.

SECTION 4

Playbooks

4. Playbooks

These playbooks provide a practical, lightweight way for small and medium-sized manufacturers and product teams to implement Secure by Design and Secure by Default without creating a heavy governance burden. Each playbook distils a single security principle into a one-page, execution-focused guide that teams can apply repeatedly across releases and product lines.

The intent is to translate security principles from abstract aspirations into concrete engineering and operational actions, with clear expectations, verifiable outcomes, and a consistent “definition of done” for security. Each playbook follows the same format to make adoption fast and repeatable:

- **Principle name**: the security concept being implemented.
- **Objective**: what the principle is trying to achieve and what failure modes it reduces.
- **Checklist**: the highest-impact actions to implement (designed to be achievable in lean teams).
- **Minimum evidence**: the smallest set of artefacts/logs/configurations that demonstrate the checklist was implemented.
- **Release gate**: a copy/paste set of pass/fail criteria that can be used in a release review (or CI/CD) to prevent regressions.

This structure is deliberately aligned to how SMEs operate: short cycles, shared responsibilities, limited specialist capacity, and a need for high signal-to-noise guidance.

Developers and manufactures should consider the following guidance when using the playbooks:

- Treat each playbook’s release gate as a standard agenda item in release readiness;
- Implement the minimum evidence as repository artefacts and CI outputs wherever possible;
- Allow exceptions only with documented rationale, owner, and review date;
- Refresh playbooks periodically based on incident learnings, vulnerability trends, and product changes.

The contents in this section should be treated as a baseline rather than a final state. As products evolve, risks change, and new features are introduced, the content should be reviewed and updated to ensure that Secure by Design and Secure by Default remain effective over time.

4.1 Trust boundaries and threat modelling

Principle	Secure architectures make trust explicit rather than assumed. Trust boundaries define where data, identities, and execution contexts cross from a more trusted domain to a less trusted one (or vice versa).
Objective	Ensure the system's architecture and data flows are understood, trust assumptions are explicit, and the highest-risk attack paths are identified early, so security controls and secure defaults are designed in, not retrofitted.
Checklist	
Draw the system (one diagram)	▪ Include: users/admins, device/app, backend services, APIs, data stores, third parties. ▪ Show key data flows and entry points (APIs, admin UI, OTA/update channel, local ports).
Mark trust boundaries and privileged paths	▪ Identify where trust changes (internet - backend, tenant - tenant, device - cloud, user - admin). ▪ Highlight high-privilege flows (auth, key management, updates, admin actions).
List critical assets	▪ E.g., Credentials/keys, customer/PII, payment tokens, device control functions, availability, audit logs, data, source code, and the IDE.
Identify and prioritise top threats (5–10 abuse cases)	▪ Map to boundaries/entry points (e.g., account takeover -admin actions; malicious update; API authz bypass; MITM during onboarding). ▪ Rate High/Med/Low (impact + likelihood). Pick top risks that drive design decisions
Define mitigations, secure defaults, and verification	▪ For each top threat: required controls and default configuration (deny-by-default, authn/authz, segmentation, signed updates, rate limits). ▪ Map each control to a verification method (tests, CI checks, config validation). ▪ Define refresh triggers (new interface, auth change, new sensitive data, new critical dependency, major architecture change).
Minimum evidence	
▪ 1 architecture/data-flow diagram with trust boundaries and entry points (stored in repo/wiki). ▪ Top threats list (5–10 abuse cases) with H/M/L priority and owners. ▪ Controls/defaults mapping for each top threat (what control, where enforced, how verified). ▪ Verification evidence: CI outputs or test results for key negative tests (unauthorised calls fail).	
Release gate	
✓ Diagram updated for this release (components, flows, dependencies reflect reality) ✓ Trust boundaries and privileged paths clearly marked ✓ Top 5–10 abuse cases reviewed; High risks have mitigations or documented exception ✓ Secure defaults confirmed for new/exposed interfaces (deny-by-default; least privilege) ✓ Verification in place: at least one "negative test" per critical boundary/privileged path (unauthorised access is denied) ✓ Threat model refresh triggered if any of: new API/interface, auth model change, new sensitive data, major dependency/supplier change, OTA/update changes, major architecture change	

4.2 Least Privilege

Principle	Every user, service, and process operate with the feasible minimum permissions needed to do its job, no more, no longer than necessary.
Objective	Reduces unauthorised access, limits blast radius, blocks lateral movement, and prevents permission creep.

Checklist	
Define "minimum viable permissions" per role/service	▪ List key roles (User, Support, Admin) and key services (API, Provisioning, Updates). ▪ For each, define allowed actions (read/write/admin) and resources (APIs, tables, buckets).
Use unique identities (no shared admin)	▪ One identity per service (service account/workload identity) , applicable to both web services and services running on a device. ▪ No shared admin users/keys; separate dev/test/prod identities.
Default-deny and explicit allow lists	▪ Deny by default; allow only required API methods, data access, and network paths. ▪ Restrict service-to-service access to specific interfaces.
Time-bound admin access (JIT) + attribution	▪ Named accounts + MFA for admins (all human users if possible). ▪ Temporary elevation for privileged actions; break-glass is separate and monitored.
Automate privilege hygiene	▪ Monthly (or each release): flag broad privileges and unused permissions/tokens; remove or time-limit exceptions.

Minimum evidence

- **Access model exists**: a short table "Role/Service, allowed actions, resources" stored in repo/wiki and kept current.
- **No shared admin**: IAM inventory shows unique service identities; no reused production admin keys.
- **Deny works**: automated "negative tests" prove unauthorised API/data access is blocked for each critical service.
- **Admins are accountable**: logs show who performed privileged actions; elevation expires automatically.
- **Creep is controlled**: recurring report flags unused/broad permissions; removals/exceptions are tracked with owner + expiry.

Release gate

- ✓ Unique service identities; no shared production admin keys
- ✓ Default-deny posture; only required paths enabled
- ✓ Admin access is MFA + time-bound + logged
- ✓ Automated authorisation tests verifying correct access restrictions for critical privileged functions.
- ✓ Monthly/release privilege review run; exceptions owned and time-limited

4.3 Strong identity and auth architecture

Principle	Design identity and authentication as a core part of the architecture. Use authoritative identity mechanisms for users, devices, services, and administrators across interfaces.
Objective	Prevent impersonation and unauthorised access.

Checklist

Define authoritative identity sources	▪ Identify and document the authoritative identity source (i.e., a system that acts as the single source of truth for creating, authentication/authorisation, and revoking identities) for users, devices, services, and administrators.
Use unique identities for all actors	▪ Assign a unique identity to each user, device, service, and administrator, prohibiting shared accounts/credentials, especially for administrative and service access.
Apply authentication across all interfaces	▪ Enforce authentication/authorisation on all access paths, including user interfaces, APIs, device-to-cloud communication, and management interfaces.
Strengthen authentication for high-risk access	▪ Require stronger authentication for privileged actions and sensitive operations (for example MFA, device binding, or cryptographic credentials).
Manage sessions and credentials securely	▪ Ensure sessions are time-limited, bound to identity, and invalidated on logout or credential change. ▪ Rotate, revoke, and expire credentials automatically when no longer needed or when compromise is suspected.

Minimum evidence

- **Authoritative identity sources defined:** a diagram or table showing the identity source for users, devices, services, and administrators exists.
- **No shared identities:** identity inventory shows unique identities, no shared production accounts or credentials.
- **Authentication enforced everywhere:** configuration or test evidence shows all interfaces enforce authentication, or are explicitly defined as public with appropriate protections applied.
- **Strong auth for privileged access:** policy or configuration shows MFA or equivalent enforced for administrative or sensitive actions.
- **Sessions and credentials controlled:** configuration or logs show session expiry, revocation on logout or credential change, and credential rotation or expiry in place.

Release gate

- ✓ Authoritative identity sources defined and documented
- ✓ Unique identities enforced; no shared production accounts or credentials
- ✓ Authentication required on all access paths (UI, APIs, device-to-cloud, management interfaces)
- ✓ Strong authentication enforced for privileged or sensitive actions
- ✓ Sessions and credentials are time-limited, revocable, and expire automatically

4.4 Attack surface minimisation

Principle	Expose only what is strictly necessary. Remove or disable all unnecessary code, services, interfaces, protocols, and dependencies by default.
Objective	Reduce the likelihood and impact of compromise by eliminating unnecessary entry points and limiting attacker options.

Checklist	
List exposed interfaces	▪ Identify all externally reachable APIs, ports, local interfaces (HMI), protocols, admin endpoints, and update channels. ▪ Remove or disable anything without a clear, current justification.
Enforce default-deny	▪ Close all ports, APIs, and interfaces by default. ▪ Explicitly allow only what is required for production.
Remove development and diagnostic functionality	▪ Remove debuggers, test endpoints, and debug modes from production builds. ▪ Remove development features.
Minimise dependencies	▪ Remove unused libraries, SDKs, and optional components from final builds. Remaining libraries should be verified for authenticity and integrity. ▪ Use the smallest possible operating system, runtime, or firmware configuration that supports the product's intended function.
Continuously monitor for legacy exposure	▪ Continuously review exposed interfaces and dependencies. ▪ Update/remove deprecated APIs, old protocols, and unused resources.

Minimum evidence

- **Exposure inventory exists:** a maintained list of externally reachable interfaces (APIs, ports, protocols, admin interfaces, update channels) stored in the repo or product documentation.
- **Default-deny enforced:** network rules, gateway configuration, or device settings show only explicitly required interfaces are enabled in production.
- **Minimal build verified:** build configuration, manifest, package list, or firmware contents show only required components are included.
- **No dev or diagnostic tooling shipped:** inspection of production artefacts confirms absence of shells, debuggers, compilers, test interfaces, or diagnostic utilities.
- **No legacy exposure:** release notes, ticket, or checklist entry confirms exposed interfaces and dependencies were reviewed and any unused or legacy elements removed.

Release gate

- ✓ Exposed interfaces reviewed; only required APIs, ports, protocols, and management interfaces enabled
- ✓ Default-deny posture confirmed; no unintended network paths or interfaces exposed
- ✓ Production build is minimal; unused libraries, services, and optional components removed
- ✓ No development or diagnostic tooling present in production artefacts
- ✓ Attack surface review completed for this release; deprecated or legacy interfaces updates/removed

4.5 Defence in depth

Principle	Apply multiple layers of security so that the failure or bypass of a single control does not result in full compromise.
Objective	Reduce the impact of security failures by layering independent controls that slow attackers, increase detection, and prevent single points of compromise.

Checklist	
Layer controls around critical assets	▪ Identify the most sensitive assets (e.g., credentials, control functions, customer data). ▪ Ensure more than one independent control protects each asset.
Assume control failure and design for containment	▪ Design systems so that bypass of one control does not grant unrestricted access.
Enable detection at multiple layers	▪ Generate security-relevant logs at key layers (identity, network, application, data). ▪ Ensure failed and suspicious actions raise alerts and are not silently blocked.
Use diverse and independent controls	▪ Avoid relying on multiple controls that share the same technology, identity source, or trust assumption.
Slow attackers and enable response	▪ Apply controls that increase attacker effort and time (rate limits, step-up authentication, staged access). ▪ Ensure response mechanisms exist to contain or isolate compromised components when alerts trigger

Minimum evidence

- **Layered protection mapped:** a simple table or diagram showing critical assets and the multiple security controls protecting each (stored in repo/wiki).
- **Independent controls confirmed:** configuration or design notes showing that layered controls do not rely on the same identity source or enforcement point.
- **Multi-layer logging enabled:** log configuration or sample logs showing security events recorded across controls.
- **Alerts and response defined:** alert rules or playbook reference showing how suspicious activity is detected and the associated containment or response action.

Release gate

- ✓ Critical assets have more than one security control applied, and the layering is documented
- ✓ Layered controls are independent; no single identity source or enforcement point is a single point of failure
- ✓ Security-relevant events are logged at multiple layers
- ✓ Alerts are in place for suspicious activity and a response action exists

4.6 Open Design

Principle	Systems should not depend on secrecy of design or hidden behaviour for protection. Security controls should remain effective even if an adversary understands how the system operates.
Objective	Ensure the product's security does not depend on secrecy of its design **or** implementation details ("security through obscurity"). Open Design makes security properties reviewable, testable, and maintainable.

Checklist

Document security-relevant design decisions	▪ Keep a short security design note covering: trust boundaries, auth model, cryptographic usage, update mechanism, logging/audit approach, and secure defaults. ▪ Record rationale for "why this control/protocol/library."
Prefer open standards and proven building blocks	▪ Use standard, widely reviewed protocols and libraries (e.g., TLS, OAuth/OIDC, JWT with clear constraints, signed updates). ▪ Avoid proprietary crypto, "home-grown" protocols, and undocumented encodings for security-critical flows.
Make interfaces and security expectations explicit	▪ Document APIs, authentication/authorization requirements, error handling, rate limits, and supported configurations. ▪ Provide a "secure configuration guide" (what must be enabled/disabled in production).
Enable review and disclosure	▪ Establish a vulnerability disclosure channel (security contact, intake process, basic triage/response targets). ▪ Encourage internal peer review for security-sensitive changes (auth, updates, crypto, boundary-crossing interfaces).
Maintain transparency of components and changes	▪ Maintain, publish, and sign an SBOM (at least for customer-facing builds) and track third-party components. ▪ Keep a security changelog/release notes for security-impacting fixes and default-setting changes.

Minimum evidence

- Security design note (1–3 pages) covering boundaries, auth, crypto choices, update path, logging, secure defaults.
- API + security requirements documentation (authn/authz, rate limits, data handling expectations).
- Secure configuration guide (production hardening defaults + required settings).
- Vulnerability disclosure process (public email/web page or customer-facing channel; triage ownership defined).
- SBOM (exported file or tool output) + dependency tracking approach.
- Security-sensitive PR review rule (CODEOWNERS, repo policy, or checklist evidence).

Release gate

- ✓ Security design note updated for any architectural/security-relevant change (auth, crypto, update path, trust boundary)
- ✓ No custom/proprietary crypto or undocumented security-critical protocols introduced
- ✓ API/security docs updated (auth requirements, scopes/roles, rate limits, secure defaults)
- ✓ Secure configuration guide updated; defaults validated for new/exposed interfaces
- ✓ SBOM generated for the release and stored/published per policy
- ✓ Vulnerability disclosure channel tested (contact works) and ownership/triage defined

4.7 Lifecycle management

Principle	Security responsibilities extend beyond initial development. Components must be maintained, updated and eventually retired in a controlled manner.
Objective	Ensure security is maintained throughout the product's full lifecycle, from requirements and design through deployment, maintenance, and end-of-life, so security does not degrade over time.

Checklist

Define support commitments and lifecycle states	▪ Publish (internally or externally) lifecycle states: Active, Maintenance, End-of-Sale, End-of-Support, End-of-Life. ▪ Define minimum support windows and internal guidelines for vulnerability triage and patch release timelines (e.g., critical vulnerabilities triaged in 48h; fixes released within X days where feasible).
Make the product updateable and recoverable	▪ Implement a secure and transactional update mechanism (authenticated updates; integrity protected; rollback/recovery plan). ▪ Ensure safe failure modes (don't brick devices; provide fallback/rollback where practical). ▪ During maintenance or recovery modes, restrict functionality and privileges to the minimum necessary to support update and recovery operations.
Operate vulnerability and patch management	▪ Track vulnerabilities from: third-party components, internal findings, customer reports. ▪ Maintain an SBOM (at least per release) and a dependency update cadence. ▪ Triage, prioritise, and document risk decisions (fix, mitigate, accept with expiry).
Monitor, log, and handle incidents	▪ Enable security-relevant logging (auth events, admin actions, update events, boundary-crossing requests). ▪ Define minimal incident response steps: detection, triage, containment, remediation, customer comms (as applicable)
Plan secure decommissioning and disposal	▪ Provide data erasure mechanisms and guidance (factory reset, key revocation, account de-provisioning). ▪ Revoke credentials/keys when devices/users/services are deactivated. ▪ Ensure end-of-life guidance includes what security updates cease and what customers must do.

Minimum evidence

- **Lifecycle policy**: lifecycle states + support window + patch SLAs (1 page).
- **Update/release process**: secure update approach + rollback/recovery notes.
- **Vulnerability management log**: intake, triage, prioritisation, disposition, owner, dates (ticket board or spreadsheet).
- **SBOM:** SBOM per release (or dependency inventory) and a dependency update cadence.
- **Monitoring and Logging:** Security logging baseline + sample logs showing admin/auth/update events.
- **Decommissioning checklist**: data wipe/reset, credential/key revocation, and customer guidance.

Release gate

- ✓ Support status and lifecycle state for this release are clear (Active/Maintenance/etc.)
- ✓ Secure update path validated (integrity/authentication checks; rollback/recovery documented)
- ✓ Vulnerability triage completed for known issues (including third-party deps); decisions recorded (fix/mitigate/accept with expiry)
- ✓ SBOM generated (or dependency inventory updated) and stored for the release
- ✓ Security logging verified for critical events (auth, admin actions, updates)
- ✓ Decommissioning actions defined for components introduced/changed (reset, wipe, key revocation)

✓ Any accepted residual security risk has an owner and review/expiry date

4.8 User centric design

Principle	Security mechanisms must be usable and understandable even by everyday users.
Objective	Ensure security controls and secure defaults are usable, understandable, and aligned to real user workflows, so security outcomes do not depend on expert behaviour.

Checklist	
Design secure defaults that require minimal user action	▪ Ship with the safest practical settings enabled (e.g., least exposure, secure comms, logging on). ▪ Avoid “optional security” for critical protections; make insecure modes explicit and gated.
Make setup and onboarding safe and simple	▪ Provide a guided first-run experience with clear security steps (e.g., change initial credentials, pair securely, enable updates). ▪ Minimise manual configuration for sensitive settings; prefer automated secure provisioning.
Use clear, actionable security messaging	▪ Write user-facing warnings and errors that explain *what happened*, *impact*, and *what to do next*. ▪ Avoid vague messages (“failed”) or blame-the-user language; include remediation steps.
Provide least-friction access management	▪ Prefer role-based access and simple permission models that match user roles (Admin, Operator, Viewer). ▪ Support safer auth options where feasible (MFA for admins, device identity, short-lived tokens). ▪ Remove/limit insecure recovery paths; make account recovery robust.
Validate usability to prevent insecure workarounds	▪ Run quick usability checks on key security flows (onboarding, password reset, admin tasks, updates). ▪ Use support tickets and pseudonymised telemetry (where appropriate) to detect confusion and misconfiguration trends.

Minimum evidence

- Secure defaults list (what ships enabled/disabled, with rationale).
- Onboarding/security setup guide (short, step-by-step; includes credential and update steps).
- Role/permission model (roles defined + what each can do).
- User-facing security message catalogue (examples of warnings/errors and required remediation text).
- Usability validation notes (at least 3–5 users or internal proxies) covering the top security flows, plus resulting fixes.

Release gate

- ✓ Secure defaults reviewed and validated (no critical protections disabled by default)
- ✓ No default admin credentials active; onboarding enforces safe initial setup
- ✓ Admin/security-critical actions have clear UX (confirmations, guidance, and safe recovery paths)
- ✓ Roles and permissions match real user workflows; privilege escalation is explicit and auditable
- ✓ Security warnings/errors are actionable (what/why/how to fix) and documented
- ✓ Basic usability check completed for key security flows (onboarding, admin task, recovery, updates); issues tracked/fixed or accepted with owner + date

4.9 Secure coding practices

Principle	Developers should follow established secure coding standards to prevent common vulnerabilities.
Objective	Reduce the introduction of common, high-impact vulnerabilities by standardising how code is written, reviewed, and tested.

Checklist

Adopt a secure coding baseline (language/framework specific)	▪ Define “must-follow” rules for your stack (input handling, auth checks, error handling, crypto usage, logging). ▪ Ban unsafe patterns (e.g., string-built SQL, eval-like functions, disabling TLS verification).
Validate inputs and encode outputs	▪ Centralise validation (schemas, allowlists, length/range checks). ▪ Encode output by context (HTML/JSON/SQL); avoid mixing data and commands.
Use safe dependency and secrets practices	▪ Pin and regularly update dependencies; remove unused packages. ▪ Secrets (e.g., API keys, tokens, credentials) must never be stored in source code or committed to source repositories. Secure secret management solutions should be used instead. ▪ Generate and store an SBOM per release (or a dependency inventory at minimum).
Make security-sensitive changes reviewable	▪ Clearly define and document roles for each stakeholder involved in the CI/CD pipeline (e.g., Developer, Support, Administrator, Integrator, CI Runner) ▪ Protect critical branches to prevent direct commits and history rewriting. ▪ Prohibit force push on protected branches to prevent history rewriting and preserve auditability. ▪ Require peer review for changes touching authn/authz, boundary-crossing APIs, crypto, update mechanisms, deserialization, and any new external exposure. ▪ Use PR checklists and code owners for sensitive areas.
Automate detection in CI/CD	▪ Run SAST + dependency scanning on each PR; fail builds on critical findings. ▪ Add targeted unit/integration “negative tests” (unauthorised access denied; injection attempts fail). ▪ Add basic fuzzing or property-based testing for high-risk parsers/inputs if feasible.

Minimum evidence

- Secure coding standard (1–2 pages or repo file) for your stack + banned patterns list.
- PR checklist / CODEOWNERS for security-sensitive modules.
- CI pipeline evidence: SAST and dependency scan results for the release.
- Secrets controls: proof secrets are not in repo (pre-commit/secret scanning results) + secrets manager usage.
- Test evidence: at least a small suite of negative tests for critical endpoints/inputs.

Release gate

- ✓ Secure coding baseline exists for the stack and is referenced in the repo
- ✓ SAST + dependency scanning run in CI; critical issues fixed or documented exception (owner + expiry)
- ✓ Secret scanning enabled; no secrets committed; rotation performed if exposure occurred
- ✓ Security-sensitive changes were peer reviewed (auth/authz, crypto, parsers, external interfaces)
- ✓ Critical endpoints have negative tests (unauthorised access denied; injection attempts fail)
- ✓ Dependencies reviewed for high/critical vulns; patch/mitigation plan recorded for any accepted risk

4.10 Logging, monitoring, and alerting

Principle	Systems should generate appropriate security-relevant logs, retain them for a defined period, and protect them from tampering so they can support investigation and compliance needs.
Objective	Provide sufficient visibility to detect misuse, investigate incidents, and maintain system integrity over time. The focus is on a small set of high-signal security events, reliable log retention, and actionable alerts that do not overwhelm teams.

Checklist

Define the "must-log" security events	▪ Authentication: login success/failure, MFA events, token issuance/refresh/revocation. ▪ Authorization: access denied, privilege/role changes, admin actions. ▪ Boundary events: external API calls, device onboarding/pairing, OTA/update events. ▪ Data/security changes: configuration changes, key/secret changes, user/service enable/disable. ▪ Do not log personal or secret data by default. In general, log only the bare minimum required to understand events in default log level
Standardised access and audit log structure and attribution	▪ For access and authorisation logging, use consistent fields capturing who (actor), what (action/event type), and when (timestamp), with additional contextual fields as appropriate. ▪ Ensure admin actions are attributable to a named identity (no shared accounts).
Centralised collection and protect log integrity	▪ Central log store (SIEM/log platform or managed logging). ▪ Separate logs based on type and or service ▪ Restrict access to logs; separate duties and write-only where feasible. ▪ Set retention targets (e.g., 30–90 days searchable, longer archival if required). Plan storage accordingly. ▪ Ensure time sync (NTP) across systems.
Implement actionable monitoring and alerts	▪ Start with a small alert set (high confidence, high impact): repeated auth failures, new admin creation, privilege escalation, unusual API error spikes, updates failing, disabled logging, access to sensitive endpoints from new locations. ▪ Tune alerts to reduce noise (rate thresholds, suppression windows, environment filters).
Practice incident triage and response	▪ Define on-call/ownership for security alerts. ▪ Document a minimal triage playbook covering: validate, scope, contain, remediate and communicate. ▪ Run a quarterly "tabletop" or dry run for one common scenario (account takeover or API key leak).

Minimum evidence

- Logging baseline**:** list of must-log events + required fields (1 page).
- Central logging proof**:** screenshot/export/config showing logs from key components arriving centrally.
- Retention setting: configured retention + access control policy for logs.
- Alert catalogue**:** list of enabled alerts with thresholds and owners (initial set can be ~5–10).
- Triage runbook**:** one-page steps and escalation contacts + one recent test (tabletop note).

Release gate

- ✓ Must-log security events implemented for new/changed components (auth, admin, updates, boundary events)
- ✓ Logs include attribution fields (actor, request/session ID, source) and are centralised
- ✓ Log access restricted; retention configured; time sync verified
- ✓ High-signal alerts enabled and assigned an owner (at least: brute force, new admin/role change, key/secret change, update failures)

- ✓ Triage runbook exists and was exercised recently (tabletop or test alert)
- ✓ Logging/alerting exceptions are documented with owner + review/expiry date

4.11 Configuration and change management

Principle	Secure operation requires configurations to be controlled, consistent, and auditable. Baseline hardening standards should be defined and applied through repeatable mechanisms such as templates and infrastructure-as-code to reduce drift.
Objective	Prevent security regressions and outages by ensuring system configuration is controlled, reviewable, and reproducible, and that changes are assessed for risk before reaching production. The priority is to eliminate "silent drift," tighten defaults, and make rollbacks reliable.

Checklist

Version and review configuration (treat it as code)	▪ Store environment config, infrastructure templates, and policies in version control. ▪ Require peer review for changes affecting exposure, identity/access, secrets, networking, logging, and updates.
Harden defaults and prevent drift	▪ Establish secure baseline configurations (deny-by-default where possible). ▪ Use automated checks to detect drift between desired vs actual config (at least for critical settings).
Separate environments and limit privilege	▪ Strict separation for dev/test/prod (accounts/projects, networks, credentials). ▪ Restrict who can deploy to prod; remove direct "manual" changes where feasible.
Implement change gating and rollback	▪ Use CI/CD gates for config and infra changes (linting, policy checks, approval). ▪ Require a rollback plan for each production change (including config-only changes). ▪ Keep "break-glass" changes rare, logged, and post-reviewed.
Track and assess security-impacting changes	▪ Tag changes that affect: external exposure, authentication/authorization, crypto, update mechanisms, data classification, third-party integrations. ▪ Trigger a lightweight threat/risk review when these tags apply.

Minimum evidence

- ▪ **Config is versioned**: repo location(s) for IaC/config/policies; PR review history.
- ▪ **Baseline config**: documented secure defaults and key settings (1 page).
- ▪ **Drift detection**: a scheduled job/report or tool output for critical config items.
- ▪ **Deployment controls**: proof prod changes go through CI/CD (or change tickets) and are attributable.
- ▪ **Rollback proof**: last rollback test result or documented rollback procedure.
- ▪ **Change log**: record of security-impacting changes and associated approvals.

Release gate

- ✓ Production config/IaC changes are versioned and peer-reviewed (no untracked manual edits)
- ✓ Secure baseline defaults applied; new components inherit baseline (network, IAM, logging)
- ✓ Dev/test/prod separated (accounts/projects/credentials); least privilege enforced for deployers
- ✓ CI/CD gates applied to config changes (policy checks/linting); approvals recorded
- ✓ Rollback plan exists for this release; rollback procedure tested recently or validated
- ✓ Security-impacting changes tagged and reviewed (exposure, IAM, secrets, crypto, updates, integrations)
- ✓ Exceptions (emergency changes) are logged and post-reviewed with owner + expiry date

4.12 Incident response and recovery

Principle	Developers must be prepared to respond quickly and effectively to security incidents that affect their products in the field, including vulnerabilities, compromised code, malicious updates, and misuse of product functionality.
Objective	Detect, contain, and recover from security incidents quickly while limiting customer impact and preventing recurrence. The priority is a clear playbook, defined ownership, fast triage, and reliable restore/rollback, supported by the minimum logging and backup evidence needed to act decisively.

Checklist

Define roles, escalation, and decision authority	▪ Name an incident lead and backups; define who can approve customer comms, access elevation, and emergency changes. ▪ Maintain an on-call/escalation contact list (internal + key suppliers).
Create a minimal incident runbook	▪ Steps: detect, triage, contain, eradicate, recover, lessons learned. ▪ Include checklists for common scenarios (account takeover, leaked API key, ransomware/supply-chain alert, compromised device fleet).
Prepare containment controls	▪ Ability to revoke/rotate credentials and keys quickly. ▪ Ability to disable accounts, block IPs, quarantine services/devices, and roll back releases/config changes. ▪ "Break-glass" access exists, is logged, and is time-bound.
Ensure recovery capability	▪ Backups are automated, tested, and protected (immutability where feasible). ▪ Restore procedures are documented for critical systems (data, configs, secrets, device firmware if applicable). ▪ Define RTO/RPO targets appropriate to the business.
Exercise and improve	▪ Run a lightweight tabletop quarterly (30–60 minutes) on one realistic scenario. ▪ Track actions to closure; update runbooks, detections, and controls based on findings.

Minimum evidence

- ✓ **Defined roles:** Incident response (IR) contact list + roles (owner, backups, escalation paths).
- ✓ **Runbooks:** 1-page incident runbook + at least one scenario checklist.
- ✓ **Containment proof:** documented steps and permissions to revoke keys, disable accounts, and roll back deployments.
- ✓ **Backup/restore proof:** backup configuration + one recent restore test result (or evidence of a successful restore).
- ✓ Post-incident / tabletop notes showing actions and improvements (even if simulated).

Release gate

- ▪ **Defined roles:** Incident response (IR) contact list + roles (owner, backups, escalation paths).
- ▪ **Runbooks:** 1-page incident runbook + at least one scenario checklist.
- ▪ **Containment proof:** documented steps and permissions to revoke keys, disable accounts, and roll back deployments.
- ▪ **Backup/restore proof:** backup configuration + one recent restore test result (or evidence of a successful restore).
- ▪ Post-incident / tabletop notes showing actions and improvements (even if simulated).

4.13 Vulnerability and patch management

Principle	Vulnerability and patch management should be practical, repeatable, and prioritise by risk. Manufacturers need a simple way for customers and researchers to report issues, and an internal process to triage findings quickly and decide what needs urgent action.
Objective	Identify, prioritise, and remediate vulnerabilities fast enough to reduce real-world exposure, across your code, dependencies, infrastructure, and (if applicable) devices/firmware. The focus is a simple intake-to-fix workflow, clear SLAs, and an update mechanism that makes patching reliable.

Checklist

Establish intake channels (don't miss issues)	▪ Sources: dependency scanning, SAST/DAST findings, supplier advisories, customer reports, security email, etc. ▪ Assign a single owner for triage and tracking.
Triage and prioritise consistently	▪ Use a lightweight severity approach (e.g., Critical/High/Med/Low) plus "internet-exposed?" and "known exploited?" flags. ▪ Decide quickly: fix now, mitigate, accept (time-bound), or defer (with rationale).
Patch dependencies and third parties proactively	▪ Maintain a regular cadence (e.g., weekly/monthly) for dependency updates. ▪ Pin versions; remove unused dependencies; track transitive dependencies.
Fix, test, and release with a secure process	▪ Ensure fixes are reviewed and tested; verify no regressions in auth/authz, input validation, and critical workflows. ▪ For devices/IoT: ensure secure OTA/update path and safe rollback where feasible.
Communicate and close the loop	▪ Track affected versions, customers/environments, and mitigation guidance. ▪ Publish security release notes or advisories as appropriate. ▪ Verify rollout completion and update the risk register.

Minimum evidence

- **Vuln tracking board/register**: issue, severity, affected components/versions, owner, status, target date.
- **Defined SLAs** (example): Critical triage ≤ 48h; remediation/release target ≤ X days (set to your reality).
- **Scanning evidence**: CI outputs for dependency scanning + SAST (and DAST if applicable).
- **Proactive dependency patches:** SBOM or dependency inventory per release (at minimum for shipped artifacts).
- **Patch release record**: link from vuln ticket → PR(s) → tests → release version → rollout confirmation.
- **Exception log**: accepted risks have owner + expiry/review date and compensating controls (if any).

Release gate

- ✓ Dependency and SAST scans executed for the release; Critical/High findings addressed or documented exception (owner + expiry)
- ✓ SBOM (or dependency inventory) generated/updated and stored for the release
- ✓ Known vulnerabilities affecting shipped components are triaged with severity, owner, and target date
- ✓ Patch process validated: fix reviewed, tests passed, and release notes updated as needed
- ✓ For internet-exposed components: mitigations or patches for Critical/High are in place before release
- ✓ OTA/update (if applicable) validated for secure delivery; rollback/recovery documented
- ✓ Accepted residual risk is time-bound and tracked to closure or review date

4.14 Supply chain controls

Principle	Developers should protect product integrity without excessive process overhead, focusing on the points where a compromise would have the largest impact: code repositories, build systems, signing keys, and the channels used to distribute updates.
Objective	Reduce the risk of compromise through third parties (software components, libraries, build tools, CI/CD, contractors, hosting providers, and hardware/firmware suppliers) by establishing minimum, repeatable controls that improve visibility, integrity, and accountability across what you buy, build, and ship.

Checklist

Know what you depend on (inventory + SBOM)	▪ Maintain an inventory of critical suppliers and third-party components. ▪ Generate an SBOM per release (or minimum: dependency manifest snapshot) to improve visibility and response speed.
Control what enters your codebase	▪ Pin dependency versions; require review for new dependencies and major upgrades. ▪ Run dependency scanning in CI; block builds on Critical/High where feasible (or require explicit exception).
Harden the build and release pipeline (integrity of artifacts)	▪ Restrict CI/CD permissions (least privilege), protect secrets, and separate build and deploy roles. ▪ Sign release artifacts (and provenance if feasible). Use a staged target such as SLSA levels to guide improvements.
Set minimum supplier expectations (lightweight due diligence)	▪ For critical suppliers: require security contact/VDP, patch timelines, and confirmation of secure development practices. ▪ Use a short questionnaire aligned to a recognised baseline (e.g., OWASP SCVS) rather than bespoke questions.
Plan for supplier failure	▪ Identify single points of failure (key providers/components) and define fallbacks (alternate package source, vendor substitution plan, rapid disable/mitigation). ▪ Ensure contracts/SLAs cover vulnerability notification and support windows (where possible).

Minimum evidence

- Supplier + component inventory (top vendors, critical libraries/tools, hosting/build services).
- SBOM (or dependency inventory) per release stored with the release artifacts.
- CI results showing dependency scanning (and ideally secret scanning) executed on PRs/releases.
- Release integrity evidence: artifact signing and restricted CI/CD access (configs/screenshots/logs).
- Supplier baseline checks for critical suppliers (completed questionnaire or SCVS-aligned checklist).
- Exception log for accepted supply-chain risks (owner + expiry date).

Release gate

- ✓ SBOM (or dependency inventory) generated and stored for this release
- ✓ Dependency scanning executed; Critical/High issues fixed or have documented exception (owner + expiry)
- ✓ New dependencies/suppliers reviewed and approved (recorded in PR/ticket)
- ✓ CI/CD and build secrets protected; build/deploy permissions least-privilege and attributable
- ✓ Release artifacts signed (and provenance captured where feasible / per chosen SLSA target)
- ✓ Critical suppliers meet baseline expectations (security contact, vulnerability notification, support/patch commitments)
- ✓ Supply-chain risk exceptions recorded with owner + review/expiry date

4.15 Minimisation of Default Services

Principle	Disable the non-essential features and services by default. Only the functionality required for the core operation of the product should be enabled by default.
Objective	To reduce the attack surface and ensure that products start in a hardened state without relying on users to disable specific functionality.

Checklist

Define core functionality	▪ Identify the minimum set of features and services required for the product to operate.
Disable non-essential features and services by default	▪ Ensure optional features and services are disabled by default.
Require explicit opt-in for additional functionality	▪ Ensure non-essential features and services can only be enabled through deliberate user action.
Explain security implications on enablement	▪ Inform users of security risks when enabling optional features and services. ▪ Avoid enabling additional features and services without clear acknowledgement.
Review defaults on change	▪ Reassess default-enabled features and services when new features are introduced.

Minimum evidence

- **Core features and services defined:** documentation or configuration identifying which features and services are required by default.
- **Non-essential features and services disabled:** default configuration or build artefacts show optional features and services and modules are off.
- **Explicit opt-in required:** configuration or documentation shows how additional features and services must be manually enabled.
- **Security implications disclosed:** documentation, UI text, or configuration prompts showing users are informed of security implications when enabling non-essential features and services.
- **Defaults reviewed for this release:** release notes, checklist, or ticket confirming default settings were reviewed after changes.

Release gate

- ✓ Only core functionality is enabled by default
- ✓ Optional features and services are disabled unless explicitly enabled by the user
- ✓ Users are informed of security implications when enabling non-essential features and services
- ✓ No new features or service is enabled by default without review
- ✓ Default configuration is reviewed and confirmed for this release

4.16 Restrictive Initial Access

Principle	Ship systems in the most restrictive access state possible. Eliminate shared or default credentials and require a secure access setup before any privileged or sensitive operations are allowed.
Objective	To prevent immediate compromise after deployment by ensuring attackers cannot gain access through default or weak initial access.

Checklist

Eliminate shared default credentials	▪ Do not ship products with universal usernames or passwords (e.g., admin/admin or root/root).
Require unique credentials per device or instance	▪ Ensure that each device or instance has unique credentials generated at manufacture, provisioning, or first boot.
Enforce a secure initial setup	▪ Require a credential change, key provisioning, or account creation before allowing privileged or sensitive access.
Restrict initial permissions	▪ Grant the minimum permissions required for initial operation ▪ Avoid granting full administrative access by default.
Protect initial access paths	▪ Ensure that first-boot, onboarding, and recovery interfaces are authenticated and not exposed unnecessarily.

Minimum evidence

- **No default credentials shipped:** build configuration or documentation confirms no shared or hard-coded credentials exist.
- **Unique credentials enforced:** provisioning records, configuration, or device inventory show per-device or per-instance credentials.
- **Secure setup required:** configuration or test evidence shows credential change or secure setup is mandatory before privileged or sensitive access.
- **Initial permissions restricted:** access model or configuration shows limited permissions at first use.
- **Initial access paths controlled:** configuration or test evidence shows onboarding and recovery interfaces are protected.

Release gate

- ✓ No shared or default credentials present in production builds
- ✓ Unique credentials generated per device or instance
- ✓ Secure setup enforced before privileged or sensitive access
- ✓ Initial permissions are restrictive by default
- ✓ Initial access and onboarding paths are protected and reviewed

4.17 Secure Communication by Default

Principle	Enforce secure communication from the start. All external communications must be encrypted and authenticated by default, with no support for insecure or plaintext protocols for convenience.
Objective	Protect data in transit and prevent interception, tampering, or impersonation by ensuring communications are always secure, without relying on user configuration or later hardening.

Checklist	
Encrypt external communications by default	▪ Require encrypted communication for all external interfaces from first use.
Disable insecure and plaintext protocols	▪ Do not allow unencrypted or weak protocols (e.g., HTTP or Telnet) to be used by default or even as fallbacks.
Authenticate communicating endpoints	▪ Ensure external endpoints authenticate each other where appropriate, for example for device-to-cloud, service-to-service, and administrative access.
Enforce secure protocol versions and configurations	▪ Use modern, secure protocol versions and configurations by default. ▪ Explicitly disable weak ciphers, legacy versions, and insecure negotiation modes.
Fail securely on connection errors	▪ Ensure communication failures result in denied connections rather than downgrade to insecure protocols or configurations.

Minimum evidence

- **Secure protocols enforced by default:** configuration or test evidence shows encrypted and authenticated communication is required from first connection.
- **Insecure protocols disabled:** configuration confirms plaintext or legacy protocols are not enabled or available.
- **Endpoint authentication in place:** configuration or test evidence shows mutual or appropriate endpoint authentication for external communications.
- **Strong protocol configuration applied:** configuration shows only approved protocol versions and cryptographic settings are enabled.
- **No insecure fallback:** test evidence shows connection attempts fail securely rather than downgrading security.

Release gate

- ✓ External communications are encrypted and authenticated from first connection
- ✓ Insecure or plaintext protocols are disabled and unavailable
- ✓ Only approved secure protocol versions and configurations are enabled
- ✓ Endpoint authentication enforced where required
- ✓ Connection failures do not result in insecure fallback

4.18 Unique Device Identity and Secrets by Default

Principle	Ship each product instance with a unique cryptographic identity and secrets by default. Do not use shared credentials, keys, or certificates across devices or installations.
Objective	Prevent large-scale compromise by ensuring that the breach of one device or leaked secret cannot be reused to attack other devices, customers, or the wider installed base.

Checklist

Generate unique identities per device or instance	▪ Assign a unique device identity (for example key pair or certificate) to every device or installation.
Eliminate shared or hard-coded secrets	▪ Do not use shared credentials, default keys, or embedded secrets across products.
Protect secrets against extraction	▪ Store device identities and per-device secrets using platform-appropriate controls that protect against extraction, tampering, or unauthorised replacement (for example secure storage, hardware-backed protection, or restricted access).
Use unique secrets for security-critical functions	▪ Ensure security-critical functions like authentication, secure communications, and update verification rely on per-device secrets, not shared material.
Support revocation and replacement	▪ Ensure device identities and secrets can be revoked and rotated if compromise is suspected.

Minimum evidence

- **Unique device identities issued:** device inventory or provisioning records show a unique identity (key or certificate) per device or instance.
- **No shared secrets present:** build artefacts or configuration confirm no shared credentials or hard-coded secrets exist.
- **Secrets protected at rest:** documentation or configuration shows secrets are stored using appropriate protection mechanisms.
- **Security functions use unique secrets:** configuration or design notes show authentication, communication, and update processes rely on per-device secrets.
- **Revocation supported:** documentation or configuration shows device identities or secrets can be revoked or replaced.

Release gate

- ✓ Unique cryptographic identity generated per device or instance
- ✓ No shared or default secrets present in production builds
- ✓ Secrets are protected against extraction
- ✓ Security-critical functions use per-device secrets
- ✓ Identity and secret revocation or replacement is supported

4.19 Mandatory Security Onboarding

Principle	Require critical security controls to be configured during initial setup. Do not allow products to enter an operational state until essential security steps are completed.
Objective	Ensure systems start in a secure state by preventing or limiting use before baseline security controls, such as strong authentication or encryption, are configured.

Checklist	
Identify mandatory security steps	▪ Define the minimum-security controls required before the product can be used (for example MFA, encryption keys, admin account setup).
Enforce security setup during first use	▪ Integrate mandatory security steps into the initial setup or onboarding flow. ▪ Do not allow users to skip or defer required security configuration.
Block operation until onboarding is complete	▪ Prevent normal operation, privileged actions, or external connectivity until mandatory security steps are completed.
Provide clear, guided configuration	▪ Guide users through security setup with clear instructions and sensible defaults. ▪ Avoid requiring expert knowledge to complete onboarding securely.
Re-trigger onboarding when security is incomplete	▪ Re-enforce onboarding if critical security settings are removed, reset, or left unconfigured.

Minimum evidence

- **Mandatory security steps defined:** documentation identifies which security controls must be completed during onboarding.
- **Onboarding enforces a security setup:** configuration, screenshots, or test evidence shows critical security steps cannot be skipped.
- **Operation blocked pre-onboarding:** configuration, screenshots, or test evidence shows the product cannot enter normal operation before onboarding is complete.
- **Guided setup is provided:** screenshots, configuration flows, or documentation show clear user guidance for security setup.
- **Onboarding is re-enforced on incomplete security configuration:** configuration, screenshots, or test evidence shows onboarding is re-triggered if required security settings are missing.

Release gate

- ✓ Mandatory security steps are defined and enforced during initial setup
- ✓ Users cannot bypass or skip critical security configuration
- ✓ Product cannot enter normal operation before onboarding completion
- ✓ Security onboarding provides clear guidance and secure defaults
- ✓ Onboarding is re-enforced if critical security settings are removed

4.20 Automated Maintenance and Updates

Principle	Enable automated security maintenance and updates by default. Products should remain secure over time without requiring manual intervention or operational effort.
Objective	To ensure that vulnerabilities are addressed quickly and the device is left vulnerable for as little time as possible.

Checklist	
Managed Security Updates by Default	▪ Enable security updates by default, ensuring the product automatically checks for updates, informs the user, and supports timely installation, with escalation where updates are repeatedly deferred.
Separate security updates from feature changes	▪ Ensure that critical security updates can be delivered independently of optional feature/functional updates.
Verify the authenticity and integrity of updates	▪ Ensure that updates are cryptographically verified before installation.
Apply updates safely	▪ Design updates to minimise disruption and avoid leaving the system in an insecure or unusable state if the update fails.
Inform users of update activity	▪ Notify users when security updates are applied or when action is required, without requiring them to manage the update process manually.

Minimum evidence

- **Automatic updates enabled:** default configuration shows security updates are enabled out of the box.
- **Security updates decoupled:** configuration or release notes show security fixes can be delivered independently of optional feature/functional updates.
- **Update verification enforced:** configuration or design notes show updates are authenticated and integrity-checked.
- **Safe update mechanism in place:** documentation or test evidence shows updates can be applied without significantly disrupting basic operation.
- **User notification supported:** logs, UI messages, or documentation show users are informed of update activity or status.

Release gate

- ✓ Automatic security updates are enabled by default
- ✓ Security updates can be delivered independently of features
- ✓ Updates are cryptographically verified before installation
- ✓ Failed updates do not leave the system insecure or unusable
- ✓ Users are informed of update status without manual intervention

4.21 Transparent Security Posture

Principle	Make the system's security posture visible and understandable. Clearly inform users when security protections are weakened and provide a simple path to return to a secure baseline.
Objective	Reduce risk from accidental or uninformed misconfiguration by ensuring users understand the security state of the system and can easily correct insecure settings.

Checklist

Define security states and baseline	▪ Define the secure baseline and any degraded or unsupported security states. ▪ Document which configurations cause a transition between states.
Expose current security status	▪ Indicate clearly whether the system is operating in a secure or degraded security state.
Warn on changes that reduce security	▪ Show clear warnings when users disable security features or apply configurations that increase the risks.
Explain impact in plain language	▪ Explain to the user the security implications of a change in non-technical terms.
Provide a simple path to recovery	▪ Offer the user a simple interaction to return to the secure baseline.

Minimum evidence

- **Security states defined:** documentation identifies the secure baseline and degraded or insecure states, including the configuration changes that trigger this state.
- **Security status visible:** screenshot, CLI output, or documentation clearly shows the current security state.
- **Warnings triggered on risk increase:** screenshots, prompts, or test evidence shows warnings when users apply security-reducing changes.
- **Impact explained clearly:** warning messages screenshot or documentation demonstrate that security implications are described in plain, non-technical language.
- **Baseline restore available:** screenshot, command, or configuration shows a simple action exists to return to the secure baseline.

Release gate

- ✓ Secure baseline and degraded security states are defined and documented
- ✓ Current security state is clearly visible to the user
- ✓ Users are warned when security protections are reduced
- ✓ Security impact is explained in clear, non-technical language
- ✓ A simple action exists to restore the secure baseline

4.22 Secure Recovery and Ownership Lifecycle

Principle	Provide secure, guided recovery and ownership transfer mechanisms by default. Recovery, reset, and transfer processes must be easy for users, operators, and administrators to follow, while remaining resistant to abuse, account takeover, and social engineering.
Objective	To enable users and administrators to recover access, reset systems, or transfer ownership safely.

Checklist	
Define supported recovery and transfer actions	▪ Identify supported recovery actions such as credential reset, account recovery, secure factory reset, and ownership transfer, clearly defining who is allowed to initiate each action and under what conditions
Verify identity before recovery or transfer	▪ Require strong identity verification before allowing recovery, reset, or ownership transfer actions.
Protect recovery paths from abuse	▪ Ensure recovery mechanisms cannot be used to bypass normal authentication or authorisation controls. ▪ Apply monitoring and rate limiting to recovery and reset workflows.
Ensure secure reset and ownership transfer	▪ Ensure factory reset and ownership transfer fully remove previous user access, credentials, and secrets.
Provide guided, secure workflows	▪ Offer clear recovery and transfer processes that reduce user error

Minimum evidence

- **Recovery and transfer actions are defined:** documentation identifies supported recovery, reset, and ownership transfer processes.
- **Strong verification is enforced:** configuration or test evidence shows recovery and transfer actions require a verified identity.
- **Abuse protections are applied:** configuration, logs, or test evidence show rate limiting and monitoring of recovery workflows.
- **Secure reset and transfer is verified:** test evidence confirms factory reset and ownership transfer remove prior credentials and access.
- **Guided workflows are available:** screenshots, CLI commands, or documentation demonstrate guided recovery and transfer processes.

Release gate

- ✓ Supported recovery and ownership transfer actions are defined
- ✓ Recovery and transfer actions require strong identity verification
- ✓ Recovery mechanisms cannot bypass normal access controls
- ✓ Factory reset and ownership transfer fully removes prior access
- ✓ Recovery and transfer workflows have clear guidance

SECTION 5

Machine Readable Security Attestations

5. Machine Readable Security Attestation

5.1 Demonstrability and verifiability

In modern software engineering, the transition from manual, document-heavy compliance to machine-readable security attestations marks a critical evolution in how trust can be verified. At its core, a machine-readable attestation is a digital claim, encoded in formats like JSON or YAML, asserting that a specific security control, process, or property has been met. Unlike static PDF reports that sit in a folder, these digital artefacts are “generatable” and “consumable” by automated systems, enabling frequent or event-driven updates and automated validation of security claims across the product supply chain.

Embedding these attestations directly into the development pipeline, security becomes an intrinsic part of the product’s DNA rather than a post-development checkbox.

- During a product’s design and development, machine-readable formats allow architects to define security requirements as code, which can then be automatically mapped to implementation evidence.
- For verification, these attestations enable automated gatekeeping; for instance, a deployment system can be configured to blocking any container that lacks a valid, machine-signed attestation of a successful vulnerability scan by default.

Furthermore, machine-readable attestations solve the "transparency gap" inherent in complex ecosystems. They provide a verifiable, tamper-evident trail of a product’s security posture from the source code to the final runtime environment. This level of transparency ensures that every stakeholder, from the developer to the end customer, has access to an objective, current view of the risk profile, effectively automating the trust relationship between producers and consumers.

Technical documentation can also benefit from machine-readable attestations. Instead of being created as a static snapshot at a particular point in time, the artifacts can help ensure that documentation stays accurate and current by displaying the security requirements, implementation evidence, and verification results.

This is a significant advantage, particularly in view of the CRA's requirement that manufacturers maintain technical documentation. Such documentation can be kept up to date and supported by verifiable implementation and test evidence.

For SME engineering teams, machine-readable attestation can support automation by employing these concepts:

- **Demonstrability**: The proactive capacity of a system and its development process to provide objective, machine-readable evidence that specific security requirements have been implemented. It represents the shift from "claiming" security in a static document to "showing" security through generated artifacts, such as signed build logs, automated test results, and standardised metadata.

- **Verifiability**: The ability for an independent party, whether an automated tool, a customer, or a regulator, to programmatically authenticate and validate the integrity of security claims. A verifiable system ensures that security attestations are transparent, tamper-evident, and mapped to a recognised root of trust, allowing for the continuous, low-friction audit of a product's security posture.
- **Reusability**: The ability to use existing attestations for further build on existing developments directly integrating cybersecurity in the development cycle and enabling an continuous cybersecurity improvement with minimal effort.
- **Reliability**: The ability to rely on existing attestations also for third party components, simplifying due diligence based on a structured attestation demonstrating security properties of a component with additional option for third party verification

5.2 Incentives

When security requirements are **demonstrable**, engineering teams move away from "security as an afterthought" and instead treat security as a primary functional requirement: every design decision, from the choice of a library to the architecture of a database, should be accompanied by the creation of a machine-readable artifact. This active generation of evidence allows teams to catch architectural flaws during the design phase, long before code is committed or products are shipped.

Also, while Security by Default focuses on delivering products that are secure "out of the box," **verifiability** provides the mechanism to ensure those defaults remain intact and effective. In a verifiable ecosystem, the "default" state is not just a configuration choice but a protected claim that can be automatically checked at every stage of the lifecycle.

Using **reusable** attestation enables the manufacturer to directly include cybersecurity into the development process and enabling the integration of cybersecurity controls in small use case specific packages leveraging existing agile methods and can also be included in quality gates in agile project management and tooling, e.g. cybersecurity as part of Definition of Ready and Done in Scrum

Relying on machine-processable attestation enables manufacturer to reduce effort for due diligence and supply chain management. A structured attestation enables integrators to simply pinpoint necessary security properties for due diligence and enhances trust with evidences and additional verification.

This creates a fail-safe environment where a system can refuse to operate if its security attestations, such as a signed proof that multi-factor authentication is enforced or that the latest vulnerability scan passed, are missing or invalid.

For SMEs, this automation eliminates the need for constant manual checks. It ensures the product's security baseline is both self-verifying and consistently visible to stakeholders.

5.3 Existing Frameworks and Initiatives

The ecosystem of machine-readable security has been rapidly evolving. Some of the key initiatives include foundational frameworks like **NIST's OSCAL (***Compliance as Code***)** and **OWASP's CycloneDX CDXA (***native security attestation***)**, which provide the structural "grammar" for automating compliance and supply chain transparency. Below are some of these initiatives that are complimentary with the proposed MRSM:

- **OSCAL**[24] (Open Security Controls Assessment Language): Developed by NIST, OSCAL is the premier framework for "Compliance as Code." It provides a standardised way to express security control catalogues, system security plans (SSPs), and assessment results. By using OSCAL, organisations can automate the generation of compliance documentation, allowing for "continuous authorisation" where the status of security controls is updated in real-time as evidence is collected.
- **CycloneDX**[25] (OWASP/ECMA-424): While originally a Software Bill of Materials (SBOM) format, CycloneDX has expanded into a full-stack transparency standard. Its Attestation (CDXA) capabilities allow organisations to document claims about security requirements alongside the BOM. It also includes specialised modules like VEX (Vulnerability Exploitability eXchange) to communicate whether a vulnerability actually affects a product, and CBOM (Cryptographic BOM) to inventory cryptographic assets for quantum readiness.
- **OpenSSF** (Open-Source Security Foundation): OpenSSF leads several projects, including **Security Insights**[26], a specification for projects to report security facts (like bug bounty info or security contacts) in a machine-processable YAML format. They also champion the **OpenSSF Scorecard**[27], which automatically assesses open-source projects against security best practices.
- **OWASP** (Open Web Application Security Project): Beyond CycloneDX, OWASP projects like the **ASVS**[28] (Application Security Verification Standard) provide the underlying requirements that these machine-readable formats aim to verify. Newer initiatives like the MLSVS (Machine Learning Security Verification Standard) are extending these principles into the AI/ML domain.
- **TC54 (Ecma International**)[29]: This technical committee is the formal standardisation body for CycloneDX. It focuses on the Transparency Exchange API, which aims to standardise how software transparency information (like SBOMs and attestations) is discovered and shared across the global supply chain.

5.4 Machine-Readable Security Manifest (MSRM) Example

This section describes an illustrative example of a machine-readable security manifest, referred to here as the MSRM. Instead of proposing a new schema or prescribing a specific format, the example is intended to demonstrate how security claims, supporting evidence, and verification of results could be expressed in a structured, machine-readable format to describe a product's security posture..

The MRSM is structured around a hierarchical data model that ensures every high-level security claim is backed by granular technical evidence. The proposed "cascade" ensures that the manifest is not merely a collection of assertions, but a verifiable chain of implementation:

- **Control Layer (Security objectives to be addressed)**:
  - Defines structured security objectives (for example, "protection against unauthorised access" or "secure update delivery"), which may be derived by Secure by Design and Default principles and/or aligned with regulatory or recognised cybersecurity frameworks;
- **Implementation Layer (How controls are implemented)**

[24] https://pages.nist.gov/OSCAL/
[25] https://cyclonedx.org/capabilities/attestations/
[26] https://openssf.org/projects/security-insights/
[27] https://openssf.org/projects/scorecard/
[28] https://owasp.org/www-project-application-security-verification-standard/
[29] https://tc54.org/tea/

- Provides claims describing the technical controls implemented to satisfy each objective, together with references to supporting evidence (e.g., "Implementation of TLS 1.3 with AES-256 encryption");
- Details the specific tools (e.g., OpenSSL version, specific compiler flags), versions, configurations (e.g., Strict-Transport-Security headers), and parameters used to instantiate the control;

- **Assessment and Verification Layer (How claims and evidence are validated)**
  - Links to verifiable outputs of automated validation processes, such as timestamped test results, static analysis (SAST) summaries, and cryptographic hashes of build artifacts.

In cases where component identifiers (such as PURL[30], CPE[31] or SWHID[32]) are referenced, the manifest can map security controls directly to specific software libraries or hardware modules listed in an SBOM. This approach would allow for a *multi-dimensional risk assessment*, where a user can verify not only that a product claims to have a "vulnerability management process" (claim) but also check that the specific third-party components listed in the SBOM were actually subjected to the security scans referenced in the manifest's verification layer.

Given that a manifest may contain sensitive technical details, such as specific compiler flags, internal test logs, or cryptographic configurations, it is often impractical or insecure to grant every stakeholder full visibility.

Technically, this access control is achieved through Scoped Attestations or Decentralised Identifiers (DIDs). Instead of sharing a single monolithic file, the manufacturer can issue cryptographically signed "sub-manifests." For example:

- **Public-Facing JSON**: Contains only high-level cryptographic signatures and compliance statements.
- **Restricted Technical Overlay**: A separate, encrypted JSON object containing detailed tool configurations and test telemetry, accessible only via a specific decryption key or an authenticated API endpoint (such as the *Transparency Exchange API*).

5.4.1 MSRM Example Schema

Below is a high-level description of an example MRSM schema

Layer	Domain	Function
Metadata & Attestation	Identity	Defines product identity, versioning, and the manufacturer's cryptographic signature.
Control Layer	Governance	Defines structured security objectives aligned with requirements, security principles, and/or applicable security policies or regulations.
Implementation Layer (Threat-Mitigation Map)	Operational	The core logic. Maps specific threats to implemented mitigation measures in line with secure by design principles, default settings, and human-readable descriptions.
Assessment and Verification Layer	Evidence	Provides machine-readable "pass/fail" results from automated gates and links to multi-format SBOMs.

[30] https://aboutcode.org/2023/purl-universal-software-package-identification/
[31] https://nvd.nist.gov/products/cpe
[32] https://www.swhid.org/

```
{
  "manifest_metadata": {
    "//": "Product Identity & Integrity",
    "version": "1.0.0",
    "timestamp": "ISO-8601-format",
    "attestation": "Digital-Signature-ID"
  },
  "requirement_layer": {
    "//": "Regulatory & Strategic Alignment",
    "security_objectives": [ "Mapping to CWE/OWASP/ENISA"]
  },
  "threat_mitigation_map": [
    {
      "threat_id": "T1...T5",
      "principle": "Design Principle (e.g., Least Privilege)",
      "description": "Narrative explanation for auditors",
      "secure_by_default_setting": {
        "//": "Technical Hardening Parameters",
        "ports": [443, 22],
        "uid": 1001,
        "mount": "Read-Only"
      },
      "verification_gate": {
        "//": "Automated Proof of Implementation",
        "method": "Audit-Tool-Name",
        "status": "PASS/FAIL",
        "evidence_hash": "SHA-256-Proof"
      }
    }
  ],
  "external_references": {
    "//": "Software Supply Chain Transparency",
    "sbom_cyclonedx": "URI-to-JSON",
    "sbom_spdx": "URI-to-JSON"
  }
}
```

5.4.2 MRSM Example Scenario

Scenario: Imagine you are a manufacturer of the *SafeGate-X1*, a specialised hardware controller used to manage security gates and water valves in a factory.

The Product**:**

- **A Standalone Linux-based Controller:** It sits in a physical cabinet, not a cloud.
- **The Application:** A C++ binary that listens for "Open/Close" commands and logs telemetry.
- **The User Interface:** A small local touch-screen and a restricted SSH port for maintenance.

The Goal: You want to prove to your industrial clients that the *SafeGate-X1* isn't just a "black box", but is built follows the secure by design principles of **Least Privilege**, **Minimal Attack Surface**, **Strong Identify**, and **vulnerability and patch management** . You will use the **MRSM** to provide the technical proof they need for their internal risk audits.

Disclaimer: This Machine-Readable Security Manifest (MRSM) and the SafeGate-X1 scenario are provided solely as conceptual models to illustrate the practical integration of security-by-design principles into automated verification workflows. They do not constitute formal legal or technical advice and should not be treated as a definitive proof of compliance with specific regulatory frameworks, such as the EU Cyber Resilience Act, or as a substitute for certified audit processes.

5.4.2.1 Threat model

You use the guidance on **Threat modelling** to define the product scope, assumptions and security objectives:

Category	Details
Purpose	Drive engineering decisions for binary hardening, port configuration, and user access control for the SafeGate-X1.
Scope Boundaries	**In-Scope:** SafeGate-X1 Firmware, Local API (HTTPS), Maintenance SSH, Local Touchscreen. **Out-of-Scope:** Corporate Network infrastructure, physical lock mechanics.
Assumptions	1. The local network is untrusted (hostile actors may have plugged into the switch). 2. Physical access to the cabinet is restricted but possible (tampering must be detected).
Security Objectives	**Integrity:** Prevent unauthorised gate triggers. **Availability:** Ensure the gate is operable during emergencies. **Safety:** Ensure "Fail-Safe" logic is never overridden by software.
Crown Jewels	Gate Actuator Control (GPIO), Cryptographic Signing Keys, Audit Logs.

We model the SafeGate-X1 as a set of interacting processes with varying trust levels.

- **Trust Boundary A (External/Network):** Separates the untrusted factory LAN from the device's listening services (HTTPS/SSH).
- **Trust Boundary B (User/System):** Separates the non-privileged `gate-service` user from the restricted `root` system functions.
- **Privileged Path (The Update Channel):** A high-integrity path from the Update Server to the internal flash storage.
- **Privileged Path (Admin Maintenance):** A short-lived, certificate-signed path for the Maintenance Engineer to access the OS shell.

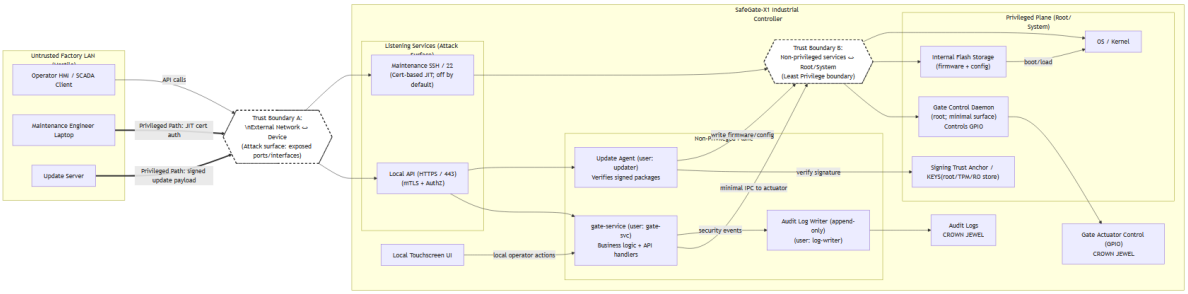


Figure 4 - Trust boundaries diagram[33]

ID	Abuse Case	Entry Point / Boundary	Likelihood	Impact	Rank
T1	RCE via Web API: Attacker exploits a buffer overflow in the web server to gain shell access.	Trust Boundary A (HTTPS)	Med	High	Top
T2	Privilege Escalation: Attacker uses a compromised web service to gain root and disable logs.	Trust Boundary B (System)	High	High	Top
T3	Port Scanning / Discovery: Attacker identifies legacy services (e.g., Telnet) left on by mistake.	Trust Boundary A (Network)	High	Med	**High**
T4	Credential Stuffing: Attacker attempts to log in via SSH using common default passwords.	Trust Boundary A (SSH)	High	High	**High**
T5	Binary Tampering: Attacker replaces the gate-logic binary with a version that keeps the gate open.	Local Storage	Low	High	**Med**

5.4.2.2 Controls, Defaults, and Verification

This section maps the threats to specific principles and the Machine-Readable Security Manifest (MRSM) verification.

Threat	Mitigation Control (Principle)	Secure-by-Default Setting	Verification Method (MRSM Gate)
T1 / T3	**Attack Surface Minimisation**: Use a "Distroless" build and a strict firewall.	All ports blocked except 443/22. No shell tools (sh, curl) in prod.	Nmap Scan Gate: Automated port audit must match the MRSM `exposed_ports` list.
T2	**Least Privilege**: Run app as non-root with Linux Capabilities.	App runs as `UID 1001`. No `SUDO` rights for the app user.	Systemd Audit: Automated check of `CapabilityBoundingSet` in service files.
T4	**Strong Identity**: Disable password auth; use signed SSH certs.	`PasswordAuthentication no` in `sshd_config`.	Config Validation: Script checks SSH config against "Hardened Baseline" before release.

[33] Built with https://mermaid.live

T5	**Vulnerability and patch management**: Signed updates and Immutable Filesystem.	Root filesystem is mounted Read-Only by default.	Mount Check: CI test verifies `/usr/bin` is not writable by any user.

5.4.2.3 SafeGate-X1 MRSM

How to understand the Hierarchy of Proof:

- **Control Layer**:
  - Strategic objectives, Threat mapping (T1–T5) - *"We must prevent unauthorized gate triggers (Integrity)."*
- **Implementation Layer**:
  - Principles (Least Privilege, Attack Surface), CWE/OWASP mapping - *"We apply Least Privilege and Attack Surface Minimisation."*
  - Technical settings (UID 1001, Read-Only FS, Port 443) - "*We run as `UID 1001`, use Linux Capabilities, and close all ports except 443."*
- **Assessment and Verification Layer**:
  - Release gates (Nmap, Lynis, Mount Check), SBOM hashes. - *"`Nmap` and `Lynis` confirm these settings; we provide the cryptographic hashes of the evidence."*

Attack Surface Minimisation (T1/T3)

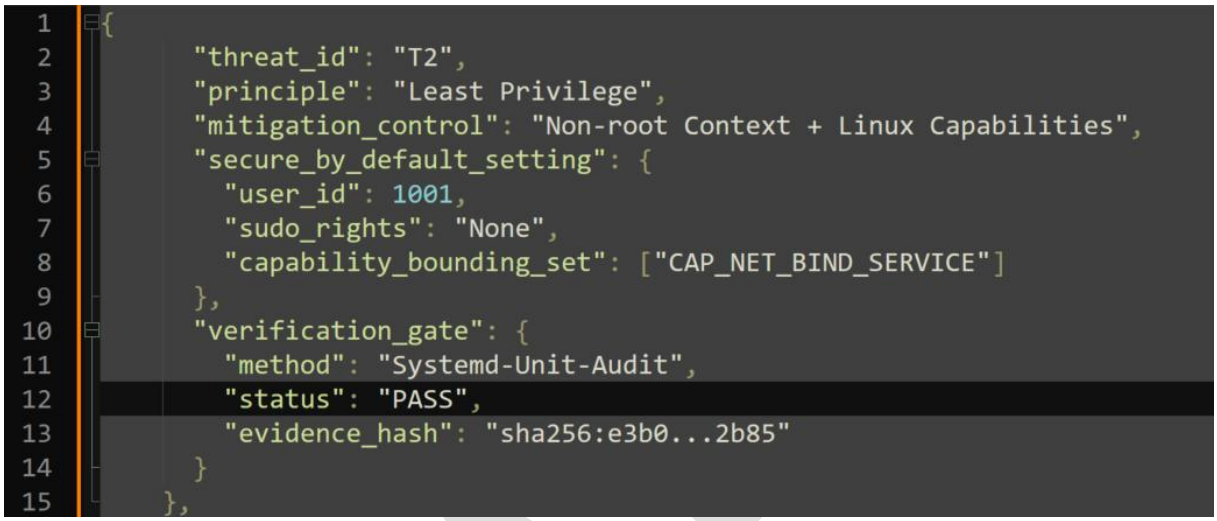
- **Logic:** If the threat is an external compromise (RCE), we remove the tools an attacker needs to escalate (sh, curl, etc.).
- **Implementation:** We use a **Distroless**[34] [35] base image.
- **Gate:** An automated **Nmap Scan** runs against the final image. It compares the active ports against the `ports_allowed` list in the JSON. If a mismatch is found, the `status` flips to `FAIL`, and the firmware is rejected.

```
{
    "threat_id": ["T1", "T3"],
    "principle": "Attack Surface Minimisation",
    "mitigation_control": "Distroless Build + Strict Firewall",
    "secure_by_default_setting": {
      "ports_allowed": [443, 22],
      "shell_tools": "Removed (sh, curl, wget excluded)",
      "build_type": "Distroless-Production"
    },
    "verification_gate": {
      "method": "Nmap-Scan-Gate",
      "status": "PASS",
      "evidence_hash": "sha256:7f8d...a11b"
    }
  },
```

Least Privilege (T2)

[34] https://buildroot.org/
[35] https://opensource.suse.com/bci-docs/guides/building-a-distroless-image/

- **Logic:** If the web service is compromised, it should not have permission to modify the system.
- **Implementation:** The application is assigned `UID 1001` with no entry in the `sudoers` file.
- **Gate:** A **Systemd Audit** script parses the service unit files. It verifies that `CapabilityBoundingSet` is restricted to only necessary network calls, ensuring the principle of Least Privilege is hard-coded into the OS execution.



Strong Identity (T4)

- **Logic:** Credential stuffing and brute-force attacks rely on passwords.
- **Implementation:** We disable password logins entirely at the configuration level.
- **Gate:** Before the "Gold Image" of the firmware is generated, a linter checks `sshd_config` to ensure `PasswordAuthentication` is set to `no`.

Vulnerability & Patch Management (T5)

- **Logic:** Even with a compromise, the attacker should not be able to achieve persistence by modifying system files.

- **Implementation:** The root filesystem is mounted as **Read-Only**.
- **Gate: CI** test runs a script inside a staging environment that attempts to `touch /usr/bin/test_file`. If the file is successfully created, the release gate fails, proving the "Secure-by-Default" setting was not correctly applied.

```
{
    "threat_id": "T5",
    "principle": "Vulnerability & Patch Management",
    "mitigation_control": "Immutable Filesystem + Signed Updates",
    "secure_by_default_setting": {
      "root_fs_mount": "Read-Only",
      "update_verification": "Hardware-HSM-Verification"
    },
    "verification_gate": {
      "method": "Mount-Check-CI",
      "status": "PASS",
      "evidence_hash": "sha256:b1c2...d3e4"
    }
}
```

5.4.2.4 Third-party Verification

Principle	Threat ID	Technical Evidence (What to look for)	Verification Gate (How it was tested)	Auditor "Spot-Check" Step
Attack Surface Minimisation	T1, T3	Distroless image; empty `/bin/` and `/usr/bin/` of non-essential tools.	`Nmap-Scan-Gate`	Run `ls /bin/sh` on the device; it should return "File not found."
Least Privilege	T2	`systemd` unit file showing `User=1001` and restricted capabilities.	`Systemd-Unit-Audit`	Run `ps aux` and verify the gate process is owned by `gate-service`, not `root`.
Strong Identity	T4	`sshd config` with `PasswordAuthentication no`.	`SSH-Config-Validation`	Attempt to SSH using a password; the connection must be immediately rejected.
Vulnerability & Patch Mgmt	T5	Partition table showing `/` mounted as `ro` (Read-Only).	`Mount-Check-CI`	Run `touch /usr/bin/test`; it should return "Read-only file system."

6. Bibliography / References

1. **Chidukwani, A., Zander, S., & Koutsakis, P**. - Cybersecurity preparedness of small-to-medium businesses: A Western Australia study with broader implications. Computers & Security, 145, 104026, 2024. https://doi.org/10.1016/j.cose.2024.104026
2. **ENISA**, ENISA Cybersecurity for SMEs – Challenges and Recommendations, 2021: https://www.enisa.europa.eu/publications/enisa-report-cybersecurity-for-smes
3. **European Commission.** SME definition. Accessed 2026. https://single-market-economy.ec.europa.eu/smes/sme-fundamentals/sme-definition_en
4. **European Commission.** Small and medium-sized enterprises (SMEs). Accessed 2026. https://eur-lex.europa.eu/EN/legal-content/glossary/small-and-medium-sized-enterprises.html
5. **ENISA**. Cybersecurity for SMEs Challenges and Recommendations. 2021. https://www.enisa.europa.eu/sites/default/files/publications/ENISA%20Report%20-%20Cybersecurity%20for%20SMES%20Challenges%20and%20Recommendations.pdf
6. **ENISA**. NIS Investments – Cybersecurity Policy Assessment. 2023. https://www.enisa.europa.eu/sites/default/files/publications/CSPA%20-%20NIS%20Investments%20-%202023.pdf
7. **ENISA**. Guidelines for Securing the Internet of Things, Secure supply chain for IoT. 2020. https://www.enisa.europa.eu/sites/default/files/publications/ENISA%20Report%20-%20Guidelines%20for%20Securing%20the%20Internet%20of%20Things.pdf
8. **ENISA**. Baseline Security Recommendations for IoT. 2017 https://www.enisa.europa.eu/sites/default/files/publications/WP2017%20O-1-1-2%201%20Baseline%20Security%20Recommendations%20for%20IoT%20in%20the%20context%20of%20CII_FINAL.pdf
9. **ENISA**. Good practices for security of IoT, Secure Software Development Lifecycle, 2019. https://www.enisa.europa.eu/sites/default/files/publications/WP2019%20-%20O.1.1.1%20Good%20practices%20for%20security%20of%20IoT.pdf
10. **OWASP**. Authorization Testing (Web Security Testing Guide). Accessed 2026. https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/05-Authorization_Testing/README
11. **OWASP**. Threat Modeling Cheat Sheet. Accessed 2026. https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html
12. **OWASP**. Threat Modeling Process. Accessed 2026. https://owasp.org/www-community/Threat_Modeling_Process
13. **NIST**. SP 800-160 Vol. 1 Rev. 1: Engineering Trustworthy Secure Systems. 2022. https://doi.org/10.6028/NIST.SP.800-160v1r1
14. **OWASP**. SAMM Model (Software Assurance Maturity Model). Accessed 2026. https://owaspsamm.org/model/
15. **OWASP**. OWASP Secure by Design Framework. 2025 https://owasp.org/www-project-secure-by-design-framework/
16. **OWASP.** Threat modeling. Accessed 2026. https://owasp.org/www-community/Threat_Modeling
17. **ETSI**. EN 303 645 V3.1.3: Cyber Security for Consumer Internet of Things Baseline Requirements. 2024. https://www.etsi.org/deliver/etsi_en/303600_303699/303645/03.01.03_60/en_303645v030103p.pdf
18. **NIST**. SP 800-213: IoT Device Cybersecurity Guidance for the Federal Government. 2021. https://doi.org/10.6028/NIST.SP.800-213

19. **ENISA**. Baseline Security Recommendations for IoT in the context of CII. 2017. https://www.enisa.europa.eu/sites/default/files/publications/WP2017%20O-1-1-2%201%20Baseline%20Security%20Recommendations%20for%20IoT%20in%20the%20context%20of%20CII_FINAL.pdf
20. **MITRE**. CWE Top 25 Most Dangerous Software Weaknesses. Accessed 2026. https://cwe.mitre.org/top25/
21. **OWASP**. Top 10 2025. Accessed 2026. https://owasp.org/Top10/2025/
22. **AboutCode**. PURLs of Wisdom: Universal Software Package Identification. 2023. https://aboutcode.org/2023/purl-universal-software-package-identification/
23. **NIST** NVD. CPE: Common Platform Enumeration. Accessed 2026. https://nvd.nist.gov/products/cpe
24. **NIST**. OSCAL: Open Security Controls Assessment Language. Accessed 2026. https://pages.nist.gov/OSCAL/
25. **CycloneDX**. Attestations Capability. Accessed 2026. https://cyclonedx.org/capabilities/attestations/
26. **CycloneDX** Authoritative Guide to Attestations, 2024, https://cyclonedx.org/guides/OWASP_CycloneDX-Authoritative-Guide-to-Attestations-en.pdf
27. **OpenSSF**. Security Insights Specification. Accessed 2026. https://openssf.org/projects/security-insights/
28. **OpenSSF**. Scorecard. Accessed 2026. https://openssf.org/projects/scorecard/
29. **OWASP**. Application Security Verification Standard (ASVS). Accessed 2026. https://owasp.org/www-project-application-security-verification-standard/
30. **OWASP**. Application Security Verification Standard (ASVS). 2025. https://github.com/OWASP/ASVS/raw/v5.0.0/5.0/OWASP_Application_Security_Verification_Standard_5.0.0_en.pdf
31. **TC54**. Transparency Exchange API Specification. Accessed 2026. https://tc54.org/tea/
32. **Buildroot**. Buildroot, Making Embedded Linux Easy. Accessed 2026. https://buildroot.org/
33. **openSUSE**. How to build a distroless image using SLE BCI. Accessed 2026. https://opensource.suse.com/bci-docs/guides/building-a-distroless-image/

A Annex: Abbreviations

API	Application Programming Interface
ASVS	Application Security Verification Standard
CBOM	Cryptography Bill of Materials
CDXA	CycloneDX Attestations
CI	Configuration Item
CI/CD	Continuous Integration and Continuous Delivery
CLI	Command Line Interface
CPE	Common Platform Enumeration
CWE	Common Weakness Enumeration
DAST	Dynamic Application Security Testing
DevOps	Development and Operations
DevSecOps	Development, Security, and Operations
EOL	End of Life
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IaC	Infrastructure as Code
IAM	Identity and Access Management
IR	Incident Response
JSON	JavaScript Object Notation
JWT	JSON Web Token
MFA	Multi Factor Authentication
MITM	Man in the Middle
MRSM	Machine-Readable Security Manifest
NIST	National Institute of Standards and Technology
NTP	Network Time Protocol
OAuth	Open Authorization
OIDC	OpenID Connect
OSCAL	Open Security Controls Assessment Language
OTA	Over the Air
OWASP	Open Worldwide Application Security Project
PR	Pull Request
PURL	Package URL
RBAC	Role-based Access Control

RPO	Recovery Point Objective
RTO	Recovery Time Objective
SAST	Static Application Security Testing
SBOM	Software Bill of Materials
SCA	Software Composition Analysis
SCVS	Software Component Verification Standard
SDK	Software Development Kit
SDLC	Software Development Life Cycle
SIEM	Security Information and Event Management
SLA	Service Level Agreement
SLSA	Supply-chain Levels for Software Artifacts
SQL	Structured Query Language
SSH	Secure Shell
STRIDE	Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege
TLS	Transport Layer Security
UI	User Interface
UX	User experience
VDP	Vulnerability Disclosure Program
VEX	Vulnerability Exploitability eXchange
YAML	YAML Ain't Markup Language or Yet Another Markup Language

B Annex: CRA Essential Requirements

ID	CRA text
ANNEX-1.PT1.1	Products with digital elements shall be designed, developed and produced in such a way that they ensure an appropriate level of cybersecurity based on the risks.
ANNEX-1.PT1.2	On the basis of the cybersecurity risk assessment referred to in Article 13(2) and where applicable, products with digital elements shall:
ANNEX-1.PT1.2.a	be made available on the market without known exploitable vulnerabilities;
ANNEX-1.PT1.2.b	be made available on the market with a secure by default configuration, unless otherwise agreed between manufacturer and business user in relation to a tailor-made product with digital elements, including the possibility to reset the product to its original state;
ANNEX-1.PT1.2.c	ensure that vulnerabilities can be addressed through security updates, including, where applicable, through automatic security updates that are installed within an appropriate timeframe enabled as a default setting, with a clear and easy-to-use opt-out mechanism, through the notification of available updates to users, and the option to temporarily postpone them;
ANNEX-1.PT1.2.d	ensure protection from unauthorised access by appropriate control mechanisms, including but not limited to authentication, identity or access management systems, and report on possible unauthorised access;
ANNEX-1.PT1.2.e	protect the confidentiality of stored, transmitted or otherwise processed data, personal or other, such as by encrypting relevant data at rest or in transit by state of the art mechanisms, and by using other technical means;
ANNEX-1.PT1.2.f	protect the integrity of stored, transmitted or otherwise processed data, personal or other, commands, programs and configuration against any manipulation or modification not authorised by the user, and report on corruptions;
ANNEX-1.PT1.2.g	process only data, personal or other, that are adequate, relevant and limited to what is necessary in relation to the intended purpose of the product with digital elements (data minimisation);
ANNEX-1.PT1.2.h	protect the availability of essential and basic functions, also after an incident, including through resilience and mitigation measures against denial-of-service attacks;
ANNEX-1.PT1.2.i	minimise the negative impact by the products themselves or connected devices on the availability of services provided by other devices or networks;
ANNEX-1.PT1.2.j	be designed, developed and produced to limit attack surfaces, including external interfaces;
ANNEX-1.PT1.2.k	be designed, developed and produced to reduce the impact of an incident using appropriate exploitation mitigation mechanisms and techniques;
ANNEX-1.PT1.2.l	provide security related information by recording and monitoring relevant internal activity, including the access to or modification of data, services or functions, with an opt-out mechanism for the user;

ANNEX-1.PT1.2.m	provide the possibility for users to securely and easily remove on a permanent basis all data and settings and, where such data can be transferred to other products or systems, ensure that this is done in a secure manner.
ANNEX-1.PT2.1	identify and document vulnerabilities and components contained in products with digital elements, including by drawing up a software bill of materials in a commonly used and machine-readable format covering at the very least the top-level dependencies of the products;
ANNEX-1.PT2.2	in relation to the risks posed to products with digital elements, address and remediate vulnerabilities without delay, including by providing security updates; where technically feasible, new security updates shall be provided separately from functionality updates;
ANNEX-1.PT2.3	apply effective and regular tests and reviews of the security of the product with digital elements;
ANNEX-1.PT2.4	once a security update has been made available, share and publicly disclose information about fixed vulnerabilities, including a description of the vulnerabilities, information allowing users to identify the product with digital elements affected, the impacts of the vulnerabilities, their severity and clear and accessible information helping users to remediate the vulnerabilities; in duly justified cases, where manufacturers consider the security risks of publication to outweigh the security benefits, they may delay making public information regarding a fixed vulnerability until after users have been given the possibility to apply the relevant patch;
ANNEX-1.PT2.5	put in place and enforce a policy on coordinated vulnerability disclosure;
ANNEX-1.PT2.6	take measures to facilitate the sharing of information about potential vulnerabilities in their product with digital elements as well as in third-party components contained in that product, including by providing a contact address for the reporting of the vulnerabilities discovered in the product with digital elements;
ANNEX-1.PT2.7	provide for mechanisms to securely distribute updates for products with digital elements to ensure that vulnerabilities are fixed or mitigated in a timely manner and, where applicable for security updates, in an automatic manner;
ANNEX-1.PT2.8	ensure that, where security updates are available to address identified security issues, they are disseminated without delay and, unless otherwise agreed between a manufacturer and a business user in relation to a tailor-made product with digital elements, free of charge, accompanied by advisory messages providing users with the relevant information, including on potential action to be taken.

C Annex: Mapping Security Principles to CRA Annex I Essential Requirements

Principle	CRA Essential requirement	Implementation support
Trust boundaries and Threat Modelling	**ANNEX-1.PT1.1**	Supports identification and assessment of cybersecurity risks by making trust assumptions, assets, and attack paths explicit during design.
	ANNEX-1.PT1.2.d	Supports protection from unauthorised access by clarifying where authentication and access controls are required between trust boundaries.
	ANNEX-1.PT1.2.e	Supports confidentiality protections by identifying where data crosses trust boundaries and requires protection.
	ANNEX-1.PT1.2.f	Supports integrity protection by identifying where data, commands, or configuration cross boundaries and may require integrity controls.
	ANNEX-1.PT1.2.j	Supports attack surface limitation by identifying exposed interfaces and unnecessary trust relationships.
Least privilege	**ANNEX-1.PT1.2.d**	Supports protection from unauthorised access by limiting what authenticated users, services, and processes are permitted to access or perform.
	ANNEX-1.PT1.2.f	Supports integrity protection by limiting which identities are authorised to modify data, programs, or configuration.
	ANNEX-1.PT1.2.g	Supports data minimisation by limiting access to data to what is necessary for the intended purpose.
Strong identity and auth architecture	**ANNEX-1.PT1.2.d**	Supports access protection by defining how identities are authenticated and managed across interfaces.
	ANNEX-1.PT1.2.l	Supports logging and monitoring of authentication and access-related activity.
Attack surface minimisation	**ANNEX-1.PT1.2.b**	Supports secure-by-default configurations by reducing enabled features and services at initial deployment.

	ANNEX-1.PT1.2.j	Supports attack surface limitation by reducing unnecessary interfaces, services, and exposed functionality.
Defence in depth	**ANNEX-1.PT1.2.h**	Supports availability and resilience by relying on multiple layers of protection rather than a single control.
	ANNEX-1.PT1.2.k	Supports impact reduction by applying multiple mitigation mechanisms that limit the effect of exploitation.
Open Design	**ANNEX-1.PT2.3**	Supports effective testing and review by using designs that can be examined and assessed.
	ANNEX-1.PT2.4	Supports vulnerability disclosure by aligning with transparent communication of security issues and fixes.
Lifecycle management	**ANNEX-1.PT1.2.c**	Supports the ability to address vulnerabilities over the product lifetime through updates.
	ANNEX-1.PT1.2.m	Supports secure decommissioning by providing mechanisms to remove data and settings at end of use.
	ANNEX-1.PT2.2	Supports timely remediation of vulnerabilities throughout the supported lifecycle.
	ANNEX-1.PT2.7	Supports controlled and secure distribution of updates over time.
User centric design	**ANNEX-1.PT1.2.b**	Supports secure-by-default configurations that users can reasonably adopt and maintain.
Secure coding practices	**ANNEX-1.PT1.2.a**	Supports release without known exploitable vulnerabilities by identifying and addressing issues during development.
	ANNEX-1.PT2.1	Supports identification and documentation of vulnerable components during development.
	ANNEX-1.PT2.3	Supports regular testing and review of product security during development and before release.
Logging, monitoring, and alerting	**ANNEX-1.PT1.2.d**	Supports detection and reporting of unauthorised access attempts.
	ANNEX-1.PT1.2.l	Supports recording and monitoring of security-relevant internal activity.
Configuration and change management	**ANNEX-1.PT1.2.f**	Supports protection of configuration integrity by controlling, reviewing, and rolling back configuration changes.

	ANNEX-1.PT1.2.d	Supports protection from unauthorised access by restricting who can deploy or modify configurations, particularly in production environments.
Incident response and recovery	**ANNEX-1.PT1.2.h**	Supports recovery and continued availability of essential functions after incidents.
	ANNEX-1.PT1.2.k	Supports reduction of incident impact through prepared response and mitigation actions.
Vulnerability and patch management	**ANNEX-1.PT2.1**	Supports identification and tracking of vulnerable components and dependencies throughout the product lifecycle.
	ANNEX-1.PT2.2	Supports timely triage and remediation of identified vulnerabilities based on risk.
	ANNEX-1.PT2.4	Supports public disclosure of fixed vulnerabilities.
	ANNEX-1.PT2.5	Supports coordinated handling and disclosure of vulnerabilities.
	ANNEX-1.PT2.6	Supports mechanisms for receiving vulnerability reports from external parties.
	ANNEX-1.PT2.7	Supports secure distribution of patches and updates.
	ANNEX-1.PT2.8	Supports timely dissemination of security updates and related user guidance.
Supply chain controls	**ANNEX-1.PT1.2.a**	Supports reduction of exploitable vulnerabilities introduced through compromised components.
	ANNEX-1.PT2.1	Supports transparency of components and dependencies through SBOM generation.
	ANNEX-1.PT2.7	Supports secure distribution of updates through protected build and release channels.
Minimisation of Default Services	**ANNEX-1.PT1.2.b**	Supports secure-by-default configuration by disabling non-essential features and services at initial deployment.
	ANNEX-1.PT1.2.i	Supports minimising negative impact on other services by disabling non-essential functionality that could otherwise be abused or misused.
	ANNEX-1.PT1.2.j	Supports limitation of attack surfaces by reducing exposed services and interfaces by default.
Restrictive Initial Access	**ANNEX-1.PT1.2.b**	Supports secure-by-default access settings by avoiding permissive initial credentials and permissions.

	ANNEX-1.PT1.2.d	Supports protection from unauthorised access by enforcing restrictive authentication and access control settings at first use.
Secure Communication by Default	**ANNEX-1.PT1.2.b**	Supports secure-by-default configuration by enforcing encrypted and authenticated communications from initial connection.
	ANNEX-1.PT1.2.e	Supports confidentiality of transmitted data by applying encryption to communications from first use.
	ANNEX-1.PT1.2.f	Supports integrity of transmitted protection by preventing unauthorised modification of data in transit.
Unique device identity and secrets by default	**ANNEX-1.PT1.2.b**	Supports secure-by-default configuration by avoiding shared credentials and shared cryptographic material.
	ANNEX-1.PT1.2.d	Supports protection from unauthorised access by avoiding shared credentials and enforcing unique device authentication.
	ANNEX-1.PT1.2.e	Supports confidentiality by preventing reuse or leakage of shared secrets across devices.
Mandatory Security Onboarding	**ANNEX-1.PT1.2.b**	Supports secure-by-default configuration by requiring critical security features to be configured before initial exposure.
	ANNEX-1.PT1.2.d	Supports protection from unauthorised access by ensuring authentication and access controls are configured at first use.
Automated Maintenance and Updates	**ANNEX-1.PT1.2.b**	Supports a secure-by-default posture by enabling automatic security updates.
	ANNEX-1.PT1.2.c	Supports the ability to address vulnerabilities through automatic security updates enabled by default.
	ANNEX-1.PT2.2	Supports timely remediation of vulnerabilities by reducing reliance on manual patch deployment.
	ANNEX-1.PT2.7	Supports secure and timely distribution of security updates by enabling automated update mechanisms.
Transparent Security Posture	**ANNEX-1.PT1.2.b**	Supports secure-by-default configuration by guiding users back to a secure baseline when insecure choices are made.
	ANNEX-1.PT1.2.l	Supports security-related information by informing users of security-relevant state changes and risks.

Secure Recovery and Ownership Lifecycle	**ANNEX-1.PT1.2.b**	Supports secure-by-default configuration by providing safe recovery and reset paths that restore a secure baseline.
	ANNEX-1.PT1.2.d	Supports protection from unauthorised access during recovery, reset, and ownership transfer processes.
	ANNEX-1.PT1.2.m	Supports secure removal of data and settings during factory reset, decommissioning, or ownership transfer.

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