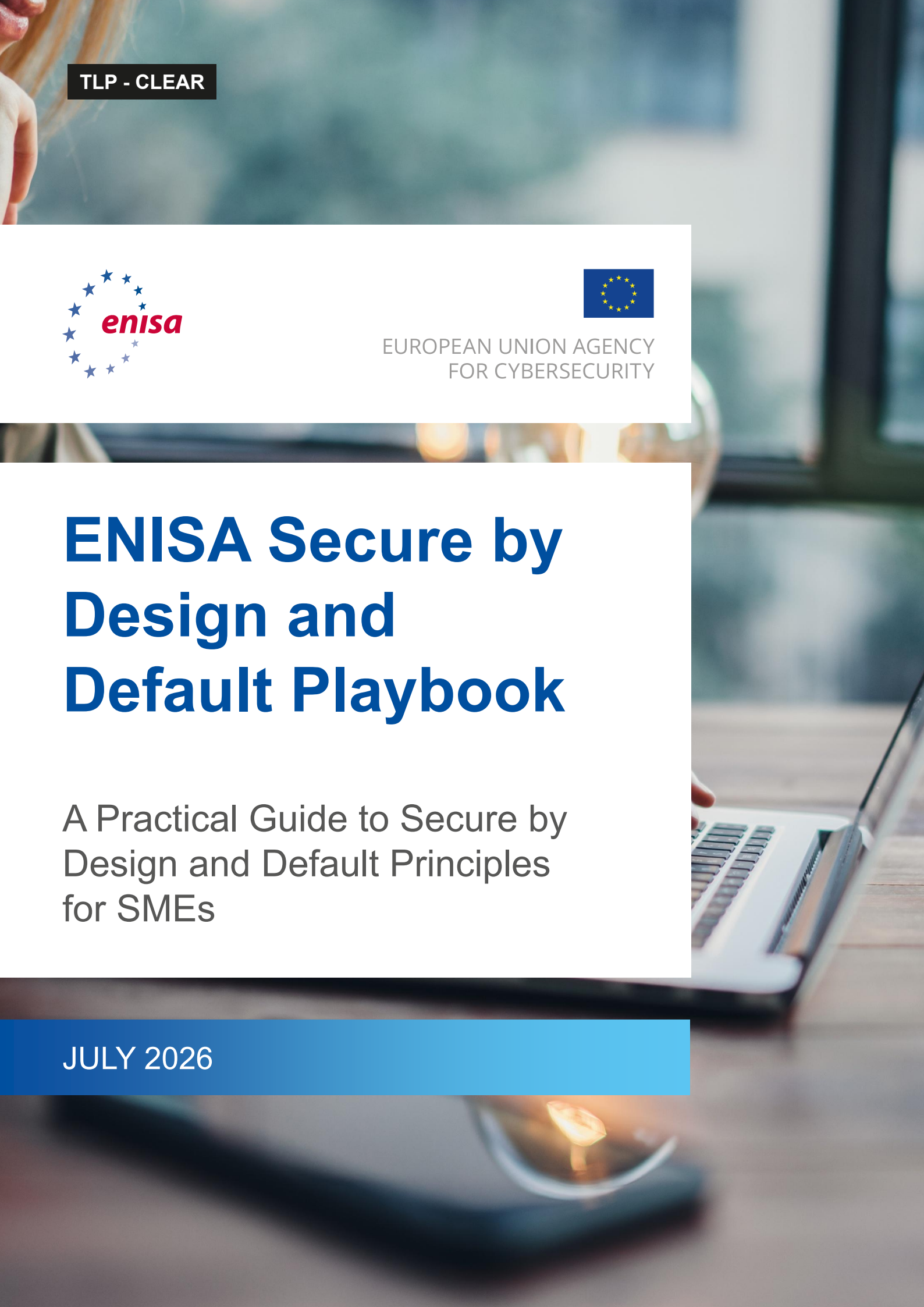

TLP - CLEAR

EUROPEAN UNION AGENCY
FOR CYBERSECURITY

ENISA Secure by Design and Default Playbook

A Practical Guide to Secure by Design and Default Principles for SMEs

JULY 2026

About ENISA

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

CONTACT

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

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.

PDF ISBN 978-92-9204-802-0 doi:10.2824/4422633 TP-01-26-016-EN-N

Table of contents

Note on this version

The current version of this report was prepared following a public consultation held by ENISA during April 2026 and May 2026. ENISA received 28 contributions from public and private stakeholders, cybersecurity experts, product manufacturers, software development practitioners and the open-source community. We would like to publicly thank all stakeholders who provided their views and feedback. All contributions have been reviewed and, to the maximum extent possible, the recommendations and observations have been addressed in this final version. This report may undergo regular revisions.

Acknowledgements

ENISA would like to acknowledge the following individuals and organisations that contributed to the public consultation and agreed to be publicly named.

Name	Affiliation
Adam Shostack	OWASP Foundation
Anthony Harrison	APH10
Arne Padmos	
Biserka Radeva	Ministry of Electronic Governance
Bob Lord	Software Safety Institute
Brunn Hilmar	Mettler-Toledo International
Cédric LEVY-BENCHETON	cetome
Dave Russo	Red Hat LLC
Diana Rossi	
Dr. Thomas Termin	WITTE Automotive / Institute for Security Systems
Friedhelm Becker	
Ioana Nilwik	
J B Toby	Avolites Ltd
Joan Romero Chaparro	craevidence.com
Johan Sydseter	OWASP Foundation
Juan Rico	Eclipse Foundation
Jukka Ruohonen	University of Southern Denmark
Kenichi Ikegami	JTEKT ELECTRONICS Corporation
Lauren Zabierek	Software Safety Institute
Madalin Neag	OpenSSF/The Linux Foundation
Martin Schneider	Fraunhofer FOKUS
Martin Zimpel	BSI
Matthew Coles	OWASP Foundation
Max Alejandro Gómez Sánchez Vergaray	OWASP Foundation
Michael Schuster	BSI
Nikos Mavrogiannopoulos	ASSA ABLOY
Patrick Lamplmair	Tributech
Philippe BOINOT	ANSSI
Sanjay Kumar Sirurmath	
Sarah Fluchs	admeritia
Sebastien Deleersnyder	OWASP Foundation
Sebastien Deleersnyder	OWASP Foundation
Thierry Dupont	
Zoe Braiterman	OWASP Foundation

Executive summary

Modern products with digital elements are increasingly expected to be **secure by design** and **secure by default**. However, many organisations – in particular small and medium-sized enterprises, in which development teams are small and security expertise can be limited – may face distinct challenges in applying these concepts consistently, requiring targeted solutions.

This report puts forward a set of principles and tangible guidance on the application of secure by design and default requirements throughout the life cycle 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 **secure by design principles** are organised into two groups, namely **architectural foundations** and **operational integrity**. The former addresses how the system is designed, implemented and built, while the latter focuses on how the system is distributed, installed, managed and maintained. Similarly, the **secure by default principles** are grouped into the categories **default hardening** and **guided protection**. The former ensures that products start in a secure and restrictive state, while the latter aims to support 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 of the principles presented in this report to the essential requirements set out in Annex I to the European Union's Cyber Resilience Act.

The report also presents an illustrative example of a machine-processable attestation, demonstrating how voluntary, manufacturer-issued machine-processable artefacts are able 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-processable attestations can 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 secure by design and default properties are maintained over time. This capability is particularly valuable for small and medium-sized enterprises 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

Secure by design represents a fundamental shift in product security, with protective measures embedded from conception rather than retrofitted post-development. For small and medium-sized enterprises (SMEs), including microenterprises, that manufacture products, the transition to a secure by design approach presents distinct challenges requiring tailored solutions ([1]).

Secure by design is increasingly expected, including under the European Union's Cyber Resilience Act (CRA) ([2]). This report does not provide legal guidance but offers practical, technically grounded approaches that can support manufacturers in applying the relevant principles in practice. It is intended as a practical starting point, rather than an exhaustive or prescriptive implementation framework. Its application should be adapted to the product's intended use, context and associated risks and to the applicable requirements. For reference, Annex C provides an indicative mapping of the secure by design and default principles described in this report to the essential requirements set out in Annex I to the CRA, which are themselves listed in Annex B.

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 takes into account the persistent challenges that SMEs face in translating security principles into engineering practice, acknowledging limitations on 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 measurable improvement.** Establish measurable baselines and use relevant evidence to assess whether security processes and controls operate as intended, to guide improvement.

1.1.1 Intended outcomes

Applying this playbook is intended to contribute to:

- fewer recurring and preventable vulnerabilities;

([1]) Chidukwani, A., Zander, S. and Koutsakis, P., 'Cybersecurity preparedness of small-to-medium businesses: A Western Australia study with broader implications', *Computers & Security*, Vol. 145, 2024, https://doi.org/10.1016/j.cose.2024.104026. 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]) Regulation (EU) 2024/2847 of the European Parliament and of the Council of 23 October 2024 on horizontal cybersecurity requirements for products with digital elements and amending Regulations (EU) No 168/2013 and (EU) 2019/1020 and Directive (EU) 2020/1828 (Cyber Resilience Act) (OJ L, 2024/2847, 20.11.2024, ELI: http://data.europa.eu/eli/reg/2024/2847/oj).

- products delivered with safer default configuration and reduced unnecessary exposure;
- earlier identification and treatment of security risks;
- better visibility and stronger protection of dependencies, build processes, releases and updates;
- faster detection, remediation and communication of vulnerabilities;
- reduced likelihood and impact of cybersecurity incidents throughout the product life cycle.

These outcomes benefit users and operators through safer and more resilient products. Manufacturers benefit through more consistent security decisions, while customers and integrators benefit through clearer and more traceable security evidence.

1.2 Scope and methodology

This guidance was developed with SMEs, including microenterprises, in mind, particularly SME manufacturers developing products with digital elements, such as embedded software, internet of things (IoT) devices, connected systems, stand-alone software and any hardware incorporating programmable components. It addresses security requirements throughout the product life cycle, from product design to decommissioning.

An SME is defined as an organisation with fewer than 250 employees and an annual turnover below EUR 50 million ([3]). SME manufacturers typically face ([4]):

- budget constraints – limited financial resources for security investments ([5]);
- 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, and relevant guidance from the US National Institute of Standards and Technology (NIST) and the Open Worldwide Application Security Project (OWASP) ([6]). We identified common requirements and implementation patterns, which were evaluated against SME capabilities to determine feasibility and adaptation requirements.

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

Cybersecurity threats, technologies and good practices continue to evolve. This playbook may therefore be updated or complemented over time by additional guidance and resources, such as the ENISA

([3]) https://eur-lex.europa.eu/EN/legal-content/glossary/small-and-medium-sized-enterprises.html; https://single-market-economy.ec.europa.eu/smes/sme-fundamentals/sme-definition_en.
([4]) 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.
([5]) ENISA, *NIS Investments – Cybersecurity policy assessment*, 2023, https://www.enisa.europa.eu/sites/default/files/publications/CSPA%20-%20NIS%20Investments%20-%202023.pdf.
([6]) For a complete list of references used in preparing this report, see the bibliography.

technical advisories ([7]). For example, while embedded, field-level and industrial products fall within the scope of this report, the guidance may need to be adapted or supplemented to account for their specific technical and operational constraints.

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 secure by design and default principles in their product development life cycle.

The primary groups that will benefit from this guidance include

- **Software developers and 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 need for fundamental security resilience.
- **SME security leads.** Personnel tasked with interpreting enterprise-grade frameworks for organisations with limited budgets, specialised niches or lean teams.
- **System architects.** Designers aiming to build robust infrastructures that prioritise security from the initial conceptual phase.

1.4 Structure of the report

The structure of the report is outlined below.

- **Section 2** covers the product security life cycle 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 showing how each of the activities can lead to tangible deliverables.
- **Section 3** covers fundamental secure by design and secure by default principles. They are explained in an easy-to-understand manner. This section lays the foundation for the next sections. The report also includes a mapping of these principles to other ENISA security best practices and well-established resources, such as MITRE's common weakness enumeration (CWE) and OWASP's top 10 critical security risks for 2025.
- **Section 4**, the most extensive part of the report, details each of the **22 principles** by covering, for each, the objective, key elements to consider when applying it in an engineering context, means of collecting 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 machine-processable attestation, offering an example of the implementation of a hierarchical data model that ensures that every high-level security claim is backed by granular technical evidence.
- **The annexes** summarise the CRA essential cybersecurity requirements, providing a mapping of the secure by design and default principles described in this report to those requirements, to show how the principles can support the implementation of the CRA.

([7]) For an example of this type of guidance, see ENISA, *ENISA technical advisory for secure use of package managers*, 2026, https://www.enisa.europa.eu/sites/default/files/2026-03/ENISA%20Technical%20Advisory%20-%20Package_Managers_Final.pdf.

SECTION 2

Secure by design and default across a product's life cycle

2. Secure by design and default across a product's life cycle

Secure by design and secure by default require more than applying principles during development. They must be operationalised end to end across the entire product life cycle, from the initial concept to the product's eventual decommissioning. This life-cycle view is especially important for connected products; where evolving threats, supply-chain dependencies and long-lived deployments can erode security despite otherwise sound design if governance and assurance do not persist over time.

ENISA's report *Guidelines for Securing the Internet of Things* highlights 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' ([8]). 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 life cycle

As highlighted in the ENISA report *Baseline security recommendations for IoT in the context of critical information infrastructures* ([9]), security must be considered throughout the entire product life cycle. Regardless of the production model used (V-model or agile), the following life-cycle processes require explicit security consideration to support product security.

- **Requirements.** As part of this process, user, business, functional and security requirements are determined. These requirements reflect the intended use of the product and will be translated into specifications that will guide design, development and maintenance/deployment decisions at later stages.
- **Design.** As part of the design process, the architecture and the design of the product are created. This process involves the creation of a set of documents that describe how the product requirements, including security requirements, will be translated into system specifications and essentially how the product will work.
- **Implementation.** As part of the implementation process, specifications and software design artefacts are implemented in the product.
- **Verification**: This process involves all necessary steps to verify that the developed product actually meets the identified requirements and design principles of the previous processes.
- **Deployment.** This process follows the acceptance of the product subject to successful testing as part of the previous processes – that is, it takes place after it has been approved for release. It involves integrating all necessary elements of the solution into the production environment and its deployment.

([8]) 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.

([9]) ENISA, *Baseline security recommendations for IoT in the context of critical information infrastructures*, 2017, https://www.enisa.europa.eu/publications/baseline-security-recommendations-for-iot; see also 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.

- **Maintenance and disposal.** Solutions deployed in production need to be constantly maintained to ensure the availability and integrity of the functionality provided. When the solution becomes obsolete, it is important to provide data erasure mechanisms that ensure secure disposal and preserve privacy management.

These processes are presented as groupings, not a mandatory sequential model. They may overlap, occur iteratively and be revisited throughout a product's life cycle. For example, in agile environments, security requirements may be captured and progressively refined through backlog items, user stories, acceptance criteria and the definition of done.

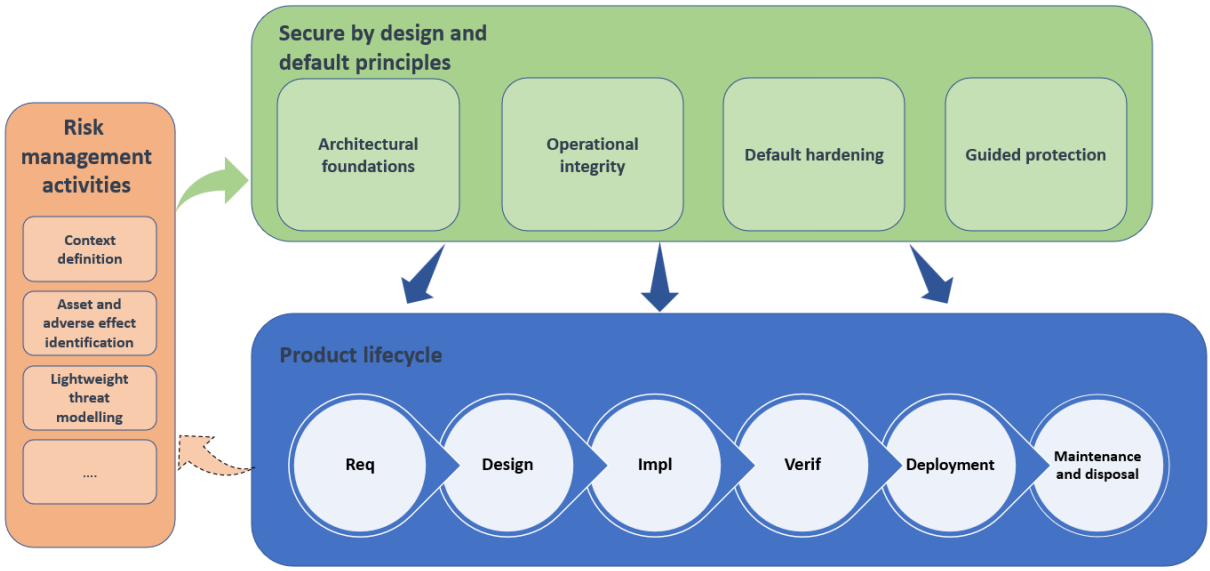

Figure 1: Secure by design and default across a product's life cycle

Figure 1 shows the relationships between the product life cycle, 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 processes of the product life cycle, from requirements through to end of life.

Risk management activities should be revisited in response to changes or events arising during the product life cycle. Similarly, while the product life-cycle processes are presented sequentially, both the life cycle and the application of secure by design and default principles across the processes are inherently iterative. Events or findings in later processes, such as testing results, deployment in new environments, incidents, discovered vulnerabilities or the implementation of new features, often require re-entry into earlier life-cycle processes.

Frameworks such as the OWASP software assurance maturity model (SAMM) ([10]) may be used to organise and mature security activities across the software life cycle. Its threat assessment and security requirements practices provide one example of how identified threats can inform testable

([10]) https://owasp.org/www-project-samm/.

security requirements. For SMEs, especially those operating with rapid iterations, the key is to keep life-cycle processes lightweight, risk driven and automation first:

- use small, reusable artefacts (one-page context, simple diagrams, checklists);
- prefer automated controls in continuous integration and continuous delivery (CI/CD) over manual reviews, reserving deep reviews for high-risk changes;
- introduce fast security gates aligned with existing agile ceremonies (definition of ready/done, pull request (PR) checks, release checklist).

Table 1: Cybersecurity activities during the product life cycle for SMEs

Life-cycle process	Actions	Deliverables
Deciding on requirements	Define the product context (users, environments, data), non-negotiable security defaults, and top risks and threat scenarios; establish clear criteria for addressing risks, based on the product's intended purpose, expected use and security impact.	One-page **security context and assumptions**, short **security requirements checklist** (including secure defaults)
Design	Maintain one architecture diagram **with** trust boundaries**;** run a lightweight threat model to identify a manageable number of threat scenarios (e.g. the top 5 to 10); decide on the critical design controls (authentication/authorisation ([a]), update mechanism, secrets, logging).	**Architecture and trust-boundary diagram**, **top threats and mitigations** (bulleted list)
Implementation	Build secure defaults into code/config; enforce dependency hygiene; protect secrets; require review for security-sensitive changes; configure automated static application security testing (SAST) / dependency scanning as part of CI environments (agile /DevOps/DevSecOps).	CI evidence (pipeline logs), lightweight **secure coding / checklist** (often a repo file)
Verification	Run automated security checks (SAST / dependency scanning, basic dynamic application security testing (DAST) where relevant); ensure default configuration is validated; run targeted penetration testing when potential risk triggers are hit (e.g. in the case of a substantial modification). Run automated tests with high coverage that include security-relevant negative tests (verifying that what shouldn't happen doesn't happen). Note that security verification should be integrated into developer workflows and CI pipelines as early as possible (following the 'shift left' principle).	**Release security checklist** (pass/fail and exceptions) and documented **known issues / residual risk**
Deployment	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 and disposal	Define a patch intake process and service-level agreements (SLAs), decide on vulnerability monitoring and incident handling processes and draw up an end-of-support/end-of-life plan; ensure secure disposal (data erasure, credential revocation).	**Vulnerability and patch process guide** (one page), **end-of-life/disposal note**, maintained **risk register** updates

([a]) 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 life cycle. 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.

In this report, 'risk management' concerns cybersecurity risks affecting the product and the users, operators, customers, data, services and systems that depend on it, taking into account the product's intended purpose and reasonably foreseeable use. It does not refer to the manufacturer's commercial or reputational risk.

These activities establish the context within which secure by design and default principles are applied. Their outputs directly informs:

- security requirements and architecture decisions;
- default configuration 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;
- the ongoing operational security posture, including patching and end-of-life handling.

Risk management is typically initiated early, but it must be iterative: revisited at defined life-cycle 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.

Risk assessments should also be informed by evidence from deployed products, including customer and researcher reports, incident investigations, observed exploitation patterns and proportionate telemetry, where available and appropriate.

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 sample set of activities that can drive secure by design and default decisions without creating heavy processes. The purpose of these activities is to translate product context and credible threats into testable security requirements, prioritise the relevant playbooks and determine the appropriate depth of controls and assurance activities, rather than applying every practice uniformly.

Table 2: Risk management activities for SMEs

Activity	*Actions*	*Deliverables*
Product context and scope	Define intended use, deployment environments, user/admin roles, data types/sensitivity and key external dependencies (cloud, mobile app, third parties).	One- or two-page 'product security context' note (scope, assumptions, dependencies)
Asset and adverse effect identification	List top data, hardware or function assets (e.g. credentials, customer data, essential functions) and key adverse effect outcomes (privacy breach, takeover, outage, fraud, safety impact).	Asset list and top adverse effects list (one page)
Lightweight threat modelling	See Section 2.3	See Section 2.3
Risk register	Record 10 to 30 risks using a simple, documented prioritisation method, with owner, treatment and status; link priority risks to backlog items/controls. The prioritisation method could take into account aspects like impact, likelihood, plausibility, exploitability or exposure, as appropriate.	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 and exploitation of default credentials are not acceptable) and establish criteria for accepting residual risk. For example, the accepted level of residual risk should not undermine the essential cybersecurity requirements, given the product's intended purpose and reasonably foreseeable use.	One-page risk acceptance and exceptions policy
Security requirements baseline	Translate top risks into testable 'must' requirements (authn/authz, secure defaults, secrets, encryption, logging, updates).	security requirements checklist (testable controls)
Release risk review gate	As part of the secure development life cycle, include a formal pre-release gate to verify compliance with the requirements defined during the above activities. This review should confirm checklist met, defaults verified, known vulnerabilities triaged, high risks treated / accepted with rationale; decide go/no-go.	Release security review record (ticket/comment) and documented exceptions
Change-triggered reassessment	Rerun context/threat/risk steps when major changes occur (architecture, auth model, data, critical dependencies/suppliers, deployment environment), including changes that could be considered substantial modifications, or after incidents.	Updated context note, threat shortlist and risk register entries (with date)

Risk assessments can also support product-specific implementation of applicable cybersecurity requirements, assessment and treatment of identified vulnerabilities and evaluation of the security impact of significant changes to a product.

2.3 Threat modelling

Threat modelling is a structured activity that supports security decision-making and may inform risk assessment by helping teams identify credible threats and attack paths. Combined with the product context and an appropriate prioritisation method, the outputs can help in selecting appropriate security requirements and controls, including secure by default settings.

One commonly used approach to identifying and categorising security threats is STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege) ([11]). STRIDE is presented in this report as an illustrative example, but other suitable threat-modelling approaches may also be used([12]).

Threat modelling should align with widely accepted threat-modelling principles ([13]), 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 and failing to review the model following substantial product modifications or changes in the threat landscape. For products incorporating artificial intelligence (AI) / machine learning (ML) components, threat identification should also consider relevant AI-specific attack patterns, such as prompt injection, model poisoning and adversarial inputs ([14]).

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).

A lightweight threat-modelling exercise can be organised around four questions (Shostack's four-question framework) ([15]).

- What are we working on?
- What can go wrong?
- What are we going to do about it?
- Did we do a good enough job?

Table 3 provides an illustrative example of how these questions can be translated into practical activities and deliverables for SMEs.

([11]) 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.
([12]) https://github.com/arnepadmos/threats.
([13]) https://www.threatmodelingmanifesto.org/.
([14]) https://owasp.org/www-project-top-10-for-large-language-model-applications/.
([15]) https://owasp.org/www-project-threat-modeling/.

Table 3: Threat modelling for SMEs

ACTIVITY	ACTIONS	DELIVERABLE
Define scope, assumptions and security objectives	Time-box a short scoping stage to make the exercise decision focused. Capture what is in and out of scope (e.g. devices, apps, cloud services, admin tools), the deployment context(s) and the assumptions and constraints you are working under (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 (e.g. confidentiality, integrity, availability, plus privacy/safety if applicable).	One-page (or equivalent ticket/wiki) 'Threat model scope and objectives' note, covering: ▪ purpose (what decisions this will drive); ▪ scope boundaries (components and environments); ▪ assumptions/constraints; ▪ security objectives and 'crown jewels' (what must not fail).
Model the system at a useful level of abstraction	Produce a simple representation of the system that answers Shostack's core question, **'What are we working on?'** This may be an architecture diagram, a data-flow diagram or another suitable representation. It should show the main components, data stores, external entities, key data flows, entry points and trust boundaries in sufficient detail to assess exposure. Complementary techniques, such as misuse cases or attack trees, may also be used where helpful. Consider reusing the architecture and trust-boundary diagram developed during the design phase described in Table 1.	System representation, covering: ▪ main components (devices, apps, cloud services, APIs); ▪ external entities (users, admins, third-party services); ▪ data stores (device storage, cloud database, logs); ▪ entry points (APIs, admin UI, update channel, local ports); ▪ trust boundaries (where trust assumptions, privileges or security properties change).
Mark trust boundaries and privilege paths; identify key assets	Annotate the diagram with (a) trust boundaries ([a]) (boundaries between environments with different security properties) and (b) the highest-privilege operations (e.g. firmware / OTA updates, remote admin, key provisioning, identity issuance). This is the step that turns architecture into security-relevant architecture.	Diagram showing: ▪ trust boundaries (internet–back end, 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	Generate a short list/representation of the answers to the question **'What can go wrong?'**, mapped to entry/exit points, data flows and trust boundaries (e.g. 'credential stuffing → account takeover → remote control', 'malicious update', 'API authorisation bypass', 'man-in-the-middle attacks during onboarding'). Prioritise them using a lightweight, documented method appropriate to the product. This might take into account impact, likelihood, plausibility, exploitability and exposure or use predefined severity criteria. OWASP ([b]) describes this as threat identification and ranking as a core step in most threat-modelling approaches.	Top threats table, showing: • a manageable number of threat scenarios (e.g. the top 5 to 10) tied to specific entry/exit points and trust boundaries; • priority or severity rating with a short rationale. Prioritised threat scenarios that drive design and default choices
Define mitigations and secure defaults, verify their effectiveness and set refresh triggers	Next, focus on **'What are we going to do about it?'** For each top threat, specify the mitigation strategy, the required control(s) and the secure by default settings that the product should ship with (e.g. 'admin interface disabled by default', 'no default passwords', 'signed updates enforced', 'least-privilege roles', 'authentication attempts rate-limited'). Then map each control to how it will be verified (e.g. CI checks, tests, configuration validation, release gates), which will help to answer the question **'Did we do a good enough job?'** Finally, define the triggers that require rerunning the model (e.g. a new internet-exposed interface, a new auth model, new sensitive data, a new critical dependency, a major architecture change).	Controls, defaults and verification checklist, including: ▪ Control/default decision for each top threat; ▪ test/verification mapping (automation first); ▪ explicit refresh triggers (so that the model stays current).

([a]) https://owasp.org/www-community/Threat_Modeling_Process;.
([b]) https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html.

A practical starting point could be to identify a small number of the highest-impact adverse-effect scenarios for users, operators or other affected parties and a manageable number of credible threat scenarios (e.g. the top 5 to 10) that could lead to them. These scenarios should be prioritised based on impact and likelihood. As part of their release review, teams should determine whether product changes affect the scenarios, assumptions or controls. 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.

Lightweight threat modelling does not necessarily require special threat-modelling expertise. Product expertise, on the other hand, is essential, because an accurate understanding of the product, its intended use and its deployment context is the foundation of a useful threat model. However, specialist security expertise is likely to be needed for higher-risk products or more complex threats.

Threat-modelling outputs can vary depending on aspects like the product's scope, the assumptions, the methodology selected and analyst judgement. These inputs should be documented and applied consistently, to increase repeatability and support review.

Threat modelling and risk assessment should be revisited throughout the product lifecycle. For example, reviews could be performed at initial product design, implementation of the technical architecture, and verification, validation, maintenance and operation, where discovered vulnerabilities and field evidence can inform subsequent assessments.

SECTION 3

Secure by design and default 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 hardened from the start and that remain so once released into real-world environments. ENISA's report *Good Practices for Security of IoT (SDLC)* ([16]) describes secure by design as a 'holistic approach' that must be applied throughout the entire life cycle 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 during the earliest stages of design to minimise the impact of cyberattacks.

- **Secure 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.
- **Secure by default** ensures that 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 Secure by design principles

Secure 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. For the purposes of this playbook, secure by design is treated as a life-cycle approach. The principles are grouped into architectural foundations (how a system is built) and operational integrity (how it is managed and maintained) (Figure 2). These categories are practical organising lenses rather than mutually exclusive classifications, and some aspects of the principles contribute to both categories ([17]).

([16]) 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.

([17]) 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 (https://owaspsamm.org/model/).

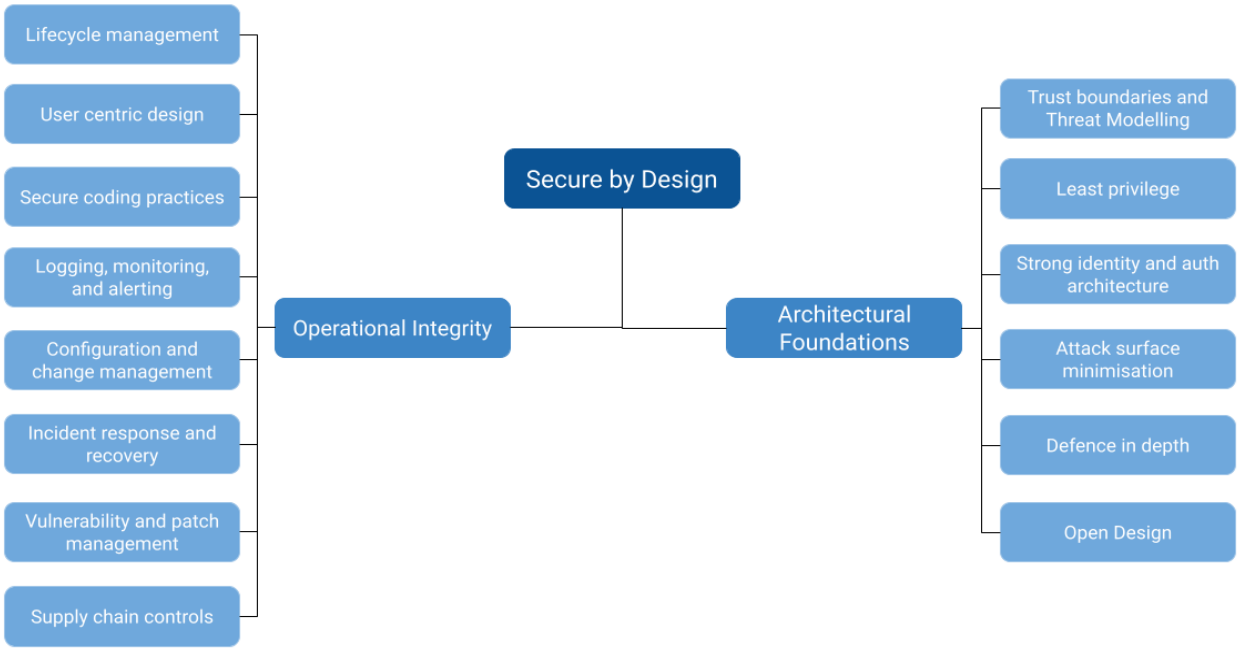

Figure 2: Secure by design principles

3.1.1 Architectural foundations

Architectural foundations are the blueprints for a system's security; they 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.** These concepts reflect two related security principles: trust should be made explicit rather than assumed, and threats should be identified before and throughout development. 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 (e.g. validation, rate limiting, encryption or isolation) are required. Threat modelling also provides a structured way to identify what could go wrong at these 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 be elevated only 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 (e.g. 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 attacks 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 non-essential 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. multi-factor authentication (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; rather, it means that the security of the product 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 practices that maintain product security throughout development, deployment, operation and 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.

- **Life-cycle 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 life-cycle 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 for 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 set-up that enables encryption automatically reduces the chance of misconfiguration and encourages the use of the security feature.
- **Secure coding and verification 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 SAST tools ([18]) while coding to identify vulnerabilities and software composition analysis to detect vulnerable third-party libraries. They can then apply dynamic testing tools ([19]) (DAST) before deployment to uncover runtime issues. These practices ensure that code-related vulnerabilities are identified early and 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 that 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 and unusual outbound connections.
- **Configuration and change management.** Secure operation requires configurations to be controlled, consistent and auditable. Baseline hardening standards should be defined (e.g. 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 (e.g. 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 manufacturers 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 modification rights should be limited to named individuals, protected with MFA and reviewed through lightweight peer approval for changes to security-critical areas. Software bills of materials (SBOMs) should be generated and maintained to support dependency transparency, vulnerability management and product life-cycle security.

3.2 Secure by default principles

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

The goal of secure by default is to minimise the attack surface after deployment by ensuring that the most secure configuration is applied automatically ([20]). This includes disabling unnecessary services, applying restrictive access controls and providing clear information to users about the default security

([18]) https://owasp.org/www-community/Source_Code_Analysis_Tools.
([19]) https://owasp.org/www-community/Vulnerability_Scanning_Tools.
([20]) 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).

posture. Secure defaults reduce reliance on user expertise and limit the potential for misconfigurations, which are a common source of security incidents.

To summarise, secure by design focuses on how the system is engineered, while secure by default focuses on how the system arrives and behaves when the user first turns it on. We can organise secure by default into two distinct categories: **default hardening** and **guided protection** (Figure 3). These categories are practical organising lenses. Some principles contribute both to the product's initial secure state and to maintaining that state during use.

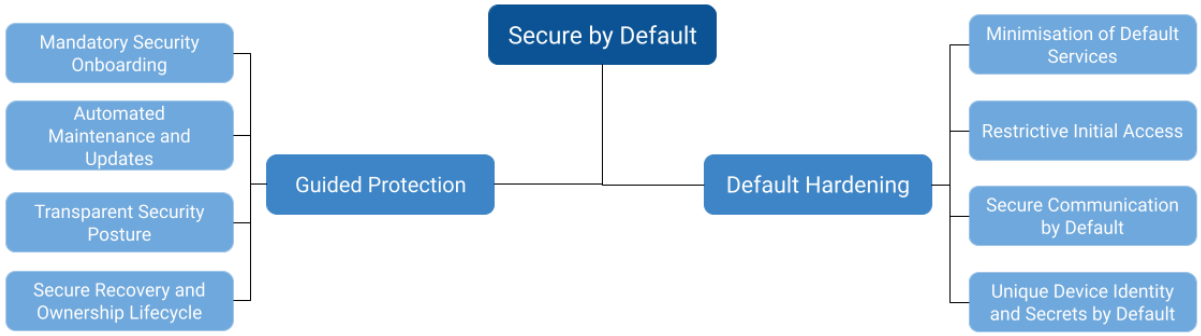

Figure 3: Secure by default principles

3.2.1 Default hardening

Default hardening 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 will not 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 could 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 set-up steps and clear feedback to ensure that 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 set-up wizard. For instance, requiring a user to configure 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 life cycle.** 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 principles 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 intention 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.** 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 that the checklist has been implemented.
- **Release gate.** A copy and paste set of pass or fail criteria that can be used in a release review (or CI/CD). Teams should apply the criteria relevant to the release and confirm that the controls continue to operate as intended where the release could affect them.

This structure is deliberately aligned with how SMEs operate: short cycles, shared responsibilities, limited specialist capacity and a need for guidance with a high signal-to-noise ratio. Unless stated otherwise, the actions in the playbooks are directed at manufacturers and teams acting on their behalf. References to users and operators describe product capabilities, information or actions that manufacturers should enable or support.

Developers and manufacturers 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.
- Use the product changes, threat model and risk assessment to determine which release-gate criteria require verification. Unchanged controls may be recorded as not affected.
- Reuse the same evidence across playbooks where appropriate. For example, a single SBOM, scan result or release record may satisfy related evidence requirements and release-gate criteria for more than one playbook.
- Allow exceptions only with a documented rationale, owner and review date.
- Refresh playbooks periodically based on incident learnings, vulnerability trends and product changes.

It is important to note that the items listed under 'Minimum evidence' do not necessarily require separate documents or artefacts. A single item of evidence may support several playbooks. Organisations should reuse existing engineering records wherever possible, including architecture diagrams, repository files, configuration records, issue-tracking tickets, test results, SBOMs, CI/CD outputs and release records.

The contents of 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.

The playbooks can be adopted progressively, prioritising those most relevant to the product's risks, intended use and development context. Progressive adoption does not imply delaying any applicable requirements or controls, including those arising from legal obligations under the CRA. A suggested approach is provided at the end of this section.

4.1 Trust boundaries and threat modelling

Principle (secure by design)	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 that the system's architecture and data flows are understood, trust assumptions are explicit and the highest-risk attack paths are identified early, so that security controls and secure defaults are designed in, not retrofitted.

Checklist

Draw the system (one diagram)	▪ Include users/admins, device/app, back-end 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–back end, tenant–tenant, device–cloud, user–admin). ▪ Highlight high-privilege flows (auth, key management, updates, admin actions).
List critical assets	▪ For instance, list credentials/keys, customer / personally identifiable information, payment tokens, device control functions, availability, audit logs, data, source code and the IDE.
Identify and prioritise top threats	▪ Map threats to the relevant entry/exit points, data flows, assets and trust boundaries, for example account takeover, malicious updates, API authorisation bypass, man-in-the-middle attacks and the tampering with, replay of or injection of fabricated or stale data and commands. ▪ Assign a high, medium or low priority using a documented method appropriate to the product, such as assessing impact, likelihood, plausibility, exploitability and exposure.
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

- One architecture/data-flow diagram with trust boundaries and entry points (stored in repo/wiki)
- Top threats list (e.g. the top 5 to 10 threat scenarios) with H/M/L priority and owners
- Control/default 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 threat scenarios reviewed; high risks have mitigations or documented exceptions
- ✓ 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 the following are relevant: new API/interface, auth model change, new sensitive data, major dependency/supplier change, OTA/update changes, major architecture change.

4.2 Least privilege

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

Checklist	
Define minimum viable permissions for each 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 at each release), flag broad privileges and unused permissions/tokens; remove or time-limit exceptions.

Minimum evidence

- **Access model exists:** for example, 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 and expiry.

Release gate

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

4.3 Strong identity and authentication architecture

Principle (secure by design)	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 (e.g. MFA, device binding, or cryptographic credentials).
Manage sessions and credentials securely	▪ Ensure that 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 exists showing the identity source for users, devices, services and administrators.
- **No shared identities:** identity inventory shows unique identities, no shared production accounts or credentials.
- **Authentication enforced:** configuration or test evidence shows that all interfaces enforce authentication or are explicitly defined as public with appropriate protections applied. Network location or reachability alone is not treated as authentication, unless explicitly justified by the product context and risk assessment and supported by appropriate compensating controls.
- **Strong auth for privileged access:** policy or configuration shows that MFA or equivalent is 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 (secure by design)	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.
Minimise data collection, processing and retention	▪ Collect, retain and store only the personal or sensitive data necessary for the product's intended purpose and securely delete it when no longer required.

Minimum evidence

- **Exposure inventory exists:** a maintained list of externally reachable interfaces (APIs, ports, protocols, admin interfaces, update channels) is stored in the repo or product documentation.
- **Default deny enforced:** network rules, gateway configuration or device settings show that only explicitly required interfaces are enabled in production.
- **Minimal build verified:** build configuration, manifest, package list or firmware contents show that only required components are included.
- **No dev or diagnostic tooling shipped that might enable exploitation:** inspection of production artefacts confirms absence (or verification that they cannot impact security) of shells, debuggers, compilers, test interfaces and diagnostic utilities.
- **No legacy exposure:** release notes, ticket or checklist entry confirms that exposed interfaces and dependencies were reviewed and any unused or legacy elements removed.
- **Data inventory and retention record:** data inventory with purposes, retention periods and deletion methods is kept.

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 tools present in production unless there is evidence that it cannot impact security
- ✓ Attack surface review completed for this release; deprecated or legacy interfaces updates/removed
- ✓ Unnecessary data processing has been removed and required deletion mechanisms verified

4.5 Defence in depth

Principle (secure by design)	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 that more than one independent control protects each asset. ▪ Where critical data or commands cross trust boundaries, consider integrity and freshness controls that remain effective if another protection layer, such as transport security, is bypassed.
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 that 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 that response mechanisms exist to contain or isolate compromised components when alerts are triggered.

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.
- **Multilayer 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 **(secure by design)**	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 that the product's security does not depend on the 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 the 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/authorisation requirements, error handling, rate limits and supported configurations. ▪ Provide a secure configuration guide (outlining what must be enabled and 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 (to 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 file, repo policy or checklist evidence)

Release gate

- ✓ Security design note updated for any architectural/security-relevant change (auth, crypto, update path, trust boundaries)
- ✓ No custom/proprietary crypto or undocumented security-critical protocols introduced
- ✓ API/security documentation 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 Life-cycle management

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

Checklist	
Define support commitments and life-cycle states	▪ Publish (internally or externally) life-cycle 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 48 hours; fixes released within X days where feasible).
Make the product updatable and recoverable	▪ Implement a secure and transactional update mechanism (authenticated updates, integrity protected, rollback/recovery plan). ▪ Ensure safe failure modes (do not 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 and customer reports. ▪ Maintain an SBOM (at least for each release) and a dependency update cadence. ▪ Triage, prioritise and document risk decisions (fix, mitigate, accept with expiry date).
Monitor, log and handle incidents	▪ Enable security-relevant logging (auth events, admin actions, update events, boundary-crossing requests). ▪ Define minimum incident response steps, for instance detection, triage, containment, remediation, customer communication
Learn from deployed products	▪ Collect evidence on how products are deployed and used, such as customer feedback, support cases, vulnerability reports, incident findings and privacy-preserving telemetry where appropriate. ▪ Review findings, including the results of root-cause analyses where applicable, and update the product context, risk assessment, threat model and secure defaults as necessary.
Plan secure decommissioning and disposal	▪ Provide data erasure mechanisms and guidance (factory reset, key revocation, account deprovisioning). ▪ Revoke credentials/keys when devices/users/services are deactivated. ▪ Ensure that end-of-life guidance includes what security updates cease and what customers must do.

Minimum evidence

- **Life-cycle policy**: lifecycle states + support window + patch SLAs (1 page).
- **Update/release process**: secure update approach and rollback/recovery notes
- **Vulnerability management log:** intake, triage, prioritisation, disposition, owner, dates (ticket board or spreadsheet)
- **SBOM:** SBOM generated in a machine-readable, widely adopted standard format (e.g. SPDX or CycloneDX) and stored with the release
- **Monitoring and logging:** security logging baseline + sample logs showing admin/auth/update events
- **Field feedback record:** summary of relevant operational evidence reviewed, deviations from deployment assumptions identified and resulting actions
- **Decommissioning checklist:** data wipe/reset, credential/key revocation and customer guidance

Release gate

✓ Support status and life-cycle 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 dependencies); decisions recorded (fix/mitigate/accept with expiry)
- ✓ SBOM generated (or dependency inventory updated) and stored for the release
- ✓ Security logging verified for critical events (authentication, admin actions, updates)
- ✓ Decommissioning actions defined for components introduced/changed (e.g. reset, wipe, key revocation)
- ✓ Any accepted residual security risk has an owner and review/expiry date

4.8 User-centric design

Principle **(secure by design)**	Security mechanisms must be usable and understandable even for everyday users.
Objective	Ensure that security controls and secure defaults are usable, understandable and aligned with real user workflows, so that 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 communications, logging enabled). ▪ Avoid optional security for critical protections; make insecure modes explicit and gated.
Make set-up 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 error messages that explain what happened, the 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 (e.g. admin, operator, viewer). ▪ Support safer auth options where feasible (e.g. 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 and disabled, with rationale)
- Onboarding / security set-up guide (short, step-by-step, including credentials and update steps)
- Roles/permissions model (roles defined + what each can do)
- User-facing security message catalogue (examples of warnings / error messages and text outlining required remediation measures)
- Usability validation notes (at least three to five 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 set-up
- ✓ 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 / error messages are actionable (what, why, how to fix) and documented
- ✓ Basic usability check completed for key security flows (onboarding, admin tasks, recovery, updates); issues tracked/fixed or accepted with owner + date

4.9 Secure coding and verification practices

Principle (secure by design)	Developers should follow established secure coding standards to prevent common vulnerabilities.
Objective	Reduce common, high-impact vulnerabilities by eliminating relevant weakness classes through architectural and technology choices, where feasible, and by standardising how remaining risks are addressed in code, review and testing.

Checklist	
Eliminate vulnerability classes by construction	▪ Prefer architectural and technology choices that prevent relevant vulnerability classes from arising, such as memory-safe languages for buffer overflows, parameterised data-access APIs for SQL injection and frameworks with context-aware output encoding for cross-site scripting. ▪ Where elimination is not feasible, identify the remaining vulnerability classes and apply appropriate coding, review, testing and containment controls.
Adopt a secure coding baseline (language/framework specific)	▪ Define must-follow rules for your stack (e.g. on 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, allow lists, 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 for each 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 or deserialisation and for any new external exposure. ▪ Use PR checklists and CODEOWNERS files for sensitive areas. ▪ Apply the same review requirements to AI-generated or AI-modified code as to human-written code.
Automate detection in CI/CD	▪ Run SAST and 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. ▪ AI-generated or AI-modified code should go through the same SAST, dependency and secrets checks as human-written code.

Minimum evidence

- Design evidence: record of relevant vulnerability classes considered and the architectural or technology choices used to eliminate them by construction
- Secure coding standards (one or two pages or a repo file) for your stack and a banned patterns list
- PR checklist / CODEOWNERS file for security-sensitive modules
- CI pipeline evidence: SAST and dependency scan results for the release
- Secrets controls: proof that secrets are not in repo (pre-commit secret scanning results) and secrets manager usage
- Test evidence: at least a small suite of negative tests for critical endpoints/inputs

Release gate

✓ Relevant vulnerability classes have been eliminated by construction where feasible; where not feasible, appropriate preventive and verification controls are in place.
✓ A secure coding baseline exists for the stack and is referenced in the repo

- ✓ SAST and dependency scanning run in CI; critical issues fixed or covered by a documented exception (with owner and expiry)
- ✓ Secret scanning enabled; no secrets committed; rotation performed if exposure occurred
- ✓ Security-sensitive changes were peer reviewed (auth/authz, crypto, parsers, external interfaces)
- ✓ Negative tests run on critical endpoints (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 (secure by design)	Systems should generate appropriate security-relevant logs, retain them for a defined period and protect them from tampering, so that 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. ▪ Authorisation: access denied, privilege or role changes, admin actions. ▪ Boundary events: external API calls, device onboarding or pairing, OTA/update events. ▪ Data or security changes: configuration changes, key or secret changes, user or service enabled or disabled. ▪ Do not log personal or secret data by default. In general, log only the bare minimum required to understand events in default log level
Standardise access and audit log structure and attribution	▪ For access and authorisation logging, use consistent fields capturing who (actor), what (action or event type) and when (timestamp), with additional contextual fields as appropriate. ▪ Ensure that admin actions are attributable to a named identity (no shared accounts).
Centralise 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 apply write-only permissions where feasible. ▪ Set retention targets (e.g. 30 days or 90 days, longer for archival material 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 duties / ownership for security alerts. ▪ Document a minimal triage playbook covering: validate, scope, contain, remediate and communicate. ▪ Carry out a quarterly tabletop exercise or dry run for one common scenario (account takeover or API key leak).

Minimum evidence

- Logging baseline: list of must-log events and required fields (one page)
- Central logging proof**:** screenshot/export/config showing logs from key components arriving centrally
- Retention setting: configured retention is in place and there is an access-control policy for logs
- Alert catalogue: list of enabled alerts with thresholds and owners (initial set can consist of around 5 to 10 alerts)
- Triage runbook: one-page steps and escalation contacts, plus 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 events, new admin / role change, key/secret change, update failures)
- ✓ Triage runbook exists and has been tested recently (tabletop exercise or test alert)
- ✓ Logging/alerting exceptions are documented, with owner and review/expiry date

4.11 Configuration and change management

Principle (secure by design)	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 that 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 and 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/authorisation, crypto, update mechanisms, data classification or third-party integrations. ▪ Ensure that a lightweight threat/risk review is triggered when these tags apply.

Minimum evidence

- **Config versioned:** repo location(s) for IaC, config, policies; PR review history
- **Baseline config:** documented secure defaults and key settings (one page)
- **Drift detection:** a scheduled job/report or tool output for critical config items
- **Deployment controls:** proof that 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) logged and post-reviewed, with owner and expiry date

4.12 Incident response and recovery

Principle (secure by design)	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 escalation thresholds, when senior management should be informed and who can approve customer communications, 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	▪ Make sure that it is possible to revoke/rotate credentials and keys quickly. ▪ Make sure that it is possible to disable accounts, block IPs, quarantine services/devices and roll back releases / config changes. ▪ Ensure that 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 exercise (30–60 minutes) quarterly on one realistic scenario. ▪ Record and review incidents, near misses and exercise findings. Track actions to closure and update risk assessments, threat models, runbooks, detections and controls as appropriate.

Minimum evidence

- Defined roles: incident-response contact list + roles (owner, backups and escalation paths established).
- Runbooks: one-page incident runbook and 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 and one recent restore test result (or evidence of a successful restore).
- Post-incident, near-miss or tabletop notes showing findings, actions and resulting improvements.

Release gate

- ✓ IR roles and escalation contacts are current
- ✓ Incident runbook exists and covers detect/triage/contain/recover with at least one common scenario
- ✓ Containment actions are ready
- ✓ Backups enabled for critical data/config; restore procedure documented and tested recently
- ✓ Post-incident, near-miss or tabletop notes exist and are updated

4.13 Vulnerability and patch management

Principle (secure by design)	Manufacturers should establish practical, repeatable vulnerability and patch management processes and prioritise remediation according to 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 (do not miss issues)	▪ Sources: dependency scanning, SAST/DAST results, advisories, customer reports, security emails, etc. ▪ Assign a single owner for triage and tracking.
Triage and prioritise consistently	▪ Use a lightweight severity approach (e.g. critical, high, medium or low) plus ‘internet-exposed’ and ‘known exploited’ flags. ▪ Ensure awareness of applicable incident and vulnerability reporting timelines. ▪ 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 or monthly) for dependency updates. ▪ Pin versions, remove unused dependencies and track transitive dependencies.
Fix, test and release with a secure process	▪ Ensure that fixes are reviewed and tested; verify no regressions in authn/authz, input validation or critical workflows. ▪ For devices/IoT: ensure secure OTA/update path and safe rollback where feasible.
Communicate, learn 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. ▪ Establish an escalation process to assess applicable regulatory reporting obligations. ▪ Where proportionate, publish machine-readable advisories (e.g. CSAF), including structured information on whether specific products or versions are affected, such as through VEX. ▪ Analyse significant or recurring vulnerabilities, field reports, incidents and exploitation patterns to identify root causes. Feed the findings back into the risk assessment, security requirements and development practices.

Minimum evidence

- **Vulnerability tracking board / register:** issue, severity, affected components/versions, owner, status, target date
- **Defined SLAs:** for example, critical triage ≤ XX hours; remediation/release target ≤ X days (specify numbers that are realistic for you)
- **Scanning evidence:** CI outputs for dependency scanning and SAST (and DAST if applicable)
- **Proactive dependency patches:** SBOM or dependency inventory for each release (at minimum for shipped artefacts)
- **Patch release record:** link from vulnerability ticket → PR(s) → tests → release version → rollout confirmation
- **Exception log:** accepted risks, with owner and expiry/review date, and compensating controls (if any)
- **Field feedback and improvement record:** recurring security issues or attack patterns, root causes identified where applicable, and the resulting risk, design or backlog actions.

Release gate

- ✓ Dependency and SAST scans executed for the release; critical and high findings addressed or documented exception (with owner and expiry date)
- ✓ SBOM (or dependency inventory) generated/updated and stored for the release

- ✓ Known vulnerabilities affecting shipped components are triaged with severity information, 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- and high-severity issues 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 (secure by design)	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, including externally sourced AI/ML models where relevant. ▪ 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. Pinning and lock files improve build repeatability but do not establish that the pinned component is trustworthy. New and updated dependencies still require appropriate review and security checks. ▪ Run dependency scanning in CI; block builds on critical- or high where feasible (or require explicit exception). ▪ Where proportionate, obtain dependencies through approved repositories or controlled registries and apply integrity, malware and policy checks before use. ▪ For externally sourced AI/ML models, record their source and version, verify their integrity and provenance, and assess relevant security risks before integration.
Harden the build and release pipeline (integrity of artefacts)	▪ Restrict CI/CD permissions (least privilege), protect secrets and separate build and deploy roles. ▪ Sign release artefacts and, where proportionate, generate verifiable provenance describing how and from which inputs they were built. ▪ For higher-risk products, use isolated and ephemeral build environments, consider restricting build-time network access to approved sources and consider hermetic builds where feasible.
Set minimum supplier expectations (lightweight due diligence)	▪ For critical suppliers, require a security contact / VDP, patch timelines and confirmation of secure development practices. ▪ Use a short questionnaire aligned with 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 that contracts and SLAs cover vulnerability notification and support windows (where possible).

Minimum evidence

- Supplier + component inventory (top vendors, critical libraries/tools, hosting/build services), including externally sourced AI/ML models where relevant
- SBOM (or dependency inventory) per release, stored with the release artefacts
- CI results showing dependency scanning (and ideally secret scanning) executed on PRs/releases
- Release integrity evidence: artefact signing and restricted CI/CD access (configs/screenshots/logs)
- Supplier baseline checks for critical suppliers (a completed questionnaire or SCVS-aligned checklist)
- Exception log for accepted supply-chain risks (with owners and expiry dates)

Release gate

- ✓ SBOM (or dependency inventory) generated and stored for this release

- ✓ Dependency scanning executed; critical- or high-severity issues fixed or covered by a documented exception (with owner and expiry date)
- ✓ New dependencies/suppliers reviewed and approved (recorded in PR/ticket)
- ✓ CI/CD and build secrets are protected; build/deploy permissions are least privilege and attributable
- ✓ Release artefacts 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 and review/expiry date

4.15 Minimisation of default services

Principle **(secure by default)**	Disable non-essential features and services by default. Only the functionality required for the core operation of the product should be enabled by default.
Objective	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 that optional features and services are disabled by default.
Require explicit opt-in for additional functionality	▪ Ensure that non-essential features and services can be enabled only 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 show that users are informed of security implications when enabling non-essential features and services.
- **Defaults reviewed for this release:** release notes, checklist or ticket confirms that default settings have been 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 services are enabled by default without review
- ✓ Default configuration has been reviewed and confirmed for this release

4.16 Restrictive initial access

Principle (secure by default)	Ship systems in the most restrictive access state possible. Eliminate shared or default credentials and require a secure access set-up before any privileged or sensitive operations are allowed.
Objective	To prevent immediate compromise after deployment by ensuring that 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 for each device or instance	▪ Ensure that each device or instance has unique credentials generated at manufacture, provisioning or first boot.
Enforce a secure initial set-up	▪ 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 unnecessarily exposed.

Minimum evidence

- **No default credentials shipped:** build configuration or documentation confirms that no shared or hard-coded credentials exist.
- **Unique credentials enforced:** provisioning records, configuration or device inventory shows per-device or per-instance credentials.
- **Secure set-up required:** configuration or test evidence shows credential change or secure set-up 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 that onboarding and recovery interfaces are protected.

Release gate

- ✓ No shared or default credentials present in production builds
- ✓ Unique credentials generated for each device or instance
- ✓ Secure set-up 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 (secure by default)	Enforce secure communication from the start. All external communications must be encrypted and authenticated by default, with no support for insecure or plain-text protocols for convenience.
Objective	Protect data in transit and prevent interception, tampering or impersonation by ensuring that communications are always secure, without relying on user configuration or later hardening.

Checklist

Protect external communications by default	▪ Require encrypted communication for all external interfaces from first use. ▪ Where data or commands pass through intermediaries, assess whether transport protection alone is sufficient or whether additional end-to-end authenticity, integrity and freshness controls are required.
Disable insecure and plain-text 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 that 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 that communication failures result in denied connections rather than downgrading to insecure protocols or configurations.
Plan for cryptographic change	▪ Design cryptographic mechanisms so that algorithms, keys, certificates and trust anchors can be replaced or migrated without substantially redesigning the product, where feasible. ▪ Assess whether the product's support lifetime, data confidentiality requirements and deployment constraints require preparation for migration to post-quantum cryptography.

Minimum evidence

- **Secure protocols enforced by default:** configuration or test evidence shows that encrypted and authenticated communication is required from first connection.
- **Insecure protocols disabled:** configuration confirms that plain-text or legacy protocols are not enabled or available.
- **Endpoint authentication in place:** configuration or test evidence shows that mutual or appropriate endpoint authentication takes place for external communications.
- **Strong protocol configuration applied:** configuration shows that only approved protocol versions and cryptographic settings are enabled.
- **No insecure fallback:** test evidence shows that connection attempts fail securely rather than downgrading security.
- **Cryptographic migration note:** identification of critical cryptographic mechanisms and how algorithms, keys, certificates and trust anchors are to be updated or replaced.

Release gate

- ✓ External communications are encrypted and authenticated from first connection
- ✓ Insecure or plain-text 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
- ✓ Cryptographic mechanisms can be replaced or migrated where required by the product's risk assessment and support lifetime.

4.18 Unique device identity and secrets by default

Principle **(secure by default)**	Ship each product instance with a unique cryptographic identity and secrets by default if it uses any private/secret value. Do not use shared credentials, keys or certificates across devices or installations.
Objective	Prevent large-scale compromise by ensuring that a breach of one device or a 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 (e.g. 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 (e.g. secure storage, hardware-backed protection or restricted access).
Use unique secrets for security-critical functions	▪ Ensure that security-critical functions like authentication, secure communications and update verification rely on per-device secrets, not shared material.
Support revocation and replacement	▪ Ensure that 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 (e.g. key pair or certificate) for each device or instance.
- **No shared secrets present:** build artefacts or configuration confirms that no shared credentials or hard-coded secrets exist.
- **Secrets protected at rest:** documentation or configuration shows that secrets are stored using appropriate protection mechanisms.
- **Security functions use unique secrets:** configuration or design notes show that authentication, communication and update processes rely on per-device secrets.
- **Revocation supported:** documentation or configuration shows that device identities or secrets can be revoked or replaced.

Release gate

- ✓ Unique cryptographic identity generated for each 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 (secure by default)	Require critical security controls to be configured during initial set-up. Do not allow products to enter an operational state until essential security steps are completed.
Objective	Ensure that 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 (e.g. MFA, encryption keys, admin account set-up).
Enforce security set-up during first use	▪ Integrate mandatory security steps into the initial set-up 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 and external connectivity until mandatory security steps are completed.
Provide clear, guided configuration	▪ Guide users through security set-up 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 set-up:** configuration, screenshots or test evidence shows that critical security steps cannot be skipped.
- **Operation blocked pre-onboarding:** configuration, screenshots or test evidence shows that the product cannot enter normal operation before onboarding is complete.
- **Guided set-up is provided:** screenshots, configuration flows or documentation shows that clear user guidance is provided on security set-up.
- **Onboarding is re-enforced on incomplete security configuration:** configuration, screenshots or test evidence shows that onboarding is re-triggered if required security settings are missing.

Release gate

- ✓ Mandatory security steps are defined and enforced during initial set-up
- ✓ 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 (secure by default)	Manufacturers should provide secure update mechanisms that minimise user and operational effort. Security updates should be enabled by default, while allowing authorised users or operators to approve or schedule installation where the deployment requires controlled updates.
Objective	Ensure that vulnerabilities are addressed quickly and the device is left vulnerable for as little time as possible.

Checklist

Enable managed security updates by default	▪ Enable security updates by default, ensuring that the product automatically checks for updates, informs the user and supports timely installation, with escalation where updates are repeatedly deferred. ▪ Where unattended installation could affect safety, availability or operational continuity, support staged rollout, maintenance windows and authorised operator approval.
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. ▪ Support staged rollout, maintenance windows, operator approval and rollback where required by safety, availability or operational-continuity considerations.
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 that security updates are enabled out of the box.
- **Security updates decoupled:** configuration or release notes show that security fixes can be delivered independently of optional feature/functional updates.
- **Update verification enforced:** configuration or design notes show that updates are authenticated and integrity-checked.
- **Safe update mechanism in place:** documentation or test evidence shows that updates can be applied without significantly disrupting basic operation.
- **Update strategy:** documentation or test evidence shows that the update process supports controlled approval, staging and rollback where required by the deployment context.
- **User notification supported:** logs, UI messages or documentation shows that users are informed of update activity or status.

Release gate

- ✓ Security update functionality is enabled by default, with an installation approach appropriate to the product's deployment context
- ✓ 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 (secure by default)	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 arising from accidental or uninformed misconfiguration by ensuring that 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 risk.
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 path 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 such states.
- **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, screenshots or documentation demonstrates that security implications are described in plain, non-technical language.
- **Baseline restore available:** screenshots, commands 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 can be taken to restore the secure baseline

4.22 Secure recovery and ownership life cycle

Principle **(secure by default)**	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 that 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 that 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 that recovery and transfer actions require a verified identity.
- **Abuse protections are applied:** configuration, logs or test evidence shows that rate limiting and monitoring of recovery workflows are in place.
- **Secure reset and transfer is verified:** test evidence confirms that factory reset and ownership transfer remove prior credentials and access.
- **Guided workflows are available:** screenshots, CLI commands or documentation demonstrates that recovery and transfer processes are guided.

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 remove prior access
- ✓ Clear guidance is provided on recovery and transfer workflows

4.23 Progressive adoption of the playbooks

This section provides an illustrative progressive adoption sequence, showing how the playbooks can be adopted and applied over time. The example begins with activities that provide context and prioritisation, followed by a foundational engineering baseline and then a broader and more mature implementation of the principles. Progressive adoption here refers to how organisations build and strengthen their security practices over time. It does not suggest delaying applicable requirements or controls, particularly those arising from legal obligations, including under the CRA. This guidance is without prejudice to such obligations, which take precedence.

4.23.1 Establish context and priorities

The product context, risk management and threat-modelling activities described in Section 2 can be used, together with Playbook 4.1, to identify the most relevant threats, trust boundaries and security objectives. This step can guide prioritisation and scoping as regards the implementation of the playbooks.

4.23.2 Establish a foundational engineering baseline

Based on the product context, security objectives, and identified risks threats, organisations should establish an appropriate engineering baseline. For most products, an initial baseline should include:

- secure coding and verification practices (Playbook 4.9);
- logging, monitoring and alerting (Playbook 4.10);
- vulnerability and patch management (Playbook 4.13);
- supply-chain controls (Playbook 4.14).

This baseline should be supplemented by the secure by default playbooks relevant to the product. For example:

- restrictive initial access (Playbook 4.16), where the product provides user, administrative or maintenance access;
- secure communication by default (Playbook 4.17), where the product communicates over a network or exchanges data with external components;
- unique device identity and secrets by default (Playbook 4.18), where devices, product instances or services require identities or credentials;
- automated maintenance and updates (Playbook 4.20), where the product can be updated after deployment.

Other playbooks should be prioritised at this stage where they are necessary to address the identified risks, threats and security objectives or product context. For example, availability-sensitive products may require early implementation of the incident response and recovery playbook (Playbook 4.12), while products supporting reset, ownership transfer or decommissioning may require the implementation of the secure recovery and ownership life-cycle playbook (Playbook 4.22).

Progressive adoption could also determine the extent to which an individual playbook is implemented. An organisation could begin with the checklist items that address its priority risks and applicable requirements, and then strengthen coverage, evidence and automation over time.

4.23.3 Broaden and strengthen implementation

The remaining applicable playbooks can be implemented and prioritised according to the product context, intended use, deployment context and identified risks and threats. Over time, teams should increase the coverage, consistency and automation. Moreover, progress should be assessed using relevant measures.

This illustrative sequencing is intended to help organisations manage implementation effort. It does not mean that playbooks addressed in a later phase are inherently less important or that applicable security or regulatory requirements may be deferred.

SECTION 5

Machine-processable attestation

5. Machine-processable attestation

5.1 Demonstrability, verifiability and assurance

In modern software engineering, the transition from manual, document-heavy compliance to machine-processable attestations marks a critical evolution in how trust can be verified. At its core, a machine-processable 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.

Machine-processable means that automated systems should be able to validate, interpret and act on it without human interpretation. This requires elements such as defined field types and semantics, controlled values, structured representation of decision-relevant information and schema versioning. Structured fields should be preferred for information used in automated processing, with free text used where additional explanation is helpful.

By 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-processable 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 block any container that lacks a valid, machine-signed attestation of a successful vulnerability scan by default.

Furthermore, machine-processable 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-processable attestations. Instead of being created as a static snapshot at a particular point in time, the artefacts 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.

A manufacturer-issued attestation by itself is not proof that a product is secure. Attestation, verification and independent assessment are distinct activities. An attestation provides claims and links it to supporting evidence; verification checks the integrity, scope, currency and consistency of those claims and that evidence; and independent assessment can evaluate whether the controls and evidence are sufficient for the product and its risks.

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

- **Demonstrability**: The proactive capacity of a system and its development process to provide objective, machine-processable evidence that specific security requirements have been implemented. It represents the shift from “claiming” security in a static document to “showing” security through generated artefacts, 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 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, because it means that every design decision, from the choice of a library to the architecture of a database, should be accompanied by the creation of a machine-processable artefact. This active generation of evidence allows teams to catch architectural flaws during the design phase, long before code is committed or products are shipped.

Furthermore, while secure 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 life cycle.

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 in which a system can refuse to operate if its attestations, such as signed proof that MFA is enforced or that the latest vulnerability scan was passed, are missing or invalid.

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

Machine-processable attestations can also support procurement and supplier assessment. Customers, integrators and procurers can use such structured claims and evidence to compare products, verify security-relevant properties and identify areas that may require further assessment.

5.3 Existing frameworks and initiatives

The ecosystem of machine-processable security has been rapidly evolving. Some of the key initiatives include foundational frameworks like **NIST's Open Security Controls Assessment Language (OSCAL)** (compliance as code), **OWASP's CycloneDX Attestations (CDXA)** (native security attestation) and **SPDX 3.0 Security Profile**, which provide the structural grammar for automating compliance and supply-chain transparency. Presented below are some of these initiatives in this area.

- **OSCAL** ([21])**.** Developed by NIST, OSCAL is the premier model for compliance as code. It provides a standardised way to express security control catalogues, system security plans and assessment results. By using OSCAL, organisations can automate the generation of compliance documentation, allowing for continuous authorisation, whereby the status of security controls is updated in real-time as evidence is collected.
- **CycloneDX (OWASP ECMA-424)** ([22])**.** While originally an SBOM format, CycloneDX has expanded into a full-stack transparency standard. Its attestation capabilities (CDXA) allow organisations to document claims about security requirements alongside generating a bill of materials. It also includes specialised modules like VEX to communicate whether a vulnerability actually affects a product, and it can produce a cryptographic bill of materials (CBOM) to inventory cryptographic assets for quantum readiness. CycloneDX Assessors Studio ([23]) provides an emerging open-source implementation for conducting structured assessments, collecting evidence, documenting claims and generating CycloneDX attestations.
- **SDPX 3.0 Security Profile** ([24])**.** The security profile captures security-related information in a SPDX Security Document. Specifically, the properties and relationships specified in the security profile are in support of exchanging information about system vulnerabilities that may exist, the severity of those vulnerabilities, and a mechanism to express how a vulnerability may affect a specific element including if a fix is available.
- **The Open Source Security Foundation (OpenSSF).** OpenSSF leads several projects, including **Security Insights** ([25]), a specification that project teams can use to report security facts (like bug bounty information or security contacts) in a machine-processable YAML format. They also champion the **OpenSSF Scorecard** ([26]), which automatically assesses open-source projects against security best practices. **OpenSSF Best Practices badge** ([27]) provides machine-readable security criteria status for many OSS projects, including against the **OpenSSF Baseline** ([28]). **Supply-chain Levels for Software Artifacts, or SLSA** ([29]) ("salsa") provides recommended machine-readable schemas for SLSA attestations.
- **OWASP** (Open Web Application Security Project). Beyond CycloneDX, OWASP projects like the **ASVS** ([30]) (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.

([21]) https://pages.nist.gov/OSCAL/.
([22]) https://cyclonedx.org/capabilities/attestations/.
([23]) https://github.com/CycloneDX/cyclonedx-assessors-studio.
([24]) https://spdx.dev/learn/areas-of-interest/security/.
([25]) https://openssf.org/projects/security-insights/.
([26]) https://openssf.org/projects/scorecard/.
([27]) https://www.bestpractices.dev/.
([28]) https://baseline.openssf.org/.
([29]) https://slsa.dev/spec/v1.2/attestation-model.
([30]) https://owasp.org/www-project-application-security-verification-standard/.

- **TC54 (Ecma International)** ([31]): 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.
- **in-toto** ([32])**.** in-toto provides an open framework for expressing verifiable claims about software supply-chain activities. SLSA Provenance uses in-toto to describe where, when and how software artefacts were built and tools such as Sigstore Cosign ([33]) can sign and verify in-toto attestations.
- **Device Security Passport** ([34])**.** The Device Security Passport (DSP) is intended to provide a structured, machine-readable record that consolidates relevant security information about an IoT component and makes it accessible to stakeholders across the supply chain.

5.4 Illustrative example

This section describes an illustrative example of how security claims, supporting evidence and assessment results can be expressed in a structured, machine-processable format to describe aspects of a product's security posture.

The example does not propose a new schema or prescribe a specific format; rather, it is intended to show the relationship between a security objective, its implementation and the evidence used to assess it. Existing standards and specifications may be used, separately or together, to represent this information.

The example follows a simple cascade pattern, in which each high-level security objective is linked to its implementation actions and the results used to verify it.

- **Control layer (security objectives to be addressed)**
  - Defines structured security objectives (e.g. 'protection against unauthorised access' or 'secure update delivery'), which may be derived from secure by design and default principles and/or aligned with regulatory or recognised cybersecurity frameworks.
- **Implementation layer (how controls are implemented)**
  - 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 artefacts.

These layers do not need to be contained in a single document. Claims, SBOMs, vulnerability information, build provenance, test results and other evidence may be maintained as separate but linked machine-processable artefacts.

([31]) https://tc54.org/tea/.
([32]) https://in-toto.io/.
([33]) https://docs.sigstore.dev/quickstart/quickstart-cosign/.
([34]) https://dossproject.eu/the-doss-device-security-passport-dsp/.

In cases where component identifiers (such as PURL ([35]), CPE ([36]) or SWHID ([37])) are referenced, security claims and evidence can be linked to specific software libraries, firmware modules or hardware components listed in an SBOM. This allows a recipient to determine not only that a security activity was performed but also which components, product versions or builds were covered by the referenced assessment.

Given that this information may contain sensitive technical details, such as specific compiler flags, internal test logs or cryptographic configurations, it may be impractical or insecure to grant every stakeholder full visibility. Sensitive evidence does not need to be included in a single publicly accessible record. Manufacturers may separate the information into:

- a **public attestation layer** – cryptographically signed security claims, provenance information and other non-sensitive information that can be independently verified;
- a **restricted technical overlay** – detailed test results, tool configurations, build parameters and other sensitive evidence, made available to authorised recipients through an authenticated and access-controlled API endpoint, such as the Transparency Exchange API.

5.4.1 Organisation of the example

For the purposes of the example, the information is organised as shown below. This organisation is illustrative and does not define a schema, field names or a required implementation format.

Information group	Function
Product identity and scope	Defines product identity, versioning, scope to which the information applies and the manufacturer's cryptographic signature
Control layer	Defines structured security objectives aligned with requirements, security principles and/or applicable security policies or regulations
Implementation layer	Maps identified threats and security objectives to implemented controls, secure-default settings and supporting implementation information
Assessment and verification layer	Records assessment results and links to supporting evidence, such as test results, configuration checks, SBOMs and build-artefact hashes

5.4.2 SafeGate-X1 example

Scenario. Imagine you are the manufacturer of 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.

([35]) https://github.com/package-url/purl-spec.
([36]) https://nvd.nist.gov/products/cpe.
([37]) https://www.swhid.org/.

The goal. You want to prove to your industrial clients that SafeGate-X1 is not just a black box and is built following the secure by design principles of **least privilege**, **minimal attack surface, strong identity** and **vulnerability and patch management**.

Disclaimer: This Machine-processable attestation example and the SafeGate-X1 scenario are provided solely as conceptual models to illustrate the practical integration of secure 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 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.

SafeGate-X1 is modelled 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.

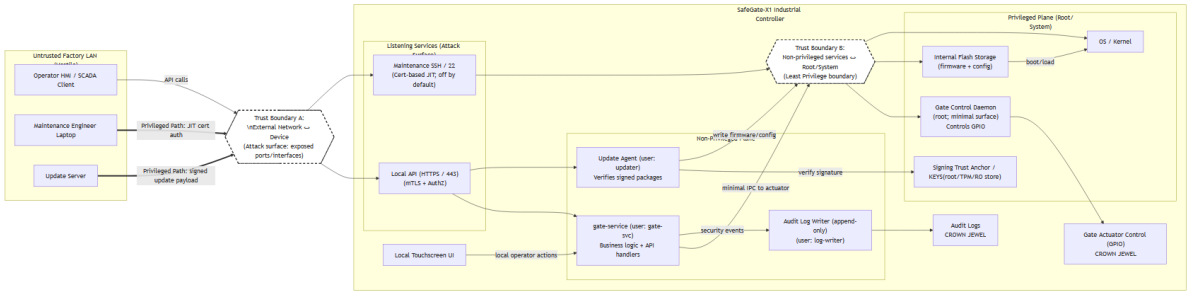

Figure 4: Trust boundaries diagram

Source: Built with https://mermaid.live.

Note that Figure 4 provides an illustrative system representation for this simplified example. It is not intended to prescribe a standard notation or a particular tool. Other suitable approaches, including dedicated threat-modelling tools, may be used.

The threats in this example are specific to the illustrated product and should be adapted to the product's functionality and deployment context. The example below uses likelihood/impact ratings for prioritisation as an example

ID	Threat scenarios	Boundary	Likelihood	Impact	Priority
T1	RCE via web API: attacker exploits a buffer overflow in the web server to gain shell access	Trust Boundary A (HTTPS)	Medium	High	**Top**
T2	Privilege escalation: attacker uses a compromised web service to gain root privilege 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	Medium	**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	**Medium**

5.4.2.2 Controls, defaults and verification

This section maps the threats to specific principles and verification methods.

Threat	Mitigation control (principle)	Secure by default setting	Verification method
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 production	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 certificates	`PasswordAuthentication no` in `sshd_config`	Config validation: script checks SSH config against hardened baseline before release
T5	**Vulnerability and patch management**: signed updates and immutable file system	Root file system is mounted read-only by default	Mount check: CI test verifies `/usr/bin` is not writable by any user

5.4.2.3 Applying the three layers

The SafeGate-X1 example can be read across the three layers as follows.

- **Control layer**
  - Records the strategic objectives and their relationship to the identified threat scenarios (T1–T5), for example: 'Unauthorised gate operation must be prevented.'
- **Implementation layer**
  - Identifies relevant principles and controls, for example: 'We apply least privilege and attack surface minimisation.'
  - Describes the technical settings used to implement them, for example: 'We run as `UID 1001`, use Linux capabilities and close all ports except 443.'
- **Assessment and verification layer**
  - Records the checks performed, release gates and SBOM hashes, for example: '`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** ([38]) base image.
- **Gate.** An automated **Nmap scan** runs against the deployed release candidate. It compares the active ports against the `ports_allowed` list in the JSON. If a mismatch is found, the release fails and the firmware is rejected.

The JSON fragment presented as Figure 5 illustrates how information from the three layers could be expressed in a machine-processable form. The field names and structure are illustrative only and do not define a specific schema or implementation format. In practice, this information may be represented using existing standards and specifications, like those described in Section 5.3.

([38]) https://buildroot.org/; https://opensource.suse.com/bci-docs/guides/building-a-distroless-image/.

```
{
  "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"
  }
}
```

Figure 5: Example of machine-processable security information

Least privilege (T2)

- **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 that the principle of least privilege is hard-coded into the operating system 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 and patch management (T5)

- **Logic.** Even following a compromise, the attacker should not be able to achieve persistence by modifying system files.
- **Implementation.** The root file system is mounted as **read-only**.
- **Gate.** A **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, as it has been proved that the secure by default setting was not correctly applied.

5.4.2.4 Illustrative example of third-party verification

The following table shows how machine-processable security information may be consumed by third-party assessors to identify relevant claims, locate supporting evidence and perform targeted verification activities.

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 and patch management	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

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. **PURL.** Accessed 2026. https://github.com/package-url/purl-spec
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. **CycloneDX** Assessor Studio, 2026, https://assessors.studio/
28. **OpenSSF**. Security Insights Specification. Accessed 2026. https://openssf.org/projects/security-insights/
29. **OpenSSF**. Scorecard. Accessed 2026. https://openssf.org/projects/scorecard/
30. **OWASP**. Application Security Verification Standard (ASVS). Accessed 2026. https://owasp.org/www-project-application-security-verification-standard/
31. **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
32. **TC54**. Transparency Exchange API Specification. Accessed 2026. https://tc54.org/tea/
33. **Buildroot**. Buildroot, Making Embedded Linux Easy. Accessed 2026. https://buildroot.org/
34. **openSUSE**. How to build a distroless image using SLE BCI. Accessed 2026. https://opensource.suse.com/bci-docs/guides/building-a-distroless-image/
35. **CISA** Categorically Unsafe Software. 2024. https://www.cisa.gov/news-events/news/categorically-unsafe-software
36. **Padmos, A.** An extensive list of resources related to threat modelling. Gotta catch 'em all!. Accessed 2026. https://github.com/arnepadmos/threats

Annex A: Abbreviations

AI	artificial intelligence
API	application programming interface
CDXA	CycloneDX Attestations
CI	configuration item
CI/CD	continuous integration and continuous delivery
CLI	command-line interface
CRA	Cyber Resilience Act
CWE	common weakness enumeration
DAST	dynamic application security testing
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IAM	identity and access management
IoT	internet of things
JSON	JavaScript Object Notation
MFA	multi-factor authentication
ML	machine learning
NIST	US National Institute of Standards and Technology
OpenSSF	Open Source Security Foundation
OSCAL	Open Security Controls Assessment Language
OTA	over the air
OWASP	Open Worldwide Application Security Project
PR	pull request
SAST	static application security testing
SBOM	software bill of materials
SLA	service-level agreement
SLSA	supply-chain levels for software artefacts
SMEs	small and medium-sized enterprises
SPDX	Software Package Data Exchange
SQL	Structured Query Language
SSH	Secure Shell
STRIDE	spoofing, tampering, repudiation, information disclosure, denial of service and elevation of privilege
TLS	Transport Layer Security
VEX	Vulnerability Exploitability eXchange
YAML	YAML Ain't Markup Language or Yet Another Markup Language

Annex B: Cyber Resilience Act 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	[Manufacturers of products with digital elements shall:] 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.

Annex C: Mapping security principles to Cyber Resilience Act 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 authentication 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 configuration 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
Life-cycle 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 life cycle
	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 configuration that users can reasonably adopt and maintain
Secure coding and verification 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 life cycle
	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 protection of transmitted data 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 that 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 life cycle	**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

TP-01-26-016-EN-N

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

ISBN 978-92-9204-802-0