Charter
of Trust

Charter of Trust – Secure Development Lifecycle: step-by-step guidelines

Charter of Trust – Principle 3
Classification CoT Public

Summary

On February 16, 2018, at the Munich Security Conference, the cornerstone for the Charter of Trust (CoT) was laid to make the digital world more secure. A continuously growing group of multinational companies has signed off on this cybersecurity initiative by endorsing its 10 fundamental principles, which foster three important objectives:

- To protect the data of individuals and companies
- To prevent damage to people, companies and infrastructures
- To create a reliable foundation on which confidence in a networked, digital world can take root and grow.

The Security by Default Taskforce: our activities

One of the key fundamental principles of CoT – Security by Default (Principle 3 Taskforce) – has worked on the following:

- Phase 1 (Products, Functionalities, Technologies)- baseline security requirements and explanatory document
- Phase 2 (Processes, Operations, Architectures) - baseline security requirements and explanatory document

Objective of the document

The purpose of this document is to provide additional information on a step-by-step approach for achieving secure development lifecycle, in addition to the Phase 1 and Phase 2 baseline requirements. The document aims to provide a deep dive into the topic of secure development lifecycle and define best practices for achieving the same. This includes the following steps:

- Identifying the basic steps for a development lifecycle model
- Developing best practices for a secure development lifecycle

Target audience

The document aims to provide guidance to current members of the Charter of Trust, future members of the Charter of Trust and other stakeholders who would like to adopt a Secure Development Lifecycle Approach in their Security by Design and Default Strategy.

Content

Abbreviations

- SDLC: Secure Development Lifecycle
- DevOps: Development Operations

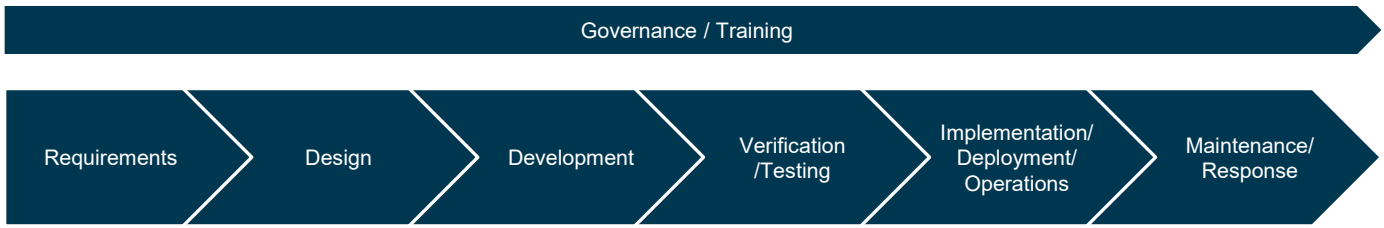
Introduction

Development activities are key for any company providing products and services to its customers. Even when it is not an explicit requirement, cybersecurity has become an implicit expectation from customers and consumers. Cybersecurity must be considered from the earliest stages of the product or service development to its end of life, within a so-called Secure Development Life Cycle process (SLDC).

The Principle 3 Taskforce has defined, in Phase 1, a set of 19 baseline requirements for Products, Functionalities and Technologies, in order to drive integration of baseline security mechanisms into the products.

In Phase 2, the Principle 3 Taskforce has defined a set of 17 baseline security requirements for Processes, Operations and Architectures, to support organisations in implementing secure development processes and environment.

The current document bridges those two sets of baseline requirements, by showing step-by-step how a product or service can be designed, within a secure development process, and integrate the baseline security mechanisms.



Guidelines for secure development lifecycle

1. Governance and training

The SDLC is a general concept that needs to be adapted to each organisation's specificities. Nevertheless, a clear governance needs to be in place to ensure an efficient and trustworthy SDLC. The governance should define:

- Rules for the physical & logical security of the development environment in order to ensure the integrity (and confidentiality, if needed) of the design (especially of the security mechanisms).
- Rules for authorising developers to work on products, or employees to access product information, based on security training and awareness, skills management, and need to know.
- Processes for capturing security functional requirements (e.g. regulatory requirements, customer requirements or based on risk analysis) and including them in the product design in the form of security mechanisms (see Principle 3 Phase 1 baseline requirements).
- Processes for progressing the design from one step to another with milestones, validation steps, clear responsibilities, traceability, and separation of roles.
- Processes for handling points of vulnerability discovered in products already in the field, with a reporting portal, responsible disclosure, and corrective/preventive field upgrade.
- A Quality Management System able to measure the efficiency of the governance and to drive the modification of any process or procedure that needs to be enhanced.

In addition, it is recommended for higher security levels that at least one dedicated team of security experts is in charge of security survey, internal security testing, support for security architecture, support for security certification.

2. Requirements

The following sections provide a set of guidelines for identification, assessment and validation of security requirements, when developing a new solution. These guidelines are best practice to ensure that a robust set of security requirements are developed as part of the overall requirements set for a solution.

2.1 Data Protection Requirements

All data within the remit of protection should be identified and for each data type being stored, transmitted or manipulated, the following eight key best practices for identifying data protection requirements should be considered:

- Protection of Information at Rest and Transit
- Data Tagging
- De-identification (Removal, Masking, Encryption, Hashing, or Replacement of Direct Identifiers)
- Minimisation of Personally Identifiable Information, (Removal, Anonymisation, Pseudonymisation).
- Non-persistance & Non-persistent Information
- Media Sanitisation
- Data Separation
- Compliance with Applicable Data Protection Regulations

2.2 Identity and Access Management Requirements

Once the data has been identified and the requirements for protection of that data have been defined, the security of all the aspects of identity and access management should be considered.
To help with this, the following nine identity and access management best practices for requirement definitions should be considered:

- Device Identification and Authentication
- Users Identification and Authentication
- Service Identification and Authentication
- Access Enforcement
- Central Management & Account Management
- Authenticator Management
- Security Clearances for Regulated Systems
- Citizenship Requirements for Regulated Systems
- Identification and Authentication

Using the Identity and access management guidance to identify the requirements will enable an organisation to:

- Confirm the identity of a new user and ensure that the level of trust and access given is commensurate with their personal and professional background.
- Bind an identified user to an identity within the organisation system with an appropriate method of authentication.
- Ensure that the authentication method gives confidence that when an identity is used, it is being used by the member of staff whose identity has been previously validated.
- Apply the principle of least privilege to limit the access or functionality that different users have.

2.3 Secure Design Requirements

By identifying the security design principles, security will be built into the design at the start of the system lifecycle. This ensures that security is considered by design and not as a separate entity after the core design of a system has been completed. To help with this, the following seven best practice guidelines for requirement definition should be considered:

- Reduced Complexity
- Secure Evolvability
- Trusted Components
- Repeatable and Documented Procedures
- Sufficient Documentation
- Requirements for Functions, Ports, Protocols, and Services in Use
- Requirements for Processing, Storage, and Service Location

2.4 Secure Configuration Requirements

Secure configuration refers to security measures that are implemented when building and installing computers and network devices to reduce cyber vulnerabilities. Manufacturers often set the default configurations of new software and devices to be as open and multifunctional as possible. Configuration settings should default to a secure state. To help with this, the following five best practice guidelines for requirement definition should be considered:

- Management and Application
- Verification
- Malicious Code Protection
- Configuration Change Control
- Security Response

2.5 Communications Protection Requirements

Communications will typically transit an untrusted network, such as the internet. An untrusted network is outside of control, which means someone may be able to access (or modify) data transiting that network. In order to secure communications, requirements should be identified using the following five best practice guidelines for requirement definition:

- Requirements for Boundary Protection
- Requirements for Transmission Confidentiality and Integrity
- Requirements for Intrusion Detection and Prevention System
- Requirements for Remote Access
- Requirements for Data in Transit

2.6 Architecture Resilience Requirements

The requirements are business-driven, availability, and reliability, characteristics for the system which specify recovery tolerances from disruptions and major incidents. The level of required resilience is dependent on business requirements, alongside any law, regulations or legal responsibilities that a business may have. In order to identify architecture resilience requirements, the following three best practice guidelines for requirement definition should be followed:

- Requirements for System Backup and Recovery
- Requirements for Failure Prevention
- Contingency Planning

2.7 Secure Monitoring and Remediation Requirements

These requirements can be either business-driven or required as part of national laws. Regulation or reporting to governmental agencies. Working to identify these requirements will aid operational security, and remediation of threats and issues that are identified. In order to identify secure monitoring and remediation requirements, the following ten best practice guidelines for requirement definition should be used:

- Monitoring and Scanning
- Penetration Testing
- Testing and Evaluation
- Risk Response
- Flaw Remediation
- Event Logging
- System Monitoring
- Content of Audit Records
- Information and Audit Record Retention
- Protection of Audit Information

3. Design

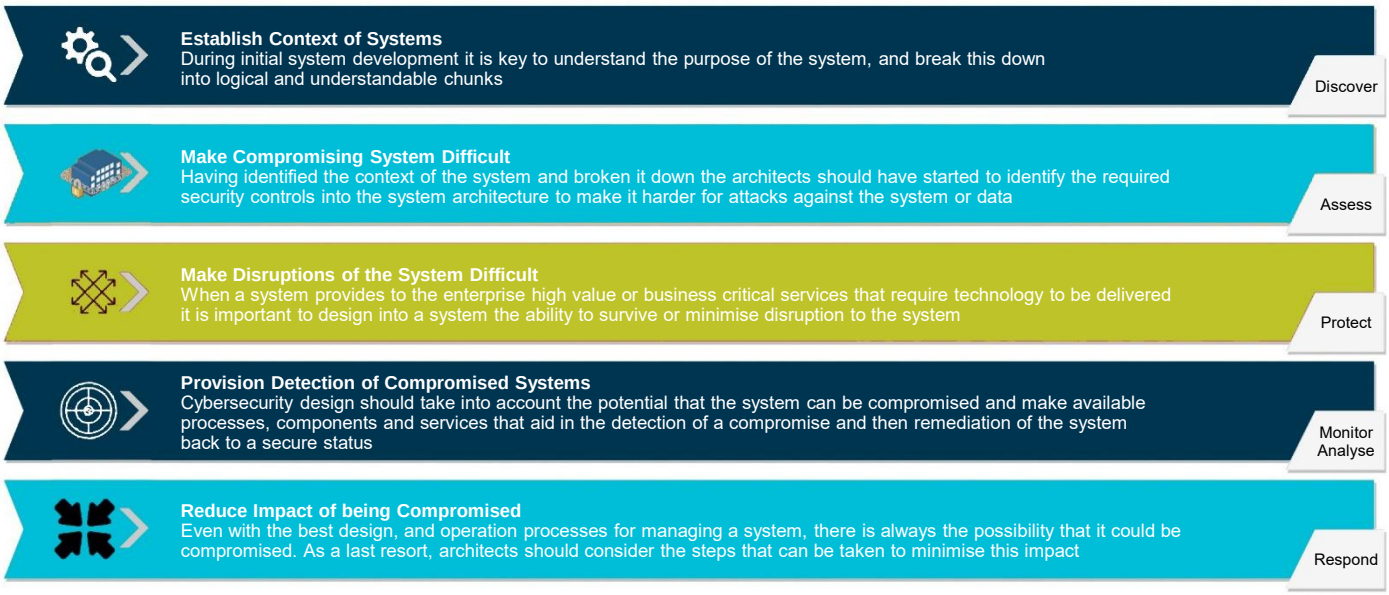
3.1 Cybersecurity Design Principles Overview

This section provides cybersecurity design principles that apply to the creation of system designs that will be implemented. Securing a new or an existing system against potential threats requires the use of multiple overlapping protection approaches (defense-in-depth) that should be applied at the start of the system life cycle, during the design and development stage.

Using a multilayered approach with overlapping protection decreases the impact of failure or circumvention of an individual protection approach, and thus will not leave the system completely unprotected in event of security breach.

Effective security design should also review non-technical issues, such as policies, operational procedures, and user education and training.

The following diagram shows the breakdown of cybersecurity design principles into sections:



3.2 Establish Context of the System before Designing

During initial system development, it is key to understand the purpose of the system, which should include:

- What benefit does it bring to the business?
- What data will be used?
- What connectivity will be required?
- What user groups will access / use the system and in what capacity?
- What other existing / new systems will be required for the solution to be operated.

With this in mind, the architects should detail the risks that come with the development and operation of the system, what the impacts of these risks are, and more importantly what level of risk acceptance is expected. This information should be captured within the risk analysis for the system design.

This initial contextualisation of the system will provide a baseline, giving a foundation upon which work can be based, where all people involved in the system design have a common understanding and can make informed design decisions based around this baseline.

This activity done early in development, builds in at the design stage, a secure viewpoint that acts as the anchor for all ongoing development of the cybersecurity for the system.

3.3 Ensure Compromising the System is Difficult

Security should be built in at the baseline of any system at the earliest stage of development. Having identified the context of the system and broken it down, the architects should start to identify the required security controls into the system architecture to make it harder for attacks against the system or data.

The following sections break down some key concepts on how to ensure that compromising the system is as difficult as possible:

- External Data Entry should not be Explicitly Trusted
- Minimise Attack Surface Available
- Ensure Confidence in Critical Security Controls
- Protect Operational and Management Systems from Attack
- Use Trusted System Components Where Possible
- Operations Should be Explicitly Authorised and Auditable
- Build in Ease of Maintenance
- Provide Unified Access Control with Easy Management
- Ensure the System is User Friendly

3.4 Make Disruption of the System Difficult

When a system provides to either high-value or critical business services to an enterprise, services that require technology to be delivered, it is important to design into a system the ability to survive, minimise or be resilient to disruption.

The following details some general steps that should be taken during system design to make the system as resilient as possible, where downtime can be as close to zero as practically possible.

- Design System Resilience Against Attack and Failure
- Design Scalability into System
- Identify Scenarios and Test for High Load and Denial of Service
- Identify Conditions where System Resilience Depends on a Third Party

3.5 Provision for Detection of Compromised Systems

Even the most secure system has the potential to be compromised. Complete cybersecurity design should take into account the potential that the system can be compromised and make available processes, components and services that aid in the detection of a compromise and then remediation of the system back to a secure status.

The following sections detail the ways in which the architect can provision for the detection of compromised systems:

- Design for Simplified Communication Flow
- Collect Events and Logs of System activity
- Design Detection Capability for Malware Command and Control
- Ensure Independence of the Monitoring System
- Obfuscate Security Rules from External Testing
- Detail Normal Operations and Detect Abnormal Operation

3.6 Reduce the Impact of being Compromised

During the initial design of a system, there should be a phase for considering what can be done to minimise the impact of a system being compromised. Even with the best design and operation processes for managing a system, there is always the possibility that it could be compromised. As a last resort, architects should consider the steps that can be taken to minimise this impact.

The following provides design options for minimising the impact of a system being compromised.

- Zone and Segment Network where possible
- Disable or Remove Superfluous Functions
- Provision for Fast Data Recovery Post Compromise
- Provide for Separation of Duties
- Utilise Data Anonymisation for Reporting
- No Unnecessary Caches of Data

4. Development

Cybersecurity development principles constitute good practices so that security is achieved at the development stage.

4.1 Software development

The main part of the development is the actual writing of the code and the building of the software. Software is usually built from components that are created from own code and 3rd-party components that require a commercial license or are considered Open Source Software (OSS). The guidance provided focuses mainly on the security of the code that is created by the organisation and the security for OSS. Security of software with a commercial license should be covered by suitable security measures for the supply chain.

A secure coding guide should be created that defines the secure coding principles that an organisation needs to adhere to. The secure coding principles should include the following non-exhaustive list of principles and measures:

- Provide trainings on Secure Coding principles for developers
- Implement suitable protections for critical data assets and implement functional security requirements according to specifications
- Follow the security by default principle and establish a secure default configuration. Check Charter of Trust Security Baseline Requirements for “Products, Functionalities, Technologies”
- Secure the development environment
- Separate the development, testing, and production environments
- Do not use production data as test data within the development and testing environments
- The concept of least privilege requires that the services/users should only have the privileges that are required to perform a certain function/task
- All input should be validated in order to avoid the possibility that malicious input is entered in the code. Malicious input can include code, scripts and commands, which, if not validated correctly, can be used to exploit vulnerabilities
- Ensure that error handling does not result in the leaking of sensitive information
- Use state-of-the-art cryptography to protect the confidentiality and integrity of sensitive data
- Consider security best practice for application security, like the OWASP Cheat Sheet Series, which provide security guidance on various application security topics.

Security coding operations that should be considered include:

- Establish configuration management and version control to uniquely identify the software assets
- Adhere to general security requirements that govern the secure development lifecycle within the organisation
- Check OSS-components before embedding them into the software
- Check whether components are provided from a trusted source
- Evaluate the impact of vulnerabilities on the product
- Check whether the OSS-project is maintained and provides regular updates
- Check whether information is provided about security vulnerabilities
- Include static and dynamic code analysis in the software development phase to identify secure coding issues early in the development cycle.

- For security sensitive code areas consider code reviews applying a 4-eyes principle
- Code signing should be used to protect the code from tampering
- Regularly update frameworks or other development to minimise risks arising from vulnerable frameworks
- Securely archive source code, development data and documents
- Develop customer documentation to support the secure configuration/deployment and operation of the product
- The software asset should be provided with configurations and configuration guidance that facilitate secure installation and operation.
- Software should be configured to have secure settings enabled by default, to reduce the likelihood of the software being deployed with weak security settings, putting it at greater risk of compromise.

4.2 Hardware development

Specific requirements for hardware development such as including security measures, e.g., anti-tampering features, a logically isolated hardware security module (HSM), or physical unclonable functions (PUF) need to be considered. However, this is beyond the scope of the current document.

5. Verification & testing

Cybersecurity verification & testing principles constitute good practices so that security is achieved at the verification & testing stage.

Testing is an essential part of the secure development lifecycle, providing the necessary assurance that implemented the security measures provide the desired security. Testing starts in the development phase and extends into the operation and maintenance phase. This chapter seeks to outline a baseline for security-related testing activities. Depending on the purpose of the product, e.g., if a product intends to achieve a security certification according to a specific standard, additional testing and verification activities are expected. These extended verification activities are considered out-of-scope for this chapter.

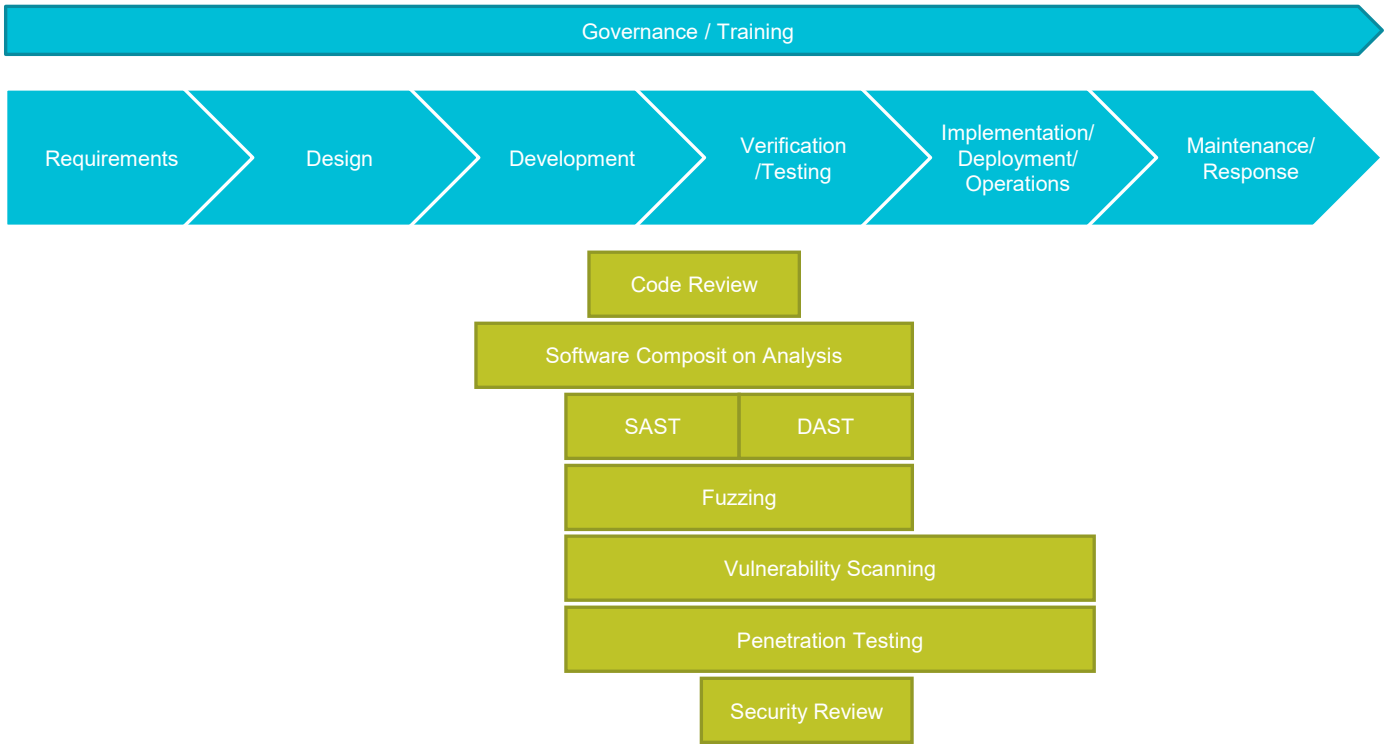


Image: Testing and Verification included into the development lifecycle

5.1 Code reviews

Code reviews (sometimes referred to as peer review) is a software quality assurance activity in which one or several people check a program, mainly by viewing and reading parts of its source code, and they do so after implementation or as an interruption of implementation. At least one of the persons must not be the code's author.

Code reviews may be done when new code is being developed or existing code is modified.
Before the new code is committed, it is reviewed by a fellow developer that was not involved in the programming. Another strategy is to perform code reviews with a team of developers. This has an educational advantage for both the developer and the reviewers and helps to increase the secure coding skill within the development team.

5.2 Software Composition Analysis (SCA)

Software as a whole is composed of various parts, including software that the organisation has developed on its own (own source code), commercially licensed 3rd-party components and open-source components. As most applications nowadays contain open-source software, the management of the security of these components is critical for the security of the product. Software Composition Analysis is a term used in the industry to describe the activities performed to analyse and identify the components from which a software is built. The result of this analysis is a Software Bill of Material (SBOM). The SBOM is used in the supply chain to manage security and license compliance risk. There are tools that support development organisations with the SCA and the subsequent creation of the SBOM. For more information about these tools, reference the Tools Listing section of the OWASP Component Analysis website.

5.3 Static Application Security Testing (SAST)

Static Application Security Testing (SAST) is a method for source code analysis that helps software developers identify security vulnerabilities during the code development. SAST-tools can be integrated into the Integrated Development Environment (IDE) helping developers find vulnerabilities early in the development cycle. SAST is analysis and application in a non-running state. The OWASP Source Code Analysis Tools website provides additional information, including a list of available SAST-tools.

5.4 Dynamic Application Security Testing (DAST)

Dynamic Application Security Testing (DAST) is a testing methodology that is applied to an application when it is executed. The system is tested with specific inputs to look at specific forms of behaviour. Web Application Vulnerability Scanners are testing web applications and support the identification vulnerabilities described, like cross-site scripting, SQL Injection, Command Injection, Path Traversal and insecure server configuration. The OWASP Vulnerability Scanning Tools site provides a list of tools that may be used for web application scanning.

5.5 Fuzzing

Fuzzing is an automated testing method that uses random or semi-random data and injects it into the interfaces of software applications. It is used to identify input validation issues and vulnerabilities. Specifically crafted input can be used to check for memory leaks, buffer overflows and input validation errors.

5.6 Vulnerability Scanning

Vulnerability scanning may be useful to test an application in a running environment, before or after additional security measures are implemented, when the application is deployed. It provides valuable insight about exposed vulnerabilities comparable to vulnerability management operated within a customer environment. It supports the prioritisation and remediation of exposed vulnerabilities.

5.7 Penetration Testing

Penetration testing is a testing technique that is actively looking for vulnerabilities and validates whether an attacker is able to exploit the vulnerabilities. It may be executed in various forms, such as the back-box test (with no knowledge of the application), the grey-box test (with some knowledge on the application), or the white-box test (with significant knowledge and support from the development team). A black-box test may be useful for testing the exposure of an application after it has been securely deployed with all the security hardening measures applied. Grey or white-box testing is more effective in identifying security issues in software. Penetration testing complements automated testing that has been done in earlier stages of the development lifecycle. Cooperation between penetrations testers and the development team helps developers understand the underlying issues and fix the vulnerabilities.

5.8 Security Review

Before a product is released, usually a security review is done as a joint activity by the product development and the security team. The review is done based on pre-defined criteria that are evaluated during the review. The following criteria may be considered as part of the review:

- Completeness of security measures planned within the SDLC
- Evaluation of the implementation against baseline security requirements
- Evaluation of the results of security testing activities
- Existence and completeness of security documentation required for secure deployment and operation
- Preparedness to support vulnerability management during the maintenance phase

As a result of the review, the release may proceed unconditionally, with restrictions, or after risk acceptance by the relevant stakeholders. A mitigation plan should be agreed for deviations from expected results. Major issues that remain unfixed upon release should require formal risk acceptance by the relevant management stakeholders.

6. Implementation, deployment, operations

Software asset deployment is the process required to make new or updated software assets available to its users.
Deployment typically includes activities such as provisioning environments, installing, and testing the deployed asset. Deployment also includes ongoing monitoring of the health and performance of newly deployed environments.

Deployment-related activities should take the following security aspects into consideration:

6.1 Awareness and training

The organisation's personnel and partners should be provided with cybersecurity awareness education and be trained to perform their cybersecurity-related duties and responsibilities, consistent with related policies, procedures, and agreements.

6.2 Data security and information protection

Information is managed consistently with the organisation's risk strategy to protect the confidentiality, integrity, and availability of information.

Security policies that address purpose, scope, roles, responsibilities, management commitment, and coordination among organisational entities, as well as processes and procedures are maintained and used to manage protection of information systems and assets.

6.3 Identity Management, authentication, and access control

At the deployment phase and during asset operations, the organisation should apply all baseline requirements and best practices related to Identity and Access Management. Access to physical and logical assets and associated facilities is limited to authorised users, processes, and devices, and is managed in a manner consistent with the assessed risk of unauthorised access.
Strong authentication mechanisms should be in place and enabled by default. Authorisation should be used to ensure legitimate use and mediate attempts to access resources in the various deployment environments.

6.4 Resilience

Technical security solutions are managed to ensure the security and resilience of systems and assets, consistent with related policies, procedures, and agreements.
The information system and assets are continuously monitored to identify cybersecurity events and verify the effectiveness of protective measures.

6.5 Configuration and documentation

Software configuration should take place according to the guidance provided with the vendor.

Software configuration settings should be altered to tailor security settings to the operating environment.

The deployment phase should manage the dependencies of all assets in the environment and use version control to keep control of the environment.

The organisation should ensure a strong and formal progressive deployment strategy that enables rollbacks in case something goes wrong.

The deployment strategy should separate a staging environment on which deployment can be automatic, and a production environment where deployment is manual, with checklists that make sure that nothing is forgotten.

The software documentation specifies configuration parameters that are as restrictive as feasible, to make sure the software is as resistant as possible to anticipated attacks and exploitation. The software documentation describes configurations and procedures for secure configuration under normal operation. The software documentation describes secure installation procedures for initial installation and installation for additional components, updates, and patches.

6.6 Vulnerability management

The organisation should maintain an up-to-date vulnerability management plan. Vulnerabilities are identified and resolved rapidly and comprehensively, according to risk-based prioritisation. The organisation maintains a coordinated vulnerability disclosure program.

6.7 Patching

Vendors disseminate timely patches or updates to address identified security issues. Patches or updates are developed and disseminated based on risk-informed prioritisation, in accordance with the vendor's vulnerability management program. Patches or updates are subjected to testing for functionality and security prior to release. All patches and updates are documented.

Patches or updates should be deployed securely. Deployment of patches or updates could be coordinated with other vendors where appropriate to address multi-vendor security issues or supply chain security issues. Patches or updates for security issues should be accompanied by advisory messages informing users of relevant information.

6.8 Secrets management

Software is developed in accordance with an encryption strategy that defines what data should be encrypted and which secure encryption mechanisms should be used.

Every stage in the application lifecycle uses secrets and certificates that must be stored securely. Effective secret management should be in place to ensure that every secret, every password, access token, and certificate is managed.

Deployment teams should store no secrets anywhere in the code or on team environments, rather they should be kept within key vaults.

7. Maintenance and response

Devices in the field are subject to numerous security attacks, and there is a high chance that any organisation has to deal, one day or another, with vulnerabilities discovered in their products.
As such, it is very important for organisations to be prepared, equipped with the appropriate processes and procedures for handling those security vulnerabilities.

The vulnerabilities reporting can take several forms:

- Report from users/researchers through a dedicated portal maintained by the organisation. This is considered to be the minimal effort for organisations selling secured products.
- Report from a dedicated security expert team internal to the organisation. Organisations dealing with very high levels of security often have an internal security team dedicated to the survey and the internal testing of its own products.
- Any reported vulnerability will be analysed internally for an assessment within a reasonable timeframe, and any critical vulnerability should be made public (responsible disclosure) in order to advise users on what to do while waiting for a fix.
- The organisation will work on fixing the vulnerability and update the concerned devices through field update. It is then important to implement field upgrade security mechanisms on devices that can support it (in terms of performance) as this is the easiest (and often only) way to correct a vulnerability on devices that are in the field.
- Some field upgrades may not be related to discovered vulnerabilities, but more related to preventive upgrades in the view of strengthening the product (e.g., with new crypto primitives and enhanced protocols).
- In both cases (corrective or preventive upgrade), field upgrade will guarantee a secure handling of user data for an extended lifetime.

8. Security by default in agile SDLC and DevOps

Agile software development is an umbrella term for a set of frameworks and practices.[1]
These frameworks and practices include the development of requirement discovery and solution improvement, done through the collaborative effort of cross-functional teams together with the future users of the system, as well as early and incremental delivery (often called “sprints”), and flexible responses to changes in requirements.

In a nutshell, integrating Security by Default principles into an agile SDLC means performing the same tasks and steps as in non-agile SDLC. Nevertheless, in an agile one, they take on a new character.

From an organisational perspective, the cross-functional team includes a security subject matter expert as a permanent member.

In addition, many of the security practices must be performed in each of the sprints with every release, e.g., running static code analysis, performing code reviews, and creating threat models for all new features.

Other security practices are performed on a regular basis, which can be spread across several sprints. These include, for example, carrying out and continuously reviewing risk assessments, analysing attack surfaces, random testing.

DevOps takes the agile SDLC even one step further. The cross-functional team is expanded from development (Dev) to include operations (OPS), bringing together formerly separate tasks. In addition, many of the tasks are automated to speed deployment and increase the quality of deliverables.

With DevOps, the integration of security activities also means a high level of automation, which is referred to as DevSecOps. DevSecOps is DevOps that continuously integrates and automates security throughout the DevOps lifecycle.

---

[1]According to the Agile Alliance (https://www.agilealliance.org/agile101/)

9. External references

9.1 Links to previous Charter of Trust material

- Charter of Trust Phase 1 Baseline Requirements (Products, Functionalities and Technologies): https://www.charteroftrust.com/wp-content/uploads/2020/05/200212-P3-Phase-1-Baseline Requirements_FINAL.pdf
- Explanatory Document for Charter of Trust Phase 1 Baseline Requirements : https://www.charteroftrust.com/wp-content/uploads/2021/06/Charter-of-Trust-Principle-3-Achieving Security-By-Default-Explanatory-Document.pdf
- Charter of Trust Phase 2 Baseline Requirements (Processes, Operations and Architectures) http://www.charteroftrust.com/wp-content/uploads/2022/08/P3-Phase-2-Baseline-Requirements.pdf

- Explanatory Document for Charter of Trust Phase 2 Baseline Requirements: https://www.charteroftrust.com/wp-content/uploads/2021/12/P3-Phase-2-Explanatory-Document.pdf

9.2 Further references on Chapter 4 Development

- SEI CERT C Coding Standard: https://wiki.sei.cmu.edu/confluence/display/c/SEI+CERT+C+Coding+Standard
- Vulnerability information and security advisories
- United States - Cisa: https://www.cisa.gov/uscert/ics/advisories
- NVD https://nvd.nist.gov/
- Japan - Jvndb : https://jvndb.jvn.jp/en/

9.3 Further references on Chapter 5 Verification and Testing

- Code review: https://en.wikipedia.org/wiki/Code_review
- OWASP Foundation: Component Analysis | OWASP Foundation
- OWASP Foundation: OWASP CycloneDX Software Bill of Materials (SBOM) Standard
- OWASP Foundation: Vulnerability Scanning Tools | OWASP Foundation
- CISA : Software Bill of Materials | CISA
- OWASP Foundation: Source Code Analysis Tools | OWASP Foundation
- Gartner: https://www.gartner.com/en/information-technology/glossary/static-application-security testing-sast
- OWASP Foundation: https://owasp.org/www-community/Fuzzing
- GitHub: GitHub - google/oss-fuzz: OSS-Fuzz - continuous fuzzing for open source software
- OWASP Foundation: https://owasp.org/www-project-web-security-testing-guide/latest/4 Web_Application_Security_Testing/
- OWASP Foundation: OWASP Top Ten | OWASP Foundation

9.4 Further references on Chapter 6 “Implementation, Deployment and Operations”

- NIST Framework for Improving Critical Infrastructure Cybersecurity, www.nist.gov
- The BSA Framework for Secure Software, www.bsa.org
- Securing Enterprise DevOps Environments, www.sogeti.com

9.5 Further References for Chapter 8 “Security by Default in Agile”

- OWASP DevSecOps Maturity Model: https://owasp.org/www-project-devsecops-maturity-model/
- OWASP DevSecOps Guideline: https://owasp.org/www-project-devsecops-guideline/